# OpenCL API 1.2 Reference Card - Page 1

**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.

[n.n.n] refers to the section in the OpenCL Specification.
[n.n.n] refers to the section in the OpenCL Extension Specification
Text shown in purple is as per the OpenCL Extension Specification.

Specifications are available at www.khronos.org/opencl.

## The OpenCL Runtime

### Command Queues [5.1]

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

*properties:* CL_QUEUE_PROFILING_ENABLE,
    CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE

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:* CL_QUEUE_CONTEXT,
    CL_QUEUE_DEVICE,
    CL_QUEUE_REFERENCE_COUNT,
    CL_QUEUE_PROPERTIES

## 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.

### Querying Platform Info and Devices [4.1, 4.2]

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

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

*param_name:* CL_PLATFORM_{PROFILE, VERSION},
    CL_PLATFORM_{NAME, VENDOR, EXTENSIONS}

cl_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:* 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:*
    CL_DEVICE_{NAME, VENDOR, PROFILE, TYPE},
    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_PREFERRED_VECTOR_WIDTH_{CHAR, INT},
    CL_DEVICE_PREFERRED_VECTOR_WIDTH_{LONG, SHORT},
    CL_DEVICE_PREFERRED_VECTOR_WIDTH_{DOUBLE, HALF},
    CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT,
    CL_DEVICE_PREFERRED_INTEROP_USER_SYNC,
    CL_DEVICE_ADDRESS_BITS, CL_DEVICE_AVAILABLE,
    CL_DEVICE_BUILT_IN_KERNELS,
    CL_DEVICE_COMPILER_AVAILABLE,
    CL_DEVICE_{DOUBLE, HALF, SINGLE}_FP_CONFIG,
    CL_DEVICE_ENDIAN_LITTLE, CL_DEVICE_EXTENSIONS,
    CL_DEVICE_ERROR_CORRECTION_SUPPORT,
    CL_DEVICE_EXECUTION_CAPABILITIES,
    CL_DEVICE_GLOBAL_MEM_CACHE_{SIZE, TYPE},
    CL_DEVICE_GLOBAL_MEM_{CACHELINE_SIZE, SIZE},
    CL_DEVICE_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_LOCAL_MEM_{TYPE, SIZE},
    CL_DEVICE_MAX_{READ, WRITE}_IMAGE_ARGS,
    CL_DEVICE_MAX_CLOCK_FREQUENCY,
    CL_DEVICE_MAX_COMPUTE_UNITS,
    CL_DEVICE_MAX_CONSTANT_{ARGS,BUFFER_SIZE},
    CL_DEVICE_MAX_{MEM_ALLOC, PARAMETER}_SIZE,
    CL_DEVICE_MAX_SAMPLERS,
    CL_DEVICE_MAX_WORK_GROUP_SIZE,
    CL_DEVICE_MAX_WORK_ITEM_{DIMENSIONS, SIZES},
    CL_DEVICE_MEM_BASE_ADDR_ALIGN,
    CL_DEVICE_OPENCL_C_VERSION, CL_DEVICE_PARENT_DEVICE,
    CL_DEVICE_PARTITION_AFFINITY_DOMAIN,
    CL_DEVICE_PARTITION_MAX_SUB_DEVICES,
    CL_DEVICE_PARTITION_{PROPERTIES, TYPE},
    CL_DEVICE_PLATFORM, CL_DEVICE_PRINTF_BUFFER_SIZE,
    CL_DEVICE_PROFILING_TIMER_RESOLUTION,
    CL_DEVICE_QUEUE_PROPERTIES,
    CL_DEVICE_REFERENCE_COUNT,
    CL_DEVICE_VENDOR_ID, 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, AFFINITY_DOMAIN}
    (Affinity domains may be:
    CL_DEVICE_AFFINITY_DOMAIN_NUMA,
    CL_DEVICE_AFFINITY_DOMAIN_{L4, L3, L2, L1}_CACHE,
    CL_DEVICE_AFFINITY_DOMAIN_NEXT_PARTITIONABLE)

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

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

## Buffer Objects

Elements of a buffer object are stored sequentially and accessed using a pointer by a kernel executing on a device. Data is stored in the same format as it is accessed by the kernel.

### 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:* CL_MEM_READ_WRITE,
    CL_MEM_{WRITE, READ}_ONLY,
    CL_MEM_HOST_NO_ACCESS,
    CL_MEM_HOST_{READ, WRITE}_ONLY,
    CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

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

*flags:* same as for **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*)

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

### Contexts [4.4]

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

*properties:* NULL or CL_CONTEXT_PLATFORM,
    CL_CONTEXT_INTEROP_USER_SYNC,
    CL_CONTEXT_{D3D10, D3D11}_DEVICE_KHR,
    CL_CONTEXT_ADAPTER_{D3D9, D3D9EX, DXVA}_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**

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

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

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

*param_name:* CL_CONTEXT_REFERENCE_COUNT,
    CL_CONTEXT_{DEVICES, NUM_DEVICES, PROPERTIES},
    CL_CONTEXT_D3D10_PREFER_SHARED_RESOURCES_KHR,
    CL_CONTEXT_D3D11_PREFER_SHARED_RESOURCES_KHR

### Get CL Extension Function Pointers [9.2]

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

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 [5.2.3]

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

*map_flags:* CL_MAP_{READ, WRITE},
    CL_MAP_WRITE_INVALIDATE_REGION

### Memory Objects [5.4.1, 5.4.2]

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 [5.4.4]

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 [5.4.5]

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

*param_name:* CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR},
    CL_MEM_{MAP, REFERENCE}_COUNT, CL_MEM_OFFSET,
    CL_MEM_CONTEXT, CL_MEM_ASSOCIATED_MEMOBJECT,
    CL_MEM_{D3D10, D3D11}_RESOURCE_KHR,
    CL_MEM_DX9_MEDIA_ADAPTER_TYPE_KHR,
    CL_MEM_DX9_MEDIA_SURFACE_INFO_KHR

## OpenCL Class Diagram [2.1]

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

### Annotations

| Relationships | |
| --- | --- |
| abstract classes | {abstract} |
| aggregations | ◆ |
| inheritance | △ |
| relationship navigability | ⌃ |

| Cardinality | |
| --- | --- |
| many | * |
| one and only one | 1 |
| optionally one | 0..1 |
| one or more | 1..* |
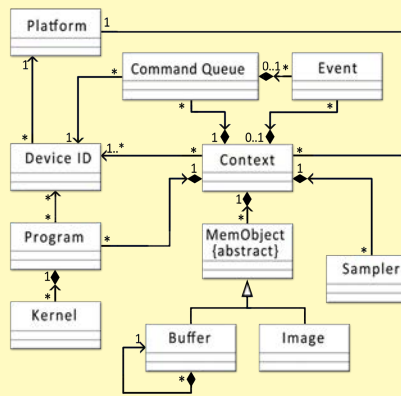


[1] Unified Modeling Language (http://www.uml.org/) is a trademark of Object Management Group (OMG).

## OpenCL Device Architecture Diagram [3.3]

The table below shows memory regions with allocation and memory access capabilities.

| | Global | Constant | Local | Private |
| --- | --- | --- | --- | --- |
| Host | Dynamic allocation Read/Write access | Dynamic allocation Read/Write access | Dynamic allocation No access | No allocation No access |
| Kernel | No allocation Read/Write access | Static allocation Read-only access | Static allocation Read/Write access | Static allocation Read/Write access |

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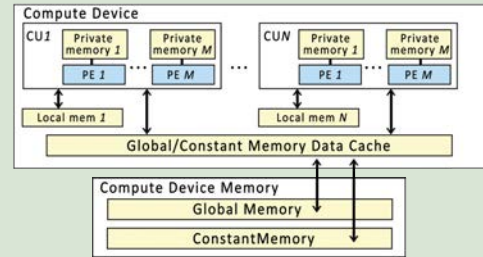


# Program Objects

## Create Program Objects [5.6.1]

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 [5.6.2]

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 [5.6.3]

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

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

## Unload the OpenCL Compiler [5.6.6]

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

## Query Program Objects [5.6.7]

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:* 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:* CL_PROGRAM_BINARY_TYPE,
    CL_PROGRAM_BUILD_{STATUS, OPTIONS, LOG}

## Compiler Options [5.6.4]

**Preprocessor**: (-D processed in order listed in *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

**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.

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

## Linker Options [5.6.5]

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

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

# Kernel and Event Objects

## Create Kernel Objects [5.7.1]

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 [5.7.2, 5.7.3]

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

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:* 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

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

*param_name:*
    CL_KERNEL_ARG_{ACCESS, ADDRESS, TYPE}_QUALIFIER,
    CL_KERNEL_ARG_NAME, CL_KERNEL_ARG_TYPE_NAME

## Execute Kernels [5.8]

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 **clEnqueueTask** (
    cl_command_queue *command_queue*,
    cl_kernel *kernel*, 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 (*user_func)(void *),
    void *args*, size_t *cb_args*, cl_uint *num_mem_objects*,
    const cl_mem *mem_list*, const void **args_mem_loc*,
    cl_uint *num_events_in_wait_list*,
    const cl_event *event_wait_list*, cl_event *event*)

## Event Objects [5.9]

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

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

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

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

*param_name:* CL_EVENT_COMMAND_{QUEUE, TYPE},
    CL_EVENT_{CONTEXT, REFERENCE_COUNT},
    CL_EVENT_COMMAND_EXECUTION_STATUS

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

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

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

## Markers, Barriers, and Waiting for Events [5.10]

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 [5.12]

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:* CL_PROFILING_COMMAND_QUEUED,
    CL_PROFILING_COMMAND_{SUBMIT, START, END}

## Flush and Finish [5.13]

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

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

## 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 |

### Other Built-in Data Types [6.1.3]

The optional types listed here other than event_t are only defined if CL_DEVICE_IMAGE_SUPPORT is CL_TRUE.

| OpenCL Type | | Description |
|---|---|---|
| image2d_t | OPTIONAL | 2D image handle |
| image3d_t | OPTIONAL | 3D image handle |
| image2d_array_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 |
| sampler_t | OPTIONAL | sampler handle |
| event_t | | event handle |

### Reserved Data Types [6.1.4]

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

## Vector Component Addressing [6.1.7]

### Vector Components

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

### Vector Addressing Equivalences

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

| | v.lo | v.hi | v.odd | v.even | | | 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 | | float8 | v.s0123 | v.s4567 | v.s1357 | v.s0246 |
| float3 * | v.s01, v.xy | v.s23, v.zw | v.s13, v.yw | v.s02, v.xz | | float16 | v.s01234567 | v.s89abcdef | v.s13579bdf | v.s02468ace |
| 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 that 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)))
```

## Specify Type Attributes [6.11.1]

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))
```

## Math Constants [6.12.2] [9.5.2]

The values of the following symbolic constants are type float, accurate within the precision of a single precision floating-point number.

| MAXFLOAT | Value of maximum non-infinite single-precision floating-point number. |
|---|---|
| HUGE_VALF | Positive float expression, evaluates to +infinity. |
| HUGE_VAL | Positive double expression, evals. to +infinity. OPTIONAL |

| INFINITY | Constant float expression, positive or unsigned infinity. |
|---|---|
| NAN | Constant float expression, quiet NaN. |

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

| M_E_F | Value of e |
|---|---|
| M_LOG2E_F | Value of $\log_2 e$ |
| M_LOG10E_F | Value of $\log_{10} e$ |

| M_LN2_F | Value of $\log_e 2$ |
|---|---|
| M_LN10_F | Value of $\log_e 10$ |
| M_PI_F | Value of $\pi$ |
| M_PI_2_F | Value of $\pi / 2$ |
| M_PI_4_F | Value of $\pi / 4$ |
| M_1_PI_F | Value of $1 / \pi$ |
| M_2_PI_F | Value of $2 / \pi$ |
| M_2_SQRTPI_F | Value of $2 / \sqrt{\pi}$ |
| M_SQRT2_F | Value of $\sqrt{2}$ |
| M_SQRT1_2_F | Value of $1 / \sqrt{2}$ |

## Integer Built-in Functions [6.12.3]

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

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

| $T$ **sub_sat** ($T x$, $T y$) | $x - y$ and saturates the result |
|---|---|
| $T$ **popcount** ($T x$) | Number of non-zero bits in $x$ |

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

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

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

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

## Preprocessor Directives & Macros [6.10]

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

| #pragma OPENCL EXTENSION extensionname : behavior | |
|---|---|

| #pragma OPENCL EXTENSION all : behavior | |
|---|---|

| __FILE__ | Current source file |
|---|---|
| __func__ | Current function name |

| __LINE__ | Integer line number |
|---|---|
| __OPENCL_VERSION__ | Integer version number, e.g: 120 |
| __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 |
| __OPENCL_C_VERSION__ | Sub. integer for OpenCL C version. |
| __ENDIAN_LITTLE__ | 1 if device is little endian |
| __IMAGE_SUPPORT__ | 1 if images are supported |

| __FAST_RELAXED_MATH__ | 1 if –cl-fast-relaxed-math optimization option is specified |
|---|---|
| FP_FAST_FMA | Defined if double **fma** is fast |
| FP_FAST_FMAF | Defined if float **fma** is fast |
| FP_FAST_FMA_HALF | Defined if half **fma** is fast |
| __kernel_exec(X, typen) | Same as: __kernel __attribute__((work_group_size_hint(X, 1, 1))) __attribute__((vec_type_hint(typen))) |

## Math Built-in Functions [6.12.2] [9.5.2]

*Ts* is type float, optionally double, or half if the half extension is enabled. *Tn* is the vector form of *Ts*, where *n* is 2, 3, 4, 8, or 16. *T* is *Ts* and *Tn*. *Q* is qualifier __global, __local, or __private. **HN** indicates that half and native variants are available using only the float or float*n* types by prepending "half_" or "native_" to the function name. Prototypes shown in brown text are available in half_ and native_ forms only using the float or float*n* types.

| | | |
|---|---|---|
| *T* **acos** (*T*) | | Arc cosine |
| *T* **acosh** (*T*) | | Inverse hyperbolic cosine |
| *T* **acospi** (*T x*) | | acos (*x*) / π |
| *T* **asin** (*T*) | | Arc sine |
| *T* **asinh** (*T*) | | Inverse hyperbolic sine |
| *T* **asinpi** (*T x*) | | asin (*x*) / π |
| *T* **atan** (*T y_over_x*) | | Arc tangent |
| *T* **atan2** (*T y, T x*) | | Arc tangent of *y* / *x* |
| *T* **atanh** (*T*) | | Hyperbolic arc tangent |
| *T* **atanpi** (*T x*) | | atan (*x*) / π |
| *T* **atan2pi** (*T x, T y*) | | atan2 (*y, x*) / π |
| *T* **cbrt** (*T*) | | Cube root |
| *T* **ceil** (*T*) | | Round to integer toward + infinity |
| *T* **copysign** (*T x, T y*) | | *x* with sign changed to sign of *y* |
| *T* **cos** (*T*) | HN | Cosine |
| *T* **cosh** (*T*) | | Hyperbolic cosine |
| *T* **cospi** (*T x*) | | cos (π *x*) |
| *T* **half_divide** (*T x, T y*)<br>*T* **native_divide** (*T x, T y*) | | *x* / *y*<br>(*T* may only be float or float*n*) |
| *T* **erfc** (*T*) | | Complementary error function |
| *T* **erf** (*T*) | | Calculates error function of *T* |
| *T* **exp** (*T x*) | HN | Exponential base e |
| *T* **exp2** (*T*) | HN | Exponential base 2 |

| | | |
|---|---|---|
| *T* **exp10** (*T*) | HN | Exponential base 10 |
| *T* **expm1** (*T x*) | | e^*x* -1.0 |
| *T* **fabs** (*T*) | | Absolute value |
| *T* **fdim** (*T x, T y*) | | Positive difference between *x* and *y* |
| *T* **floor** (*T*) | | Round to integer toward - infinity |
| *T* **fma** (*T a, T b, T c*) | | Multiply and add, then round |
| *T* **fmax** (*T x, T y*)<br>*Tn* **fmax** (*Tn x, Ts y*) | | Return *y* if *x* < *y*, otherwise it returns *x* |
| *T* **fmin** (*T x, T y*)<br>*Tn* **fmin** (*Tn x, Ts y*) | | Return *y* if *y* < *x*, otherwise it returns *x* |
| *T* **fmod** (*T x, T y*) | | Modulus. Returns *x* − *y* * trunc (*x/y*) |
| *T* **fract** (*T x, Q T \*iptr*) | | Fractional value in *x* |
| *Ts* **frexp** (*T x, Q int \*exp*)<br>*Tn* **frexp** (*T x, Q int*n *\*exp*) | | Extract mantissa and exponent |
| *T* **hypot** (*T x, T y*) | | Square root of $x^2 + y^2$ |
| int[*n*] **ilogb** (*T x*) | | Return exponent as an integer value |
| *Ts* **ldexp** (*T x, int n*)<br>*Tn* **ldexp** (*T x, int*n *n*) | | $x * 2^n$ |
| *T* **lgamma** (*T x*)<br>*Ts* **lgamma_r** (*T x, Q int \*signp*)<br>*Tn* **lgamma_r** (*T x, Q int*n *\*signp*) | | Log gamma function |
| *T* **log** (*T*) | HN | Natural logarithm |
| *T* **log2** (*T*) | HN | Base 2 logarithm |
| *T* **log10** (*T*) | HN | Base 10 logarithm |
| *T* **log1p** (*T x*) | | ln (1.0 + *x*) |
| *T* **logb** (*T x*) | | Exponent of *x* |
| *T* **mad** (*T a, T b, T c*) | | Approximates *a* * *b* + *c* |
| *T* **maxmag** (*T x, T y*) | | Maximum magnitude of *x* and *y* |
| *T* **minmag** (*T x, T y*) | | Minimum magnitude of *x* and *y* |

| | | |
|---|---|---|
| *T* **modf** (*T x, Q T \*iptr*) | | Decompose floating-point number |
| float[*n*] **nan** (uint[*n*] *nancode*)<br>half[*n*] **nan** (ushort[*n*] *nancode*)<br>double[*n*] **nan** (ulong[*n*] *nancode*) | | Quiet NaN<br>(Return is scalar when *nancode* is scalar) |
| *T* **nextafter** (*T x, T y*) | | Next representable floating-point value after *x* in the direction of *y* |
| *T* **pow** (*T x, T y*) | | Compute *x* to the power of *y* |
| *Ts* **pown** (*T x*, int *y*)<br>*Tn* **pown** (*T x*, int*n y*) | | Compute $x^y$, where *y* is an integer |
| *T* **powr** (*T x, T y*) | HN | Compute $x^y$, where *x* is >= 0 |
| *T* **half_recip** (*T x*)<br>*T* **native_recip** (*T x*) | | 1 / *x*<br>(*T* may only be float or float*n*) |
| *T* **remainder** (*T x, T y*) | | Floating point remainder |
| *Ts* **remquo** (*T x, T y, Q int \*quo*)<br>*Tn* **remquo** (*T x, T y, Q int*n *\*quo*) | | Remainder and quotient |
| *T* **rint** (*T*) | | Round to nearest even integer |
| *Ts* **rootn** (*T x*, int *y*)<br>*Tn* **rootn** (*T x*, int*n y*) | | Compute *x* to the power of 1/*y* |
| *T* **round** (*T x*) | | Integral value nearest to *x* rounding |
| *T* **rsqrt** (*T*) | HN | Inverse square root |
| *T* **sin** (*T*) | HN | Sine |
| *T* **sincos** (*T x, Q T \*cosval*) | | Sine and cosine of *x* |
| *T* **sinh** (*T*) | | Hyperbolic sine |
| *T* **sinpi** (*T x*) | | sin (π *x*) |
| *T* **sqrt** (*T*) | HN | Square root |
| *T* **tan** (*T*) | HN | Tangent |
| *T* **tanh** (*T*) | | Hyperbolic tangent |
| *T* **tanpi** (*T x*) | | tan (π *x*) |
| *T* **tgamma** (*T*) | | Gamma function |
| *T* **trunc** (*T*) | | Round to integer toward zero |

## Geometric Built-in Functions [6.12.5] [9.5.4]

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

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

## Vector Data Load/Store [6.12.7] [9.5.6]

*T* is type char, uchar, short, ushort, int, uint, long, ulong, or float, optionally double, or half if the half extension is enabled. *Tn* refers to the vector form of type *T*, where *n* is 2, 3, 4, 8, or 16. *Q* is an Address Space Qualifier listed in 6.5 unless otherwise noted. When red, *Q* cannot be __constant. *R* defaults to the current rounding mode, or is one of the Rounding Modes listed in 6.2.3.2.

| | |
|---|---|
| *Tn* **vload***n* (size_t *offset*,<br>const *Q T \*p*) | Read vector data from address (*p* + (*offset* * *n*)) |
| void **vstore***n* (*Tn data*,<br>size_t *offset*, *Q T \*p*) | Write vector data to address (*p* + (*offset* * *n*) |
| float **vload_half** (size_t *offset*,<br>const *Q* half *\*p*) | Read a half from address (*p* + *offset*) |
| float*n* **vload_half***n* (size_t *offset*,<br>const *Q* half *\*p*) | Read a half*n* from address (*p* + (*offset* * *n*)) |
| void **vstore_half** (float *data*,<br>size_t *offset*, *Q* half *\*p*)<br>void **vstore_half**_*R* (float *data*,<br>size_t *offset*, *Q* half *\*p*)<br>void **vstore_half** (double *data*,<br>size_t *offset*, *Q* half *\*p*) | Write a half to address (*p* + *offset*) |
| void **vstore_half**_*R* (double *data*,<br>size_t *offset*, *Q* half *\*p*) | Write a half to address (*p* + *offset*) |
| void **vstore_half***n* (float*n data*,<br>size_t *offset*, *Q* half *\*p*)<br>void **vstore_half***n*_*R* (float*n data*,<br>size_t *offset*, *Q* half *\*p*)<br>void **vstore_half***n* (double*n data*,<br>size_t *offset*, *Q* half *\*p*)<br>void **vstore_half***n*_*R* (double*n data*, size_t *offset*, *Q* half *\*p*) | Write a half vector to address (*p* + (*offset* * *n*)) |
| float*n* **vloada_half***n* (size_t *offset*,<br>const *Q* half *\*p*) | Read half vector data from (*p* + (*offset* * *n*)). For half3, read from (*p* + (*offset* * 4)). |
| void **vstorea_half***n* (float*n data*,<br>size_t *offset*, *Q* half *\*p*)<br>void **vstorea_half***n*_*R* (float*n data*,<br>size_t *offset*, *Q* half *\*p*)<br>void **vstorea_half***n* (double*n data*,<br>size_t *offset*, *Q* half *\*p*)<br>void **vstorea_half***n*_*R* (<br>double*n data*, size_t *offset*,<br>*Q* half *\*p*) | Write half vector data to (*p* + (*offset* * *n*)). For half3, write to (*p* + (*offset* * 4)). |

## Async Copies and Prefetch Functions [6.12.10] [9.5.7]

*T* is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, ulong*n*, float, float*n*, optionally double or double*n*, or half or half*n* if the half extension is enabled.

| | |
|---|---|
| event_t **async_work_group_copy** (__local *T \*dst*, const __global *T \*src*,<br>size_t *num_gentypes*, event_t *event*)<br>event_t **async_work_group_copy** (__global *T \*dst*, const __local *T \*src*,<br>size_t *num_gentypes*, event_t *event*) | Copies *num_gentypes T* elements from *src* to *dst* |
| event_t **async_work_group_strided_copy** (__local *T \*dst*, const __global *T \*src*,<br>size_t *num_gentypes*, size_t *src_stride*, event_t *event*)<br>event_t **async_work_group_strided_copy** (__global *T \*dst*, const __local *T \*src*,<br>size_t *num_gentypes*, size_t *dst_stride*, event_t *event*) | Copies *num_gentypes T* elements from *src* to *dst* |
| void **wait_group_events** (int *num_events*, event_t *\*event_list*) | Wait for events that identify the **async_work_group_copy** operations to complete |
| void **prefetch** (const __global *T \*p*, size_t *num_gentypes*) | Prefetch *num_gentypes* * sizeof(*T*) bytes into the global cache |

## Work-Item Built-in Functions [6.12.1]

These functions query the number of dimensions, the global and local work size specified to **clEnqueueNDRangeKernel**, and the global and local identifier of each work-item when this kernel is executed on a device. *D* is the dimension index.

| | |
|---|---|
| uint **get_work_dim** () | Number of dimensions in use |
| size_t **get_global_size** (uint *D*) | Number of global work-items |
| size_t **get_global_id** (uint *D*) | Global work-item ID value |
| size_t **get_local_size** (uint *D*) | Number of local work-items |
| size_t **get_local_id** (uint *D*) | Local work-item ID |
| size_t **get_num_groups** (uint *D*) | Number of work-groups |
| size_t **get_group_id** (uint *D*) | Returns the work-group ID |
| size_t **get_global_offset** (uint *D*) | Returns global offset |

## Common Built-in Functions [6.12.4] [9.5.3]

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

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

## Relational Built-in Functions [6.12.6]

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

| | |
|---|---|
| int  **isequal** (float x, float y)<br>intn **isequal** (floatn x, floatn y)<br>int  **isequal** (double x, double y)<br>longn **isequal** (doublen x, doublen y)<br>int  **isequal** (half x, half y)<br>shortn **isequal** (halfn x, halfn y) | Compare of x == y |
| int  **isnotequal** (float x, float y)<br>intn **isnotequal** (floatn x, floatn y)<br>int  **isnotequal** (double x, double y)<br>longn **isnotequal** (doublen x, doublen y)<br>int  **isnotequal** (half x, half y)<br>shortn **isnotequal** (halfn x, halfn y) | Compare of x != y |
| int  **isgreater** (float x, float y)<br>intn **isgreater** (floatn x, floatn y)<br>int  **isgreater** (double x, double y)<br>longn **isgreater** (doublen x, doublen y)<br>int  **isgreater** (half x, half y)<br>shortn **isgreater** (halfn x, halfn y) | Compare of x > y |
| int  **isgreaterequal** (float x, float y)<br>intn **isgreaterequal** (floatn x, floatn y)<br>int  **isgreaterequal** (double x, double y)<br>longn **isgreaterequal** (doublen x, doublen y)<br>int  **isgreaterequal** (half x, half y)<br>shortn **isgreaterequal** (halfn x, halfn y) | Compare of x >= y |
| int  **isless** (float x, float y)<br>intn **isless** (floatn x, floatn y)<br>int  **isless** (double x, double y)<br>longn **isless** (doublen x, doublen y)<br>int  **isless** (half x, half y)<br>shortn **isless** (halfn x, halfn y) | Compare of x < y |
| int  **islessequal** (float x, float y)<br>intn **islessequal** (floatn x, floatn y)<br>int  **islessequal** (double x, double y)<br>longn **islessequal** (doublen x, doublen y)<br>int  **islessequal** (half x, half y)<br>shortn **islessequal** (halfn x, halfn y) | Compare of x <= y |
| int  **islessgreater** (float x, float y)<br>intn **islessgreater** (floatn x, floatn y)<br>int  **islessgreater** (double x, double y)<br>longn **islessgreater** (doublen x, doublen y)<br>int  **islessgreater** (half x, half y)<br>shortn **islessgreater** (halfn x, halfn y) | Compare of (x < y) \|\| (x > y) |
| int  **isfinite** (float)<br>intn **isfinite** (floatn)<br>int  **isfinite** (double)<br>longn **isfinite** (doublen)<br>int  **isfinite** (half)<br>shortn **isfinite** (halfn) | Test for finite value |

| | |
|---|---|
| int  **isinf** (float)<br>intn **isinf** (floatn)<br>int  **isinf** (double)<br>longn **isinf** (doublen)<br>int  **isinf** (half)<br>shortn **isinf** (halfn) | Test for + or − infinity |
| int  **isnan** (float)<br>intn **isnan** (floatn)<br>int  **isnan** (double)<br>longn **isnan** (doublen)<br>int  **isnan** (half)<br>shortn **isnan** (halfn) | Test for a NaN |
| int  **isnormal** (float)<br>intn **isnormal** (floatn)<br>int  **isnormal** (double) | Test for a normal value |
| longn **isnormal** (doublen)<br>int  **isnormal** (half)<br>shortn **isnormal** (halfn) | Test for a normal value |
| int  **isordered** (float x, float y)<br>intn **isordered** (floatn x, floatn y)<br>int  **isordered** (double x, double y)<br>longn **isordered** (doublen x, doublen y)<br>int  **isordered** (half x, half y)<br>shortn **isordered** (halfn x, halfn y) | Test if arguments are ordered |
| int  **isunordered** (float x, float y)<br>intn **isunordered** (floatn x, floatn y)<br>int  **isunordered** (double x, double y)<br>longn **isunordered** (doublen x, doublen y)<br>int  **isunordered** (half x, half y)<br>shortn **isunordered** (halfn x, halfn y) | Test if arguments are unordered |
| int  **signbit** (float)<br>intn **signbit** (floatn)<br>int  **signbit** (double)<br>longn **signbit** (doublen)<br>int  **signbit** (half)<br>shortn **signbit** (halfn) | Test for sign bit |
| int **any** (Ti x) | 1 if MSB in component of x is set; else 0 |
| int **all** (Ti x) | 1 if MSB in all components of x are set; else 0 |
| T **bitselect** (T a, T b, T c)<br>half **bitselect** (half a, half b, half c)<br>halfn **bitselect** (halfn a, halfn b, halfn c) | Each bit of result is corresponding bit of a if corresponding bit of c is 0 |
| T **select** (T a, T b, Ti c)<br>T **select** (T a, T b, Tu c)<br>halfn **select** (halfn a, halfn b, shortn c)<br>half **select** (half a, half b, short c)<br>halfn **select** (halfn a, halfn b, ushortn c)<br>half **select** (half a, half b, ushort c) | For each component of a vector type, result[i] = if MSB of c[i] is set ? b[i] : a[i] For scalar type, result = c ? b : a |

## Atomic Functions [6.12.11] [9.3]

These functions functions provide atomic operations on 32-bit signed and unsigned integers and single precision floating-point to locations in __global or __local memory. *T* is type int or unsigned int. *T* may also be type float for **atomic_xchg**, and type long or ulong for extended 64-bit atomic functions. *Q* is volatile __global or volatile __local.

| | |
|---|---|
| T **atomic_add** (Q T *p, T val) | Read, add, and store |
| T **atomic_sub** (Q T *p, T val) | Read, subtract, and store |
| T **atomic_xchg** (Q T *p, T val) | Read, swap, and store |
| T **atomic_inc** (Q T *p) | Read, increment, and store |
| T **atomic_dec** (Q T *p) | Read, decrement, and store |
| T **atomic_cmpxchg** (Q T *p, T cmp, T val) | Read, store (*p ==cmp) ? val : *p |
| T **atomic_min** (Q T *p, T val) | Read, store min(*p, val) |
| T **atomic_max** (Q T *p, T val) | Read, store max(*p, val) |
| T **atomic_and** (Q T *p, T val) | Read, store (*p & val) |
| T **atomic_or** (Q T *p, T val) | Read, store (*p \| val) |
| T **atomic_xor** (Q T *p, T val) | Read, store (*p ^ val) |

Optional extensions enable forms of these functions using the atom_ prefix that implement atomic operations on 64-bit signed and unsigned integers. To use any of these forms, include the following in the OpenCL program source:

    #pragma OPENCL EXTENSION extension-name : enable

Use cl_khr_int64_base_atomics for *extension-name* to enable 64-bit versions of the following functions:

    atom_add       atom_sub       atom_inc
    atom_dec       atom_xchg      atom_cmpxchg

Use cl_khr_int64_extended_atomics for *extension-name* to enable 64-bit versions of the following functions:

    atom_min       atom_max       atom_and
    atom_or        atom_xor

## Conversions and Type Casting Examples [6.2]

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

*R* can be one of the following rounding modes:

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

## Synchronization and Explicit Memory Fence Functions [6.12.8, 6.12.9]

*flags* argument is the memory address space, set to a combination of CLK_LOCAL_MEM_FENCE and CLK_GLOBAL_MEM_FENCE. Explicit memory fence functions provide ordering between memory operations of a work-item.

| | |
|---|---|
| void **barrier** (<br>cl_mem_fence_flags flags) | Work-items in a work-group must execute this before any can continue |
| void **mem_fence** (<br>cl_mem_fence_flags flags) | Orders loads and stores of a work-item executing a kernel |
| void **read_mem_fence** (<br>cl_mem_fence_flags flags) | Orders memory loads |
| void **write_mem_fence** (<br>cl_mem_fence_flags flags) | Orders memory stores |

## Miscellaneous Vector Functions [6.12.12]

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

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

## printf Function [6.12.13]

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)

**OpenCL Image Processing:** Following is a subset of the OpenCL specification that pertains to image processing and graphics.

## Image Objects

### 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:*
  CL_MEM_READ_WRITE,
  CL_MEM_{WRITE, READ}_ONLY,
  CL_MEM_HOST_{WRITE, READ}_ONLY,
  CL_MEM_HOST_NO_ACCESS,
  CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

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

### Read, Write, Copy Image Objects [5.3.3]

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

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

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

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

### Copy Between Image, Buffer Objects [5.3.4]

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

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

### Map and Unmap Image Objects [5.3.5]

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*)

Also see **clGetMemObjectInfo** [5.4.5]

### Query Image Objects [5.3.6]

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:* CL_IMAGE_{ARRAY, ELEMENT}_SIZE,
  CL_IMAGE_{ROW, SLICE}_PITCH,
  CL_IMAGE_{FORMAT, BUFFER, HEIGHT, WIDTH, DEPTH},
  CL_IMAGE_NUM_{SAMPLES, MIP_LEVELS},
  CL_IMAGE_DX9_MEDIA_PLANE_KHR,
  CL_IMAGE_{D3D10, D3D11}_SUBRESOURCE_KHR

## Image Formats [5.3.1.1, 9.5]

Supported image formats: image_channel_order with image_channel_data_type.

Built-in support: [Table 5.8]

| |
|---|
| **CL_RGBA:** CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32} |
| **CL_BGRA:** CL_UNORM_INT8 |

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_LUMINANCE:** CL_UNORM_INT{8,16}, CL_HALF_FLOAT, CL_FLOAT, CL_SNORM_INT{8,16} |
| **CL_RG, CL_RA:** CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16, 32}, CL_UNSIGNED_INT{8,16,32}, CL_SNORM_INT{8,16} |
| **CL_RGB:** CL_UNORM_SHORT_{555,565}, CL_UNORM_INT_101010 |
| **CL_ARGB:** CL_UNORM_INT8, CL_SIGNED_INT8, CL_UNSIGNED_INT8, CL_SNORM_INT8 |
| **CL_BGRA:** CL_{SIGNED, UNSIGNED}_INT8, CL_SNORM_INT8 |

## Image Read and Write Built-in Functions

[6.12.14] [9.4, 9.5.8]

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. To enable the **read_imageh** and **write_imageh** forms, enable the extension cl_khr_fp16. To enable the type image3d_t in functions **write_image{f, i, ui, h}**, enable 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*)

### Read and write functions for 1D images (continued)

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*)

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*)

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*)

### Read and write functions for 2D images (continued)

void **write_imagef** (image2d_t *image*, int2 *coord*, float4 *color*)
void **write_imagef** (image2d_array_t *image*, int4 *coord*, float4 *color*)

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.

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*)

Use this pragma to enable writes to type image3d_t:
#pragma OPENCL EXTENSION cl_khr_3d_image_writes : enable

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*)

## 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

**OpenCL Image Processing (continued):** Following is a subset of the OpenCL specification that pertains to image processing and graphics.

## Sampler Objects [5.5]

cl_sampler **clCreateSampler** (
    cl_context *context*, cl_bool *normalized_coords*,
    cl_addressing_mode *addressing_mode*,
    cl_filter_mode *filter_mode*, cl_int *errcode_ret*)
*addressing_mode:* CL_ADDRESS_[MIRRORED_]REPEAT,
    CL_ADDRESS_CLAMP[_TO_EDGE], CL_ADDRESS_NONE
*filter_mode:* CL_FILTER_{NEAREST, LINEAR}

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

## Sampler Declaration Fields [6.12.14.1]
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, MIRRORED_REPEAT}

*filter-mode:* CLK_FILTER_NEAREST,  CLK_FILTER_LINEAR

## Direct3D 10 Sharing [9.9]
Provide interoperability between OpenCL and Direct3D 10. If supported, cl_khr_d3d10_sharing will be present in CL_PLATFORM_EXTENSIONS or CL_DEVICE_EXTENSIONS.

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:* CL_MEM_{READ, WRITE}_ONLY, CL_MEM_READ_WRITE

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 **clCreateFromD3D10BufferKHR**

cl_int **clEnqueueAcquireD3D10ObjectsKHR** (
    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_int **clEnqueueReleaseD3D10ObjectsKHR** (
    cl_command_queue *command_queue*,
    cl_uint *num_objects*, const cl_mem *mem_objects*,
    cl_uint *num_events_in_wait_list*,
    const cl_event *event_wait_list*, cl_event *event*)

## Direct3D 11 Sharing [9.11]
Provide interoperability between OpenCL and Direct3D 11. If supported, cl_khr_d3d11_sharing will be present in CL_PLATFORM_EXTENSIONS or CL_DEVICE_EXTENSIONS.

cl_mem **clCreateFromD3D11Texture2DKHR** (
    cl_context *context*, cl_mem_flags *flags*,
    ID3D11Texture2D *resource*,
    UINT *subresource*, cl_int *errcode_ret*)
*flags:* CL_MEM_{READ, WRITE}_ONLY, CL_MEM_READ_WRITE

## Image Query Functions [6.12.14.5]

### 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_height** (image{2,3}d_t *image*)
int **get_image_height** (image2d_array_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*)

### 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*)

### 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_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*)

## OpenGL Sharing
Functions available if cl_khr_gl_sharing or cl_apple_gl_sharing is supported. Creating OpenCL memory objects from OpenGL objects  using **clCreateFromGLBuffer**, **clCreateFromGLTexture**, and **clCreateFromGLRenderbuffer** ensure the OpenGL object will not be deleted while the corresponding OpenCL memory object exists.

### CL Buffer Objects > GL Buffer Objects [9.7.2]
cl_mem **clCreateFromGLBuffer** (cl_context *context*,
    cl_mem_flags *flags*, GLuint *bufobj*, cl_int *errcode_ret*)
*flags:* CL_MEM_{READ, WRITE}_ONLY, CL_MEM_READ_WRITE

### CL Image Objects > GL Textures [9.7.3]
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}

### CL Image Objects > GL Renderbuffers [9.7.4]
cl_mem **clCreateFromGLRenderbuffer** (
    cl_context *context*, cl_mem_flags *flags*,
    GLuint *renderbuffer*, cl_int *errcode_ret*)
*flags:* CL_MEM_{READ, WRITE}_ONLY, CL_MEM_READ_WRITE

### Query Information [9.7.5]
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}

### Share Objects [9.7.6]
cl_int **clEnqueueAcquireGLObjects** (
    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_int **clEnqueueReleaseGLObjects** (
    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 [9.8.2]
cl_event **clCreateEventFromGLsyncKHR** (
    cl_context *context*, GLsync *sync*, cl_int *errcode_ret*)

### CL Context > GL Context, Sharegroup [9.6.5]
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

## DX9 Media Surface Sharing [9.10]
These functions allow applications to use media surfaces as OpenCL memory objects. If this extension in supported, cl_khr_dx9_media_sharing will be present in CL_PLATFORM_EXTENSIONS or CL_DEVICE_EXTENSIONS.

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_mem_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_DEVICES_FOR_DXP_MEDIA_ADAPTER_KHR,
    CL_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:* CL_MEM_{READ, WRITE}_ONLY, CL_MEM_READ_WRITE
*adapter_type:* CL_ADAPTER_{D3D9, D3D9EX, DXVA}_KHR

cl_int **clEnqueueAcquireDX9MediaSurfacesKHR** (
    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_int **clEnqueueReleaseDX9MediaSurfacesKHR** (
    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_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_PREFERRED_DEVICES_FOR_D3D11_KHR,
    CL_ALL_DEVICES_FOR_D3D11_KHR

cl_mem **clCreateFromD3D11BufferKHR** (
    cl_context *context*, cl_mem_flags *flags*,
    ID3D11Buffer *resource*, cl_int *errcode_ret*)
*flags:* CL_MEM_{READ, WRITE}_ONLY, CL_MEM_READ_WRITE

cl_mem **clCreateFromD3D11Texture3DKHR** (
    cl_context *context*, cl_mem_flags *flags*,
    ID3D11Texture3D *resource*, UINT *subresource*,
    cl_int *errcode_ret*)
*flags:* CL_MEM_{READ, WRITE}_ONLY, CL_MEM_READ_WRITE

cl_int **clEnqueueAcquireD3D11ObjectsKHR** (
    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_int **clEnqueueReleaseD3D11ObjectsKHR** (
    cl_command_queue *command_queue*,
    cl_uint *num_objects*, const cl_mem *mem_objects*,
    cl_uint *num_events_in_wait_list*,
    const cl_event *event_wait_list*, cl_event *event*)

## OpenCL 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.