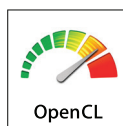


OpenCL (Open Computing Language) is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, GPUs, and other processors. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance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



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

OpenCL API Reference

The OpenCL Platform Layer

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

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

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

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

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

param_name: CL_PLATFORM_PROFILE, CL_PLATFORM_VERSION,
CL_PLATFORM_NAME, CL_PLATFORM_VENDOR, CL_PLATFORM_EXTENSIONS,
CL_PLATFORM_ICD_SUFFIX_KHR [Table 4.1]

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

device_type: [Table 4.2]
CL_DEVICE_TYPE_ACCELERATOR, ALL, CPU,
CL_DEVICE_TYPE_CUSTOM, DEFAULT, GPU

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

param_name: [Table 4.3]
CL_DEVICE_ADDRESS_BITS, CL_DEVICE_AVAILABLE,
CL_DEVICE_BUILT_IN_KERNELS,
CL_DEVICE_COMPILER_AVAILABLE,
CL_DEVICE_DOUBLE, CL_DEVICE_HALF_FP_CONFIG,
CL_DEVICE_ENDIAN_LITTLE, CL_DEVICE_EXTENSIONS,
CL_DEVICE_ERROR_CORRECTION_SUPPORT,
CL_DEVICE_EXECUTION_CAPABILITIES,
CL_DEVICE_GLOBAL_MEM_CACHE_SIZE, CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE, CL_DEVICE_GLOBAL_MEM_SIZE,
CL_DEVICE_GLOBAL_VARIABLE_PREFERRED_TOTAL_SIZE,
CL_DEVICE_PREFERRED_PLATFORM_LOCAL,
CL_DEVICE_PREFERRED_ATOMIC_ALIGNMENT,
CL_DEVICE_GLOBAL_VARIABLE_SHARING,
CL_DEVICE_HOST_UNIFIED_MEMORY,
CL_DEVICE_IMAGE_MAX_ARRAY_BUFFER_SIZE,
CL_DEVICE_IMAGE_SUPPORT,
CL_DEVICE_IMAGE2D_MAX_WIDTH_HEIGHT,
CL_DEVICE_IMAGE3D_MAX_WIDTH_HEIGHT_DEPTH,
CL_DEVICE_IMAGE_BASE_ADDRESS_ALIGNMENT,
CL_DEVICE_IMAGE_PITCH_ALIGNMENT,
CL_DEVICE_LINKER_AVAILABLE,
CL_DEVICE_LOCAL_MEM_TYPE_SIZE,
CL_DEVICE_MAX_READ_IMAGE_ARGS,
CL_DEVICE_MAX_WRITE_IMAGE_ARGS,
CL_DEVICE_MAX_CLOCK_FREQUENCY_PIPE_ARGS,
CL_DEVICE_MAX_COMPUTE_UNITS_SAMPLERS,
CL_DEVICE_MAX_CONSTANT_ARGS_BUFFER_SIZE,
CL_DEVICE_MAX_MEM_ALLOC_PARAMETER_SIZE,
CL_DEVICE_MAX_GLOBAL_VARIABLE_SIZE,
CL_DEVICE_MAX_ON_DEVICE_QUEUE_EVENTS,
CL_DEVICE_MAX_WORK_GROUP_SIZE,
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS_SIZES,
CL_DEVICE_MEM_BASE_ADDR_ALIGN,
CL_DEVICE_NAME,
CL_DEVICE_NATIVE_VECTOR_WIDTH_CHAR_INT,
CL_DEVICE_NATIVE_VECTOR_WIDTH_LONG_SHORT,
CL_DEVICE_NATIVE_VECTOR_WIDTH_DOUBLE_HALF,
CL_DEVICE_NATIVE_VECTOR_WIDTH_FLOAT,
CL_DEVICE_OPENCL_C_VERSION_PARENT_DEVICE,
CL_DEVICE_PARTITION_AFFINITY_DOMAIN,
CL_DEVICE_PARTITION_MAX_SUB_DEVICES,
CL_DEVICE_PARTITION_PROPERTIES_TYPE,
CL_DEVICE_PIPE_MAX_ACTIVE_RESERVATIONS,
CL_DEVICE_PIPE_MAX_PACKET_SIZE,

CL_DEVICE_PLATFORM_PRINTF_BUFFER_SIZE,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR_INT,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_HALF,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT,
CL_DEVICE_PREFERRED_INTEROP_USER_SYNC,
CL_DEVICE_PROFILE,
CL_DEVICE_PROFILING_TIMER_RESOLUTION,
CL_DEVICE_SPIR_VERSIONS,
CL_DEVICE_QUEUE_ON_DEVICE_PROPERTIES,
CL_DEVICE_QUEUE_ON_HOST_PROPERTIES,
CL_DEVICE_QUEUE_ON_DEVICE_MAX_SIZE,
CL_DEVICE_QUEUE_ON_DEVICE_PREFERRED_SIZE,
CL_DEVICE_REFERENCE_COUNT_VENDOR_ID,
CL_DEVICE_SVM_CAPABILITIES,
CL_DEVICE_TERMINATE_CAPABILITY_KHR,
CL_DEVICE_TYPE_VENDOR,
CL_DEVICE_DRIVER_VERSION

Partitioning a Device [4.3]

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

properties: CL_DEVICE_PARTITION_EQUALLY,
CL_DEVICE_PARTITION_BY_COUNTS,
CL_DEVICE_PARTITION_BY_AFFINITY_DOMAIN

`cl_int clRetainDevice (cl_device_id device)`

`cl_int clReleaseDevice (cl_device_id device)`

Contexts [4.4]

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

The OpenCL Runtime

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

Command Queues [5.1]

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

properties: [Table 5.1] CL_QUEUE_SIZE,
CL_QUEUE_PROPERTIES (bitfield which may be set to an OR of CL_QUEUE_* where * may be: OUT_OF_ORDER_EXEC_MODE_ENABLE, PROFILING_ENABLE, ON_DEVICE_DEFAULT)

`cl_int clRetainCommandQueue (
cl_command_queue command_queue)`

`cl_int clReleaseCommandQueue (
cl_command_queue command_queue)`

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

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

properties: [Table 4.5]

NULL or CL_CONTEXT_PLATFORM,
CL_CONTEXT_INTEROP_USER_SYNC,
CL_CONTEXT_D3D10_D3D11_DEVICE_KHR,
CL_CONTEXT_ADAPTER_D3D9_D3D9EX_KHR,
CL_CONTEXT_ADAPTER_DXVA_KHR,
CL_CONTEXT_MEMORY_INITIALIZE_KHR,
CL_CONTEXT_TERMINATE_KHR,
CL_GL_CONTEXT_KHR, CL_CGL_SHAREGROUP_KHR,
CL_EGL_GLX_DISPLAY_KHR, CL_WGL_HDC_KHR

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

properties: See clCreateContext

device_type: See clGetDeviceIDs

`cl_int clRetainContext (cl_context context)`

`cl_int clReleaseContext (cl_context context)`

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

param_name: CL_CONTEXT_REFERENCE_COUNT,
CL_CONTEXT_DEVICES, CL_CONTEXT_PROPERTIES,
CL_CONTEXT_D3D10_D3D11_PREFER_SHARED_RESOURCES_KHR [Table 4.6]

`cl_int clTerminateContextKHR (cl_context context)`

Get CL Extension Function Pointers [9.2]

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

Buffer Objects

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

Create Buffer Objects [5.2.1]

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

flags: [Table 5.3] CL_MEM_READ_WRITE,
CL_MEM_WRITE_READ_ONLY,
CL_MEM_HOST_NO_ACCESS,
CL_MEM_HOST_READ_WRITE_ONLY,
CL_MEM_USE_ALLOC_COPY, CL_MEM_HOST_PTR

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

flags: See clCreateBuffer

buffer_create_type: CL_BUFFER_CREATE_TYPE_REGION

Read, Write, Copy Buffer Objects [5.2.2]

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

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

(Continued on next page >)

Buffer Objects (continued)

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

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

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

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

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

Map Buffer Objects [5.2.4]

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

map_flags: CL_MAP_READ, CL_MAP_WRITE, CL_MAP_WRITE_INVALIDATE_REGION

Conversions and Type Casting Examples [6.2]

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

Memory Objects

A memory object is a handle to a reference counted region of global memory. Includes Buffer Objects, Image Objects, and Pipe Objects. Items in blue apply when the appropriate extension is supported.

Memory Objects [5.5.1, 5.5.2]

```
cl_int clRetainMemObject (cl_mem memobj)
```

```
cl_int clReleaseMemObject (cl_mem memobj)
```

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

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

Migrate Memory Objects [5.5.4]

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

flags: CL_MIGRATE_MEM_OBJECT_HOST,
CL_MIGRATE_MEM_OBJECT_CONTENT_UNDEFINED

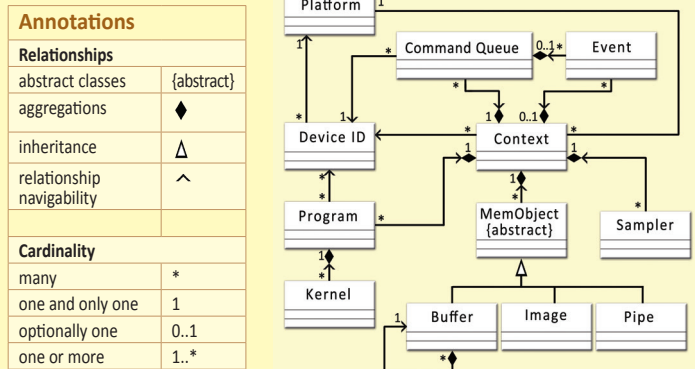
Query Memory Object [5.5.5]

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

param_name: CL_MEM_TYPE, CL_MEM_FLAGS, CL_MEM_SIZE, CL_MEM_HOST_PTR, CL_MEM_OFFSET,
CL_MEM_MAP_REFERENCE_COUNT, CL_MEM_ASSOCIATED_MEMOBJECT,
CL_MEM_CONTEXT, CL_MEM_USES_SVM_POINTER,
CL_MEM_D3D10_RESOURCE_KHR,
CL_MEM_DX9_MEDIA_ADAPTER_TYPE, CL_MEM_SURFACE_INFO_KHR [Table 5.12]

OpenCL Class Diagram

The figure below describes the OpenCL specification as a class diagram using the Unified Modeling Language¹ (UML) notation. The diagram shows both nodes and edges which are classes and their relationships. As a simplification it shows only classes, and no attributes or operations.



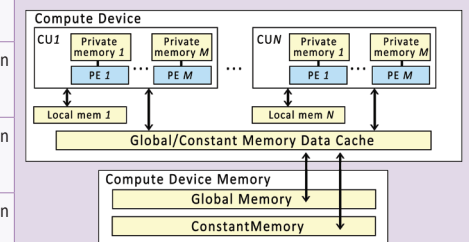
¹ Unified Modeling Language (<http://www.uml.org/>) is a trademark of Object Management Group (OMG).

OpenCL Device Architecture Diagram

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

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

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



Pipes

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

Create Pipe Objects [5.4.1]

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

flags:

0 or CL_MEM_READ_WRITE, CL_MEM_READ_WRITE_ONLY,
CL_MEM_HOST_NO_ACCESS

Pipe Object Queries [5.4.2]

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

param_name:

CL_PIPE_PACKET_SIZE, CL_PIPE_MAX_PACKETS

Shared Virtual Memory

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

SVM Sharing Granularity [5.6.1]

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

flags: [Table 5.13]

CL_MEM_READ_WRITE, CL_MEM_READ_WRITE_ONLY,
CL_MEM_SVM_FINE_GRAIN_BUFFER, CL_MEM_SVM_ATOMICS

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

Enqueuing SVM Operations [5.6.2]

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

(Continued on next page >)

Shared Virtual Memory (continued)

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

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

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

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

Kernel Objects

A kernel is a function declared in a program, identified by the `__kernel` qualifier. A kernel object encapsulates the specific `__kernel` function and the argument values to be used when executing it. Items in blue apply when the appropriate extension is supported.

Create Kernel Objects [5.9.1]

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

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

```
cl_int clRetainKernel (cl_kernel kernel)
cl_int clReleaseKernel (cl_kernel kernel)
```

Kernel Arguments and Queries [5.9.2, 5.9.3]

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

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

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

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

param_name: [Table 5.19]
 CL_KERNEL_FUNCTION_NAME,
 CL_KERNEL_NUM_ARGS,
 CL_KERNEL_REFERENCE_COUNT,
 CL_KERNEL_ATTRIBUTES, CONTEXT, PROGRAM}

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

param_name: CL_KERNEL_GLOBAL_WORK_SIZE,
 CL_KERNEL_COMPILE_WORK_GROUP_SIZE,
 CL_KERNEL_LOCAL_PRIVATE_MEM_SIZE,
 CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_MULTIPLE [Table 5.20]

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

param_name:
 CL_KERNEL_ARG_ACCESS_ADDRESS_QUALIFIER,
 CL_KERNEL_ARG_NAME,
 CL_KERNEL_ARG_TYPE {NAME, QUALIFIER} [Table 5.21]

Program Objects

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

Create Program Objects [5.8.1]

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

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

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

```
cl_int clRetainProgram (cl_program program)
cl_int clReleaseProgram (cl_program program)
```

Building Program Executables [5.8.2]

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

Separate Compilation and Linking [5.8.3]

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

cl_int clGetKernelSubGroupInfoKHR

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

param_name:
 CL_KERNEL_MAX_SUB_GROUP_SIZE_FOR_NDRANGE,
 CL_KERNEL_SUB_GROUP_COUNT_FOR_NDRANGE

Execute Kernels [5.10]

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

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

Flush and Finish [5.15]

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

Event Objects

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

Event Objects [5.11]

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

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

Unload the OpenCL Compiler [5.8.6]

```
cl_int clUnloadPlatformCompiler (
    cl_platform_id platform)
```

Query Program Objects [5.8.7]

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

param_name: [Table 5.16]
 CL_PROGRAM_REFERENCE_COUNT,
 CL_PROGRAM_CONTEXT, NUM_DEVICES, DEVICES},
 CL_PROGRAM_SOURCE, BINARY_SIZES, BINARIES},
 CL_PROGRAM_NUM_KERNELS, KERNEL_NAMES}

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

param_name: [Table 5.17]
 CL_PROGRAM_BINARY_TYPE,
 CL_PROGRAM_BUILD_STATUS, OPTIONS, LOG},
 CL_PROGRAM_BUILD_GLOBAL_VARIABLE_TOTAL_SIZE

Compiler Options [5.8.4]

SPIR options require the `cl_khr_spir` extension.

Preprocessor: (-D processed in order for `clBuildProgram` or `clCompileProgram`)

-D name -D name=definition -I dir

Math intrinsics:

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

Optimization options:

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

Warning request/suppress:

-w -Werror

Control OpenCL C language version:

-cl-std=CL1.1 // OpenCL 1.1 specification
 -cl-std=CL1.2 // OpenCL 1.2 specification
 -cl-std=CL2.0 // OpenCL 2.0 specification

Query kernel argument information:

-cl-kernel-arg-info

Debugging options:

-g // generate additional errors for built-in
 // functions that allow you to enqueue
 // commands on a device

SPIR binary options:

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

Linker Options [5.8.5]

Library linking options:

-create-library -enable-link-options

Program linking options:

-cl-denorms-are-zero -cl-no-signed-zeroes
 -cl-finite-math-only -cl-fast-relaxed-math
 -cl-unsafe-math-optimizations

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

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

```
cl_int clGetEventInfo (cl_event event,
    cl_event_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
param_name: CL_EVENT_COMMAND {QUEUE, TYPE},
             CL_EVENT_CONTEXT, REFERENCE_COUNT},
             CL_EVENT_COMMAND_EXECUTION_STATUS [Table 5.22]
```

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

(Continued on next page >)

Event Objects (continued)

`cl_int clReleaseEvent (cl_event event)`

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

Markers, Barriers, Waiting for Events [5.12]

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

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

Profiling Operations [5.14]

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

param_name: [Table 5.23]
CL_PROFILING_COMMAND_QUEUED, CL_PROFILING_
COMMAND_COMPLETE,
CL_PROFILING_COMMAND_{SUBMIT, START, END}

OpenCL C Language Reference

Supported Data Types

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

Built-in Scalar Data Types [6.1.1]

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

Built-in Vector Data Types [6.1.2]

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

Other Built-in Data Types [6.1.3]

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

OpenCL Type	Description
image2d_ <i>[msaa_]</i> t	OPTIONAL 2D image handle
image3d_t	OPTIONAL 3D image handle
image2d_array_ <i>[msaa_]</i> t	OPTIONAL 2D image array
image1d_t	OPTIONAL 1D image handle
image1d_buffer_t	OPTIONAL 1D image buffer

image1d_array_t	OPTIONAL	1D image array
image2d_ <i>[msaa_]</i> depth_t	OPTIONAL	2D depth image
image2d_array_ <i>[msaa_]</i> depth_t	OPTIONAL	2D depth image array
sampler_t	OPTIONAL	sampler handle
queue_t		
ndrange_t		
clk_event_t		
reserve_id_t		
event_t		event handle
cl_mem_fence_flags		

Reserved Data Types [6.1.4]

OpenCL Type	Description
bool _n	boolean vector
half _n	16-bit, vector
quad, quad _n	128-bit float, vector
complex half, complex half _n imaginary half, imaginary half _n	16-bit complex, vector
complex float, complex float _n imaginary float, imaginary float _n	32-bit complex, vector
complex double, complex double _n imaginary double, imaginary double _n	64-bit complex, vector
complex quad, complex quad _n imaginary quad, imaginary quad _n	128-bit complex, vector
float _n xm	<i>n</i> *m matrix of 32-bit floats
double _n xm	<i>n</i> *m matrix of 64-bit floats

Vector Component Addressing [6.1.7]

Vector Components

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

Vector Addressing Equivalences

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

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

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

Operators and Qualifiers

Operators [6.3]

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

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

Address Space Qualifiers [6.5]

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

Function Qualifiers [6.7]

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

Preprocessor Directives & Macros [6.10]

#pragma OPENCL FP_CONTRACT	on-off-switch on-off-switch: ON, OFF, DEFAULT
__FILE__	Current source file
__func__	Current function name
__LINE__	Integer line number
__OPENCL_VERSION__	Integer version number, e.g: 200
__CL_VERSION_1_0	Substitutes integer 100 for 1.0
__CL_VERSION_1_1	Substitutes integer 110 for 1.1
__CL_VERSION_1_2	Substitutes integer 120 for 1.2
__CL_VERSION_2_0	Substitutes integer 200 for 2.0
__OPENCL_C_VERSION__	Sub. integer for OpenCL C version
__ENDIAN_LITTLE__	1 if device is little endian
__IMAGE_SUPPORT__	1 if images are supported
__FAST_RELAXED_MATH__	1 if -cl-fast-relaxed-math optimization option is specified
__CL_DEVICE_MAX_GLOBAL_VARIABLE_SIZE__	Max size in bytes for a program scope or static function variable
FP_FAST_FMA	Defined if double fma is fast
FP_FAST_FMAF	Defined if float fma is fast
FP_FAST_FMA_HALF	Defined if half fma is fast
__kernel_exec(X, typen)	Same as:
__kernel __attribute__((work_group_size_hint(X, 1, 1)))	
__attribute__((vec_type_hint(typen)))	

Work-Item Built-in Functions [6.13.1]

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

uint get_work_dim ()	Number of dimensions in use
size_t get_global_size (uint dimindx)	Number of global work-items
size_t get_global_id (uint dimindx)	Global work-item ID value
size_t get_local_size (uint dimindx)	Number of local work-items if kernel executed with uniform work-group size

(Continued on next page >)

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 Functions (continued)

<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]

T_s is type float, optionally double, or half if the `cl_khr_fp16` extension is enabled. T_n is the vector form of T_s , where n is 2, 3, 4, 8, or 16. T is T_s and T_n .

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

<code>T acos (T)</code>	Arc cosine
<code>T acosh (T)</code>	Inverse hyperbolic cosine
<code>T acospi (T x)</code>	$\arccos(x) / \pi$
<code>T asin (T)</code>	Arc sine
<code>T asinh (T)</code>	Inverse hyperbolic sine
<code>T asinpi (T x)</code>	$\arcsin(x) / \pi$
<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>	$\arctan(x) / \pi$
<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>	x / y (T may only be float or floatn)
<code>T native_divide (T x, T y)</code>	
<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>	Return y if $x < y$, otherwise it returns x
<code>Tn fmax (Tn x, Tn y)</code>	
<code>T fmin (T x, T y)</code>	Return y if $y < x$, otherwise it returns x
<code>Tn fmin (Tn x, Tn y)</code>	
<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>	Extract mantissa and exponent
<code>Tn frexp (T x, intn *exp)</code>	
<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>	$x * 2^n$
<code>Tn ldexp (T x, intn n)</code>	
<code>T lgamma (T x)</code>	Log gamma function
<code>Ts lgamma_r (T x, int *signp)</code>	
<code>Tn lgamma_r (Tn x, intn *signp)</code>	
<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
<code>half[n] nan (ushort[n] nancode)</code>	(Return is scalar when <i>nancode</i> is scalar)
<code>double[n] nan (ulong[n] nancode)</code>	

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 must appear immediately before the loop to be affected.)

```
__attribute__((opengl_unroll_hint(n)))
__attribute__((opengl_unroll_hint))
```

<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 pown (T x, int y)</code>	Compute x^y , where y is an integer
<code>Tn pown (T x, intn y)</code>	
<code>T powr (T x, T y)</code> HN	Compute x^y , where x is ≥ 0
<code>T half_recip (T x)</code>	$1 / x$ (T may only be float or floatn)
<code>T native_recip (T x)</code>	
<code>T remainder (T x, T y)</code>	Floating point remainder
<code>Ts remquo (T x, T y, int *quo)</code>	Remainder and quotient
<code>Tn remquo (Tn x, Tn y, intn *quo)</code>	
<code>T rint (T)</code>	Round to nearest even integer
<code>Ts rootn (T x, int y)</code>	Compute x to the power of $1/y$
<code>Tn rootn (T x, intn y)</code>	
<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
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_2
<code>M_LN10_F</code>	Value of \log_{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, ucharn, short, shortn, ushort, ushortn, int, intn, uint, uintn, long, longn, ulong, or ulongn, where n is 2, 3, 4, 8, or 16. T_u is the unsigned version of T . T_{sc} is the scalar version of T .

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

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

<code>short[n] upsample (char[n] hi, uchar[n] lo)</code>	$\text{result}[i] = ((\text{short})hi[i] << 8) lo[i]$
<code>ushort[n] upsample (uchar[n] hi, uchar[n] lo)</code>	$\text{result}[i] = ((\text{ushort})hi[i] << 8) lo[i]$

<code>int[n] upsample (short[n] hi, ushort[n] lo)</code>	$\text{result}[i] = ((\text{int})hi[i] << 16) lo[i]$
<code>uint[n] upsample (ushort[n] hi, ushort[n] lo)</code>	$\text{result}[i] = ((\text{uint})hi[i] << 16) lo[i]$
<code>long[n] upsample (int[n] hi, uint[n] lo)</code>	$\text{result}[i] = ((\text{long})hi[i] << 32) lo[i]$
<code>ulong[n] upsample (uint[n] hi, uint[n] lo)</code>	$\text{result}[i] = ((\text{ulong})hi[i] << 32) lo[i]$

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

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

Common Built-in Functions [6.13.4] [9.4.3]

These functions operate component-wise and use round to nearest even rounding mode. T_s is type float, optionally double, or half if `cl_khr_fp16` is enabled. T_n is the vector form of T_s , where n is 2, 3, 4, 8, or 16. T is T_s and T_n .

<code>T clamp (T x, T min, T max)</code>	Clamp x to range given by <i>min</i> , <i>max</i>
<code>Tn clamp (Tn x, Tn min, Tn max)</code>	
<code>T degrees (T radians)</code>	radians to degrees
<code>T max (T x, T y)</code>	Max of x and y
<code>Tn max (Tn x, Tn y)</code>	
<code>T min (T x, T y)</code>	Min of x and y
<code>Tn min (Tn x, Tn y)</code>	

(Continued on next page >)

Common Functions (continued)

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

Geometric Built-in Functions [6.13.5] [9.4.4]

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

float{3,4} cross (float{3,4} <i>p0</i> , float{3,4} <i>p1</i>)	Cross product
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>)	Vector distance
<i>Ts distance</i> (<i>T p0</i> , <i>T p1</i>)	
<i>Ts dot</i> (<i>T p0</i> , <i>T p1</i>)	Dot product

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

Relational Built-in Functions [6.13.6]

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

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

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

int isnan (double)	Test for a NaN
longn isnan (doublen)	
int isnan (half)	
shortn isnan (halfn)	
int isnormal (float)	Test for a normal value
floatn isnormal (floatn)	
int isnormal (double)	
longn isnormal (doublen)	
int isnormal (half)	Test for a normal value
shortn isnormal (halfn)	
int isordered (float <i>x</i> , float <i>y</i>)	
floatn isordered (floatn <i>x</i> , floatn <i>y</i>)	
int isordered (double <i>x</i> , double <i>y</i>)	Test if arguments are ordered
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>)	
int isunordered (float <i>x</i> , float <i>y</i>)	Test if arguments are unordered
floatn 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>)	Test for sign bit
shortn isunordered (halfn <i>x</i> , halfn <i>y</i>)	
int signbit (float)	
floatn signbit (floatn)	
int signbit (double)	Test for sign bit
longn signbit (doublen)	
int signbit (half)	
shortn signbit (halfn)	
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>)	Each bit of result is corresponding bit of <i>a</i> if corresponding bit of <i>c</i> is 0
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>)	
<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>)	For each component of a vector type, result[i] = if MSB of <i>c</i> [i] is set ? <i>b</i> [i] : <i>a</i> [i]. For scalar type, result = <i>c</i> ? <i>b</i> : <i>a</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>)	

Vector Data Load/Store [6.13.7] [9.4.6]

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

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

void vstore_half_R (double <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>)
void vstore_halfn (floatn <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>))
void vstore_halfn_R (floatn <i>data</i> , <i>size_t</i> <i>offset</i> , half * <i>p</i>)	
void vstore_halfn (doublen <i>data</i> , <i>size_t</i> <i>offset</i> , half * <i>p</i>)	
void vstore_halfn_R (doublen <i>data</i> , <i>size_t</i> <i>offset</i> , half * <i>p</i>)	
floatn vloada_halfn (<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)).
void vstorea_halfn (floatn <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)).
void vstorea_halfn_R (floatn <i>data</i> , <i>size_t</i> <i>offset</i> , half * <i>p</i>)	
void vstorea_halfn (doublen <i>data</i> , <i>size_t</i> <i>offset</i> , half * <i>p</i>)	
void vstorea_halfn_R (doublen <i>data</i> , <i>size_t</i> <i>offset</i> , half * <i>p</i>)	

Synchronization & Memory Fence Functions [6.13.8]

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

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

Async Copies and Prefetch [6.13.10] [9.4.7]

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

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

Atomic Functions [6.13.11]

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

Atomic Functions

In the following definitions, **A** refers to one of the atomic_* types. **C** refers to its corresponding non-atomic type. **M** refers to the type of the other argument for arithmetic operations. For atomic integer types, **M** is **C**. For atomic pointer types, **M** is ptrdiff_t. The type atomic_* is a 32-bit integer. **atomic_long** and **atomic_ulong** require extension **cl_khr_int64_base_atomics** or **cl_khr_int64_extended_atomics**. The atomic_double type requires double precision support. The default scope is work_group for local atomics and all_svm_devices for global atomics.

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

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

Atomic Types and Enum Constants

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

Atomic integer and floating-point types

atomic_int atomic_uint	atomic_long atomic_ulong	atomic_float atomic_double	atomic_intptr_t atomic_uintptr_t	atomic_size_t atomic_ptrdiff_t
---------------------------	-----------------------------	-------------------------------	-------------------------------------	-----------------------------------

Atomic Macros

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

64-bit Atomics [9.3]

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

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

Address Space Qualifier Functions [6.13.9]

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

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

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. Subgroups require the **cl_khr_subgroups** extension. Double and vector types require double precision support.

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

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

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

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

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

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

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

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

Pipe Built-in Functions [6.13.16.2-4]

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

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

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

Miscellaneous Vector Functions [6.13.12]

Tm and Tn are type `charn`, `ucharn`, `shortn`, `ushortn`, `intn`, `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 `ucharn`, `ushortn`, `uintn`, or `ulongn`.

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

Event Built-in Functions [6.13.17.8]

T is type `int`, `uint`, `long`, `ulong`, or `float`, optionally `double`, or half if the `cl_khr_fp16` extension is enabled.

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

Helper Built-in Functions [6.13.17.9]

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

OpenCL Image Processing Reference

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

Image Objects

Items in blue apply when the appropriate extension is supported.

Create Image Objects [5.3.1]

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

$flags$: See `clCreateBuffer`

Query List of Supported Image Formats [5.3.2]

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

$flags$: See `clCreateBuffer`

$image_type$: `CL_MEM_OBJECT_IMAGE1D`, `CL_MEM_OBJECT_IMAGE1D_BUFFER`, `CL_MEM_OBJECT_IMAGE1D_ARRAY`

Read, Write, Copy, Fill Image Objects [5.3.4]

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

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

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

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

Copy Between Image, Buffer Objects [5.3.5]

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

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

(Continued on next page >)

Image Objects (continued)

Map and Unmap Image Objects [5.3.6]

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

map_flags: CL_MAP_READ {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 5.9] CL_IMAGE {FORMAT, BUFFER},
            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
```

Also see `clGetMemObjectInfo` [5.4.5]

Image Formats [5.3.1.1]

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

Built-in support: [Table 5.8]

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

CL_DEPTH (read + write): CL_FLOAT, CL_UNORM_INT16

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

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

CL_RGBA (read + 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 + write): CL_UNORM_INT8

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

Optional support: [Table 5.6]

```
CL_R, CL_A: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_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_INT101010

CL_ARGB: CL_UNORM_INT8, CL_SIGNED_INT8,
CL_UNSIGNED_INT8, CL_SNORM_INT8

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

Image Read and Write Functions [6.13.14]

The built-in functions defined in this section can only be used with image memory objects created with `clCreateImage`. *sampler* specifies the addressing and filtering mode to use. Writing to sRGB images from a kernel requires the `cl_khr_srgb_image_writes` extension. `read_imageh` and `write_imageh` require the `cl_khr_fp16` extension. MSAA images require the `cl_khr_gl_msaa_sharing` extension, and image 3D writes require the extension `cl_khr_3d_image_writes`.

Read and write functions for 1D images

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

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

float4 read_imagef (image1d_t image, int coord)

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

float4 read_imagef (image1d_array_t image, int2 coord)

float4 read_imagef (image1d_buffer_t image, int coord)

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

int4 read_imagei (image1d_t image, int coord)

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

int4 read_imagei (image1d_array_t image, int2 coord)

int4 read_imagei (image1d_buffer_t image, int coord)

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

uint4 read_imageui (image1d_t image, int coord)

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

uint4 read_imageui (image1d_array_t image, int2 coord)

uint4 read_imageui (image1d_buffer_t image, int coord)

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

half4 read_imageh (image1d_t image, int coord)

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

half4 read_imageh (image1d_array_t image, int2 coord)

half4 read_imageh (image1d_buffer_t image, int coord)

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

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

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

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

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

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

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

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

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

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

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

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

Read and write functions for 2D images

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

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

float4 read_imagef (image2d_t image, int2 coord)

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

float4 read_imagef (image2d_array_t image, int4 coord)

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

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

float read_imagef (image2d_depth_t image, int2 coord)

float read_imagef (image2d_array_depth_t image, int4 coord)

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

int4 read_imagei (image2d_t image, int2 coord)

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

int4 read_imagei (image2d_array_t image, int4 coord)

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

uint4 read_imageui (image2d_t image, int2 coord)

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

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

Read and write functions for 2D images (continued)

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

half4 read_imageh (image2d_t image, int2 coord)

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

half4 read_imageh (image2d_array_t image, int4 coord)

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

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

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

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

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

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

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

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

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

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

Read and write functions for 3D images

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

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

float4 read_imagef (image3d_t image, int4 coord)

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

int4 read_imagei (image3d_t image, int4 coord)

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

uint4 read_imageui (image3d_t image, int4 coord)

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

half4 read_imageh (image3d_t image, int4 coord)

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

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

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

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

(Continued on next page >)

Image Read and Write (continued)

Extended mipmap read and write functions [9.18.2.1]

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

<code>float read_imagef (image2d_ [depth_]t image, sampler_t sampler, float2 coord, float lod)</code>
<code>int4 read_imagei (image2d_t image, sampler_t sampler, float2 coord, float lod)</code>
<code>uint4 read_imageui (image2d_t image, sampler_t sampler, float2 coord, float lod)</code>
<code>float read_imagef (image2d_ [depth_]t image, sampler_t sampler, float2 coord, float2 gradient_x, float2 gradient_y)</code>
<code>int4 read_imagei (image2d_t image, sampler_t sampler, float2 coord, float2 gradient_x, float2 gradient_y)</code>
<code>uint4 read_imageui (image2d_t image, sampler_t sampler, float2 coord, float2 gradient_x, float2 gradient_y)</code>
<code>float4 read_imagef (image1d_t image, sampler_t sampler, float coord, float lod)</code>
<code>int4 read_imagei (image1d_t image, sampler_t sampler, float coord, float lod)</code>
<code>uint4 read_imageui (image1d_t image, sampler_t sampler, float coord, float lod)</code>
<code>float4 read_imagef (image1d_t image, sampler_t sampler, float coord, float gradient_x, float gradient_y)</code>
<code>int4 read_imagei (image1d_t image, sampler_t sampler, float coord, float gradient_x, float gradient_y)</code>
<code>uint4 read_imageui (image1d_t image, sampler_t sampler, float coord, float gradient_x, float gradient_y)</code>
<code>float4 read_imagef (image3d_t image, sampler_t sampler, float4 coord, float lod)</code>
<code>int4 read_imagei (image3d_t image, sampler_t sampler, float4 coord, float lod)</code>
<code>uint4 read_imageui (image3d_t image, sampler_t sampler, float4 coord, float lod)</code>
<code>float4 read_imagef (image3d_t image, sampler_t sampler, float4 coord, float4 gradient_x, float4 gradient_y)</code>
<code>int4 read_imagei (image3d_t image, sampler_t sampler, float4 coord, float4 gradient_x, float4 gradient_y)</code>
<code>uint4 read_imageui (image3d_t image, sampler_t sampler, float4 coord, float4 gradient_x, float4 gradient_y)</code>
<code>float4 read_imagef (image1d_array_t image, sampler_t sampler, float2 coord, float lod)</code>
<code>int4 read_imagei (image1d_array_t image, sampler_t sampler, float2 coord, float lod)</code>
<code>uint4 read_imageui (image1d_array_t image, sampler_t sampler, float2 coord, float lod)</code>
<code>float4 read_imagef (image1d_array_t image, sampler_t sampler, float2 coord, float gradient_x, float gradient_y)</code>

Sampler Objects [5.7]

Items in blue require the `cl_khr_mipmap_image` extension.

`cl_sampler` `clCreateSamplerWithProperties`

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

sampler_properties: [Table 5.14]

```
CL_SAMPLER_NORMALIZED_COORDS,
CL_SAMPLER_ADDRESSING_MODE,
CL_SAMPLER_MIP_FILTER_MODE,
CL_SAMPLER_LOD_MIN, CL_SAMPLER_LOD_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 5.15]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Extended multi-sample image read functions [9.12.3]

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

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

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

```
float4 read_imagef (f, i, ui) (image2d_msaa_t image,
  int2 coord, int sample)
```

```
float4 read_imagef (f, i, ui) (image2d_array_msaa_t image,
  int4 coord, int sample)
```

Sampler Declaration Fields [6.13.14.1]

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

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

normalized-mode:

```
CLK_NORMALIZED_COORDS {TRUE, FALSE}
```

address-mode:

```
CLK_ADDRESS {REPEAT, CLAMP, NONE},
CLK_ADDRESS {CLAMP_TO_EDGE},
CLK_ADDRESS {MIRRORED_REPEAT}
```

filter-mode: CLK_FILTER_NEAREST, CLK_FILTER_LINEAR

Image Query Functions [6.13.14.5] [9.12]

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

Query image width, height, and depth in pixels

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

```
int get_image_width (image1d_buffer_t image)
```

```
int get_image_width (image {1,2}d_array_t image)
```

```
int get_image_width (image2d_array_ [depth_]t image)
```

```
int get_image_width (image2d_array_ [msaa]_t image)
```

```
int get_image_width (image2d_array_ [msaa]_depth_t
  image)
```

```
int get_image_height (image {2,3}d_t image)
```

```
int get_image_height (image2d_array_t image)
```

```
int get_image_height (image2d_array_ [depth_]t image)
```

```
int get_image_height (image2d_array_ [msaa]_t image)
```

```
int get_image_height (image2d_array_ [msaa]_depth_t
  image)
```

```
int get_image_depth (image3d_t image)
```

Query image array size

```
size_t get_image_array_size (image1d_array_t image)
```

```
size_t get_image_array_size (image2d_array_t image)
```

```
size_t get_image_array_size (image2d_array_depth_t image)
```

```
size_t get_image_array_size (
  image2d_array_msaa_depth_t image)
```

Query image dimensions

```
int2 get_image_dim (image2d_t image)
```

```
int2 get_image_dim (image2d_array_t image)
```

```
int4 get_image_dim (image3d_t image)
```

```
int2 get_image_dim (image2d_array_ [depth_]t image)
```

```
int2 get_image_dim (image2d_array_ [msaa]_t image)
```

```
int2 get_image_dim (image2d_array_ [msaa]_depth_t
  image)
```

Query image Channel data type and order

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

```
int get_image_channel_data_type (image1d_buffer_t image)
```

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

```
int get_image_channel_data_type
  (image2d_array_ [depth_]t image)
```

```
int get_image_channel_data_type (
  image2d_array_ [msaa]_t image)
```

```
int get_image_channel_data_type (
  image2d_array_ [msaa]_depth_t image)
```

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

```
int get_image_channel_order (image1d_buffer_t image)
```

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

```
int get_image_channel_order
  (image2d_array_ [depth_]t image)
```

```
int get_image_channel_order (image2d_array_ [msaa]_t image)
```

```
int get_image_channel_order (
  image2d_array_ [msaa]_depth_t image)
```

Extended query functions [9.18.2.1]

These functions require the `cl_khr_mipmap_image` extension.

```
int get_image_num_mip_levels (image1d_t image)
```

```
int get_image_num_mip_levels (image2d_ [depth_]t image)
```

```
int get_image_num_mip_levels (image3d_t image)
```

```
int get_image_num_mip_levels (image1d_array_t image)
```

```
int get_image_num_mip_levels (
  image2d_array_ [depth_]t image)
```

```
int get_image_num_samples (
  image2d_array_ [msaa]_t image)
```

```
int get_image_num_samples (
  image2d_array_ [msaa]_depth_t image)
```

Access Qualifiers [6.6]

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

```
__read_only, read_only
__write_only, write_only
```

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

See www.khronos.org/registry/cl/

OpenCL Extensions Reference

Using OpenCL Extensions [9]

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

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

To test if an extension is supported:

`clGetPlatformInfo()` or `clGetDeviceInfo()`

To get the address of the extension function:

`clGetExtensionFunctionAddressForPlatform()`

`cl_apple_gl_sharing` (see `cl_khr_gl_sharing`)

`cl_khr_3d_image_writes`

`cl_khr_byte_addressable_store`

OpenGL Sharing [9.5 - 9.7]

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

CL Context > GL Context, Sharegroup [9.5.5]

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

param_name: CL_DEVICES (FOR_GL_CONTEXT_KHR, CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR)

CL Buffer Objects > GL Buffer Objects [9.6.2]

```
cl_mem clCreateFromGLBuffer (cl_context context,
    cl_mem_flags flags, GLuint bufobj, cl_int *errcode_ret)
flags: CL_MEM_READ_ONLY, WRITE_ONLY, READ_WRITE
```

CL Image Objects > GL Textures [9.6.3]

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

flags: See `clCreateFromGLBuffer`

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

DX9 Media Surface Sharing [9.9]

These functions require the extension

`cl_khr_dx9_media_sharing`. The associated header file is `cl_dx9_media_sharing.h`.

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

media_adapter_type: CL_ADAPTER_{D3D9, D3D9EX, DXVA}_KHR

media_adapter_set: CL_{ALL, PREFERRED}_DEVICES_FOR_DX9_MEDIA_ADAPTER_KHR

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

flags: See `clCreateFromGLBuffer`

adapter_type: CL_ADAPTER_{D3D9, D3D9EX, DXVA}_KHR

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

EGL Interoperability [9.19, 9.20]

Create CL Image Objects from EGL [9.19]

These functions require the extension `cl_khr_egl_image`.

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

```
cl_khr_context_abort
cl_khr_d3d10_sharing
cl_khr_d3d11_sharing
cl_khr_depth_images
cl_khr_dx9_media_sharing
cl_khr_egl_event
cl_khr_egl_image
cl_khr_fp16
cl_khr_fp64
cl_khr_gl_depth_images
cl_khr_gl_event
cl_khr_gl_msaa_sharing
cl_khr_gl_sharing
cl_khr_global_int32_base_atomics - atomic_*
```

```
cl_khr_global_int32_extended_atomics - atomic_*( )
cl_khr_icd
cl_khr_image2d_from_buffer
cl_khr_initialize_memory
cl_khr_int64_base_atomics - atom_*( )
cl_khr_int64_extended_atomics - atom_*( )
cl_khr_local_int32_base_atomics - atomic_*( )
cl_khr_local_int32_extended_atomics - atomic_*( )
cl_khr_mipmap_image
cl_khr_mipmap_image_writes
cl_khr_srgb_image_writes
cl_khr_spir
cl_khr_subgroups
cl_khr_terminate_context
```

CL Image Objects > GL Renderbuffers [9.6.4]

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

flags: See `clCreateFromGLBuffer`

Query Information [9.6.5]

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

*gl_object_type returns:

CL_GL_OBJECT_TEXTURE_BUFFER, CL_GL_OBJECT_TEXTURE_{1D, 2D, 3D}, CL_GL_OBJECT_TEXTURE_{1D, 2D}_ARRAY, CL_GL_OBJECT_{BUFFER, RENDERBUFFER}

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

param_name:

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

Share Objects [9.6.6]

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

CL Event Objects > GL Sync Objects [9.7.4]

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

Requires the `cl_khr_gl_event` extension.

Direct3D 11 Sharing [9.10.7.3 - 9.10.7.6]

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

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

d3d_device_source: CL_D3D11_DEVICE_KHR, CL_D3D11_DXGI_ADAPTER_KHR

d3d_device_set: CL_ALL_DEVICES_FOR_D3D11_KHR, CL_PREFERRED_DEVICES_FOR_D3D11_KHR

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

flags: See `clCreateFromGLBuffer`

Direct3D 10 Sharing [9.8.7]

These functions require the `cl_khr_d3d10_sharing` extension. The associated header file is `cl_d3d10.h`.

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

d3d_device_source:

CL_D3D10_{DEVICE, DXGI_ADAPTER}_KHR

d3d_device_set:

CL_{ALL, PREFERRED}_DEVICES_FOR_D3D10_KHR

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

flags: See `clCreateFromGLBuffer`

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

flags: See `clCreateFromD3D10BufferKHR`

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

flags: See `clCreateFromGLBuffer`

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

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

flags: See `clCreateFromGLBuffer`

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

flags: See `clCreateFromGLBuffer`

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

Create CL Event Objects from EGL [9.20]

This function requires the extension `cl_khr_egl_event`.

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


OpenCL Reference Card Index

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

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



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

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