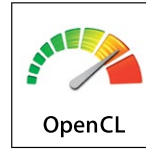


**OpenCL™ (Open Computing Language)** is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, GPUs, and other processors. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance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

Specification documents and online reference are available at [www.khronos.org/ocl](http://www.khronos.org/ocl).



**[n.n.n]** and **purple text**: sections and text in the OpenCL API 2.2 Spec.  
**[n.n.n]** and **green text**: sections and text in the OpenCL C++ 2.2 Spec.  
**[n.n.n]** and **brown text**: sections and text in the OpenCL C 2.0 Spec.  
**[n.n.n]** and **blue text**: sections and text in the OpenCL Extension 2.2 Spec.

## OpenCL API Reference

Section and table references are to the OpenCL API 2.2 specification.

### The OpenCL Platform Layer

The OpenCL platform layer implements platform-specific features that allow applications to query OpenCL devices, device configuration information, and to create OpenCL contexts using one or more devices. **Items in blue apply only when the appropriate extension is enabled** (see Extensions on page 21 of this reference guide).

#### 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 clCldGetPlatformIDsKHR` (`cl_uint num_entries,`  
`cl_platform_id *platforms, cl_uint *num_platforms`)

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

*param\_name*: CL\_PLATFORM\_{PROFILE, VERSION},  
 CL\_PLATFORM\_{NAME, VENDOR, EXTENSIONS},  
 CL\_PLATFORM\_HOST\_TIMER\_RESOLUTION,  
 CL\_PLATFORM\_ICD\_SUFFIX\_KHR [Table 4.1]

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

*device\_type*: [Table 4.2]

CL\_DEVICE\_TYPE\_{ACCELERATOR, ALL, CPU},  
 CL\_DEVICE\_TYPE\_{CUSTOM, DEFAULT, GPU}

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

*param\_name*: [Table 4.3]

CL\_DEVICE\_ADDRESS\_BITS, CL\_DEVICE\_AVAILABLE,  
 CL\_DEVICE\_BUILT\_IN\_KERNELS,  
 CL\_DEVICE\_COMPILER\_AVAILABLE,  
 CL\_DEVICE\_{DOUBLE, HALF, SINGLE}\_FP\_CONFIG,  
 CL\_DEVICE\_ENDIAN\_LITTLE,  
 CL\_DEVICE\_EXTENSIONS,  
 CL\_DEVICE\_ERROR\_CORRECTION\_SUPPORT,  
 CL\_DEVICE\_EXECUTION\_CAPABILITIES,  
 CL\_DEVICE\_GLOBAL\_MEM\_CACHE\_{SIZE, TYPE},  
 CL\_DEVICE\_GLOBAL\_MEM\_{CACHELINE\_SIZE, SIZE},  
 CL\_DEVICE\_GLOBAL\_VARIABLE\_PREFERRED\_TOTAL\_SIZE,  
 CL\_DEVICE\_IL\_VERSION,  
 CL\_DEVICE\_IMAGE\_MAX\_{ARRAY, BUFFER}\_SIZE,  
 CL\_DEVICE\_IMAGE\_SUPPORT,  
 CL\_DEVICE\_IMAGE2D\_MAX\_{WIDTH, HEIGHT},  
 CL\_DEVICE\_IMAGE3D\_MAX\_{WIDTH, HEIGHT,  
 DEPTH},  
 CL\_DEVICE\_IMAGE\_BASE\_ADDRESS\_ALIGNMENT,  
 CL\_DEVICE\_IMAGE\_PITCH\_ALIGNMENT,  
 CL\_DEVICE\_LINKER\_AVAILABLE,  
 CL\_DEVICE\_LOCAL\_MEM\_{TYPE, SIZE},  
 CL\_DEVICE\_MAX\_{CLOCK\_FREQUENCY, PIPE\_ARGS},  
 CL\_DEVICE\_MAX\_{COMPUTE\_UNITS, SAMPLERS},  
 CL\_DEVICE\_MAX\_CONSTANT\_{ARGS, BUFFER\_SIZE},  
 CL\_DEVICE\_MAX\_GLOBAL\_VARIABLE\_SIZE,  
 CL\_DEVICE\_MAX\_{MEM\_ALLOC, PARAMETER}\_SIZE,  
 CL\_DEVICE\_MAX\_NUM\_SUB\_GROUPS,  
 CL\_DEVICE\_MAX\_ON\_DEVICE\_{QUEUES, EVENTS},  
 CL\_DEVICE\_MAX\_{READ, WRITE}\_IMAGE\_ARGS,  
 CL\_DEVICE\_MAX\_READ\_WRITE\_IMAGE\_ARGS,  
 CL\_DEVICE\_MAX\_SUB\_GROUPS,  
 CL\_DEVICE\_MAX\_WORK\_GROUP\_SIZE,  
 CL\_DEVICE\_MAX\_WORK\_ITEM\_{DIMENSIONS, SIZES},  
 CL\_DEVICE\_MEM\_BASE\_ADDR\_ALIGN,  
 CL\_DEVICE\_NAME,  
 CL\_DEVICE\_NATIVE\_VECTOR\_WIDTH\_{  
 (CHAR, INT, DOUBLE, HALF, LONG, SHORT, FLOAT),  
 CL\_DEVICE\_NATIVE\_VECTOR\_WIDTH\_FLOAT,

CL\_DEVICE\_{OPENCL\_C\_VERSION, PARENT\_DEVICE},  
 CL\_DEVICE\_PARTITION\_AFFINITY\_DOMAIN,  
 CL\_DEVICE\_PARTITION\_MAX\_SUB\_DEVICES,  
 CL\_DEVICE\_PARTITION\_{PROPERTIES, TYPE},  
 CL\_DEVICE\_PIPE\_MAX\_ACTIVE\_RESERVATIONS,  
 CL\_DEVICE\_PIPE\_MAX\_PACKET\_SIZE,  
 CL\_DEVICE\_{PLATFORM, PRINTF\_BUFFER\_SIZE},  
 CL\_DEVICE\_PREFERRED\_Y\_ATOMIC\_ALIGNMENT  
 (where Y may be LOCAL, GLOBAL, PLATFORM),  
 CL\_DEVICE\_PREFERRED\_VECTOR\_WIDTH\_Z  
 (where Z may be CHAR, INT, DOUBLE, HALF, LONG,  
 SHORT, FLOAT),  
 CL\_DEVICE\_PREFERRED\_INTEROP\_USER\_SYNC,  
 CL\_DEVICE\_PROFILE,  
 CL\_DEVICE\_PROFILING\_TIMER\_RESOLUTION,  
 CL\_DEVICE\_SPIR\_VERSIONS,  
 CL\_DEVICE\_SUB\_GROUP\_INDEPENDENT\_FORWARD\_  
 PROGRESS  
 CL\_DEVICE\_QUEUE\_ON\_{DEVICE, HOST}\_PROPERTIES,  
 CL\_DEVICE\_QUEUE\_ON\_DEVICE\_MAX\_SIZE,  
 CL\_DEVICE\_QUEUE\_ON\_DEVICE\_PREFERRED\_SIZE,  
 CL\_DEVICE\_{REFERENCE\_COUNT, VENDOR\_ID},  
 CL\_DEVICE\_SVM\_CAPABILITIES,  
 CL\_DEVICE\_TERMINATE\_CAPABILITY\_KHR,  
 CL\_DEVICE\_{TYPE, VENDOR},  
 CL\_DEVICE\_VENDOR\_ID,  
 CL\_{DEVICE, DRIVER}\_VERSION,  
 CL\_DEVICE\_MAX\_NAMED\_BARRIER\_COUNT\_KHR

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

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

#### Partitioning a device [4.3]

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

*properties*: [Table 4.4] CL\_DEVICE\_PARTITION\_EQUALLY,  
 CL\_DEVICE\_PARTITION\_BY\_COUNTS,  
 CL\_DEVICE\_PARTITION\_BY\_AFFINITY\_DOMAIN

### 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*: Points to a zero-terminated list of properties and their values: [Table 5.1] CL\_QUEUE\_SIZE,  
 CL\_QUEUE\_PROPERTIES (bitfield which may be set to an OR of CL\_QUEUE\_\* where \* may be: OUT\_OF\_ORDER\_EXEC\_MODE\_ENABLE, PROFILING\_ENABLE, ON\_DEVICE[DEFAULT]),  
 CL\_QUEUE\_THROTTLE\_{HIGH, MED, LOW}\_KHR (requires the `cl_khr_throttle_hint` extension),  
 CL\_QUEUE\_PRIORITY\_KHR (bitfield which may be one of CL\_QUEUE\_PRIORITY\_HIGH\_KHR,  
 CL\_QUEUE\_PRIORITY\_MED\_KHR,  
 CL\_QUEUE\_PRIORITY\_LOW\_KHR (requires the `cl_khr_priority_hints` extension))

`cl_int clRetainDevice` (`cl_device_id device`)

`cl_int clReleaseDevice` (`cl_device_id device`)

#### Contexts [4.4]

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

*properties*: [Table 4.5]

NULL or CL\_CONTEXT\_PLATFORM,  
 CL\_CONTEXT\_INTEROP\_USER\_SYNC,  
 CL\_CONTEXT\_{D3D10, D3D11}\_DEVICE\_KHR,  
 CL\_CONTEXT\_ADAPTER\_{D3D9, D3D9EX}\_KHR,  
 CL\_CONTEXT\_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\_X where X may be REFERENCE\_COUNT,  
 DEVICES, NUM\_DEVICES, PROPERTIES,  
 D3D10\_PREFER\_SHARED\_RESOURCES\_KHR,  
 D3D11\_PREFER\_SHARED\_RESOURCES\_KHR [Table 4.6]

`cl_int clTerminateContextKHR` (`cl_context context`)

#### Get CL extension function pointers [9.2]

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

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

`cl_int clRetainCommandQueue` (`cl_command_queue command_queue`)

`cl_int clReleaseCommandQueue` (`cl_command_queue command_queue`)

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

*param\_name*: [Table 5.2]

CL\_QUEUE\_CONTEXT,  
 CL\_QUEUE\_DEVICE[DEFAULT], CL\_QUEUE\_SIZE,  
 CL\_QUEUE\_REFERENCE\_COUNT,  
 CL\_QUEUE\_PROPERTIES

## Buffer Objects

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

### Create buffer objects [5.2.1]

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

flags: [Table 5.3] CL\_MEM\_READ\_WRITE, CL\_MEM\_{WRITE, READ}\_ONLY, CL\_MEM\_HOST\_NO\_ACCESS, CL\_MEM\_HOST\_{READ, WRITE}\_ONLY, CL\_MEM\_{USE, ALLOC, COPY}\_HOST\_PTR

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

flags: See clCreateBuffer

buffer\_create\_type: CL\_BUFFER\_CREATE\_TYPE\_REGION

### Read, write, copy, & fill buffer objects [5.2.3]

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

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

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

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

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

## Image Formats [5.3.1.1]

Supported combinations of image\_channel\_order and image\_channel\_data\_type.

### Built-in support [Table 5.8]

**CL\_R (read or write):** CL\_[HALF\_]FLOAT, CL\_UNORM\_INT{8,16}, CL\_SNORM\_INT{8,16}, CL\_SIGNED\_INT{8,16,32}, CL\_UNSIGNED\_INT{8,16,32}

**CL\_DEPTH (read or write):** CL\_FLOAT, CL\_UNORM\_INT16

**CL\_DEPTH\_STENCIL (read only):** CL\_FLOAT, CL\_UNORM\_INT24

(Requires the extension *cl\_khr\_gl\_depth\_images*)

**CL\_RG (read or write):** CL\_[HALF\_]FLOAT, CL\_UNORM\_INT{8,16}, CL\_SNORM\_INT{8,16}, CL\_SIGNED\_INT{8,16,32}, CL\_UNSIGNED\_INT{8,16,32}

**CL\_RGBA (read or write):** CL\_[HALF\_]FLOAT, CL\_UNORM\_INT{8,16}, CL\_UNORM\_INT\_101010\_2, CL\_SNORM\_INT{8,16}, CL\_SIGNED\_INT{8,16,32}, CL\_UNSIGNED\_INT{8,16,32}

**CL\_BGRA (read or write):** CL\_UNORM\_INT8

**CL\_sRGBA (read only):** CL\_UNORM\_INT8

(Requires the extension *cl\_khr\_srgb\_image\_writes*)

### Supported image channel order values [Table 5.6]

**CL\_R, CL\_A (read and write):** CL\_[HALF\_]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\_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\_SNORM\_INT{8,16}

**CL\_RG, CL\_RA:** CL\_[HALF\_]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

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

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

### Map buffer objects [5.2.4]

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

map\_flags: CL\_MAP\_{READ, WRITE}, CL\_MAP\_WRITE\_INVALIDATE\_REGION

## Image Objects

Items in blue apply when the appropriate extension is enabled.

### Create image objects [5.3.1]

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

flags: See clCreateBuffer

### Query list of supported image formats [5.3.2]

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

flags: See clCreateBuffer

image\_type: CL\_MEM\_OBJECT\_IMAGE{1D, 2D, 3D}, CL\_MEM\_OBJECT\_IMAGE1D\_BUFFER, CL\_MEM\_OBJECT\_IMAGE{1D, 2D}\_ARRAY

### Read, write, copy, & fill image objects [5.3.3-4]

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

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

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

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

### Copy between image & buffer objects [5.3.5]

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

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

### Map and unmap image objects [5.3.6]

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

map\_flags: CL\_MAP\_{READ, WRITE}, CL\_MAP\_WRITE\_INVALIDATE\_REGION

### Query image objects [5.3.7]

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

param\_name: [Table 5.10] CL\_IMAGE\_FORMAT, CL\_IMAGE\_{ARRAY, ELEMENT}\_SIZE, CL\_IMAGE\_{ROW, SLICE}\_PITCH, CL\_IMAGE\_{HEIGHT, WIDTH, DEPTH}, CL\_IMAGE\_NUM\_{SAMPLES, MIP\_LEVELS}, CL\_IMAGE\_DX9\_MEDIA\_PLANE\_KHR, CL\_IMAGE\_{D3D10, D3D11}\_SUBRESOURCE\_KHR



## 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}\_ONLY, CL\_MEM\_{READ\_WRITE, 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 [5.6]

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

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

flags: [Table 5.14]  
CL\_MEM\_READ\_WRITE,  
CL\_MEM\_{WRITE, READ}\_ONLY,  
CL\_MEM\_SVM\_FINE\_GRAIN\_BUFFER,  
CL\_MEM\_SVM\_ATOMICS

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

### Enqueuing SVM operations

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

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

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

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

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

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

### Flush and Finish [5.15]

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

## 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 enabled.

### 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)
```

## Sampler Objects [5.7]

Items in blue require the `cl_khr_mipmap_image` extension.

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

sampler\_properties: [Table 5.15]  
CL\_SAMPLER\_NORMALIZED\_COORDS,  
CL\_SAMPLER\_{ADDRESSING, FILTER}\_MODE,  
CL\_SAMPLER\_MIP\_FILTER\_MODE,  
CL\_SAMPLER\_LOD\_{MIN, MAX}

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

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

param\_name: CL\_SAMPLER\_REFERENCE\_COUNT,  
CL\_SAMPLER\_{CONTEXT, FILTER\_MODE},  
CL\_SAMPLER\_ADDRESSING\_MODE,  
CL\_SAMPLER\_NORMALIZED\_COORDS [Table 5.16]

## Program Objects

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

### Create program objects [5.8.1]

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

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

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

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

### Retain and release program objects [5.8.2]

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

```
cl_int clSetProgramReleaseCallback (
    cl_program program, void (CL_CALLBACK *pfn_notify)(
        cl_program prog, void *user_data),
    void *user_data)
```

### Set SPIR-V specialization constants [5.8.3]

```
cl_int clSetProgramSpecializationConstant (
    cl_program program, cl_uint spec_id, size_t spec_size,
    const void *spec_value)
```

### Building program executables [5.8.4]

```
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)
```

### 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: [Table 5.13]  
CL\_MEM\_{TYPE, FLAGS, SIZE, HOST\_PTR},  
CL\_MEM\_CONTEXT, CL\_MEM\_OFFSET,  
CL\_MEM\_{MAP, REFERENCE}\_COUNT,  
CL\_MEM\_ASSOCIATED\_MEMOBJECT,  
CL\_MEM\_USES\_SVM\_POINTER,  
CL\_MEM\_{D3D10, D3D11}\_RESOURCE\_KHR,  
CL\_MEM\_DX9\_MEDIA\_ADAPTER\_TYPE\_KHR,  
CL\_MEM\_DX9\_MEDIA\_SURFACE\_INFO\_KHR

### Sampler declaration fields [6.13.14.1]

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

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

normalized-mode:  
CLK\_NORMALIZED\_COORDS\_{TRUE, FALSE}

address-mode:  
CLK\_ADDRESS\_X, where X may be NONE, REPEAT,  
CLAMP, CLAMP\_TO\_EDGE, MIRROR\_REPEAT

filter-mode: CLK\_FILTER\_NEAREST, CLK\_FILTER\_LINEAR

### Separate compilation and linking [5.8.5]

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

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

### Unload the OpenCL compiler [5.8.8]

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

### Query program objects [5.8.9]

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

param\_name: [Table 5.17]  
CL\_PROGRAM\_{IL, REFERENCE\_COUNT},  
CL\_PROGRAM\_{CONTEXT, NUM\_DEVICES, DEVICES},  
CL\_PROGRAM\_{SOURCE, BINARY\_SIZES, BINARIES},  
CL\_PROGRAM\_{NUM\_KERNELS, KERNEL\_NAMES},  
CL\_PROGRAM\_SCOPE\_{GLOBAL\_{C,D}TORS\_PRESENT}

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

param\_name: [Table 5.18]  
CL\_PROGRAM\_BINARY\_TYPE,  
CL\_PROGRAM\_BUILD\_{STATUS, OPTIONS, LOG},  
CL\_PROGRAM\_BUILD\_GLOBAL\_VARIABLE\_TOTAL\_SIZE

(Continued on next page >)

## Program Objects (continued)

### Compiler options [5.8.6]

#### 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 and 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  
-cl-std=C++ OpenCL C++ 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:

Requires the `cl_khr_spir` extension.

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

#### Double and half-precision floating-point in C++:

-cl-fp16-enable Enable full half data type support  
cl\_khr\_fp16 macro.  
-cl-fp64-enable Enable full half data type support  
cl\_khr\_fp64 macro.

#### Linker options [5.8.7]

##### Library linking options:

-create-library -enable-link-options

##### Program linking options:

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

## Kernel Objects

A kernel object encapsulates the specific `__kernel` function and the argument values to be used when executing it.

### Create kernel objects [5.9.1]

`cl_kernel` `clCreateKernel` (`cl_program` *program*,  
const char \**kernel\_name*, `cl_int` \**errcode\_ret*)

`cl_int` `clCreateKernelsInProgram` (`cl_program` *program*,  
`cl_uint` *num\_kernels*, `cl_kernel` \**kernels*,  
`cl_uint` \**num\_kernels\_ret*)

`cl_int` `clRetainKernel` (`cl_kernel` *kernel*)

`cl_int` `clReleaseKernel` (`cl_kernel` *kernel*)

### Kernel arguments and queries [5.9.2-4]

`cl_int` `clSetKernelArg` (`cl_kernel` *kernel*,  
`cl_uint` *arg\_index*, `size_t` *arg\_size*,  
const void \**arg\_value*)

`cl_int` `clSetKernelArgSVMPointer` (`cl_kernel` *kernel*,  
`cl_uint` *arg\_index*, const void \**arg\_value*)

`cl_int` `clSetKernelExecInfo` (`cl_kernel` *kernel*,  
`cl_kernel_exec_info` *param\_name*,  
`size_t` *param\_value\_size*, const void \**param\_value*)

*param\_name*: CL\_KERNEL\_EXEC\_INFO\_SVM\_PTRS,  
CL\_KERNEL\_EXEC\_INFO\_SVM\_FINE\_GRAIN\_SYSTEM

`cl_kernel` `clCloneKernel` (`cl_kernel` *source\_kernel*,  
`cl_int` \**errcode\_ret*)

`cl_int` `clGetKernelInfo` (`cl_kernel` *kernel*,  
`cl_kernel_info` *param\_name*,  
`size_t` *param\_value\_size*, void \**param\_value*,  
`size_t` \**param\_value\_size\_ret*)

*param\_name*: [Table 5.20]

CL\_KERNEL\_FUNCTION\_NAME, NUM\_ARGS,  
CL\_KERNEL\_REFERENCE\_COUNT,  
CL\_KERNEL\_ATTRIBUTES, CONTEXT, PROGRAM)

`cl_int` `clGetKernelWorkGroupInfo` (`cl_kernel` *kernel*,  
`cl_device_id` *device*,  
`cl_kernel_work_group_info` *param\_name*,  
`size_t` *param\_value\_size*, void \**param\_value*,  
`size_t` \**param\_value\_size\_ret*)

*param\_name*: CL\_KERNEL\_GLOBAL\_WORK\_SIZE,  
CL\_KERNEL\_COMPILE\_WORK\_GROUP\_SIZE,  
CL\_KERNEL\_COMPILE\_MAX\_NUM\_SUB\_GROUPS,  
CL\_KERNEL\_LOCAL\_PRIVATE\_MEM\_SIZE,  
CL\_KERNEL\_PREFERRED\_WORK\_GROUP\_SIZE\_MULTIPLE

`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*: [Table 5.23] CL\_KERNEL\_ARG\_NAME,  
CL\_KERNEL\_ARG\_ACCESS\_ADDRESS\_QUALIFIER,  
CL\_KERNEL\_ARG\_TYPE\_NAME\_QUALIFIER

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

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

`cl_int` `clSetEventCallback` (`cl_event` *event*,  
`cl_int` *command\_exec\_callback\_type*,  
void (CL\_CALLBACK \**pf\_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*)

## Event Objects

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

### Event objects [5.11]

`cl_event` `clCreateUserEvent` (  
`cl_context` *context*, `cl_int` \**errcode\_ret*)

`cl_int` `clSetUserEventStatus` (  
`cl_event` *event*, `cl_int` *execution\_status*)

`cl_int` `clWaitForEvents` (`cl_uint` *num\_events*,  
const `cl_event` \**event\_list*)

`cl_int` `clGetEventInfo` (`cl_event` *event*,  
`cl_event_info` *param\_name*, `size_t` *param\_value\_size*,  
void \**param\_value*, `size_t` \**param\_value\_size\_ret*)

*param\_name*: [Table 5.24]

CL\_EVENT\_COMMAND\_QUEUE\_TYPE,  
CL\_EVENT\_CONTEXT\_REFERENCE\_COUNT,  
CL\_EVENT\_COMMAND\_EXECUTION\_STATUS

## Memory Model: SVM [3.3.3]

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

- **Coarse-Grained buffer SVM:** (Required) Sharing at the granularity of regions of OpenCL buffer memory objects.
- **Fine-Grained buffer SVM:** (Optional) Sharing occurs at the granularity of individual loads/stores into bytes within OpenCL buffer memory objects.
- **Fine-Grained system SVM:** Sharing occurs at the granularity of individual loads/stores into bytes occurring anywhere within the host memory.

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

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

OpenCL C++ Language Reference

Section and table references are to the OpenCL 2.2 C++ Language specification.

Supported Data Types [3.1]

Header <opengl\_def>

cl\_\* types have exactly the same size as their host counterparts defined in <cl\_platform.h> file. Half types require cl\_khr\_fp16. Double types require that cl\_khr\_fp64 be enabled and that CL\_DEVICE\_DOUBLE\_FP\_CONFIG is not zero.

Built-in scalar data types

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	cl_double	64-bit IEEE 754
half	cl_half	16-bit float (storage only)
void	void	empty set of values

Built-in vector data types

n is 2, 3, 4, 8, or 16. The halfn vector data type is required to be supported as a data storage format.

OpenCL Type	API Type	Description
booln		
[u]charn	cl_[u]charn	8-bit [un]signed
[u]shortn	cl_[u]shortn	16-bit [un]signed
[u]intn	cl_[u]intn	32-bit [un]signed
[u]longn	cl_[u]longn	64-bit [un]signed
floatn	cl_floatn	32-bit float
doublen	cl_doublen	64-bit float
halfn	cl_halfn	16-bit float

Other types [3.7.1, 3.8.1]

Header <opengl\_image>

Image and sampler types require CL\_DEVICE\_IMAGE\_SUPPORT is CL\_TRUE. See header <opengl\_pipe> for pipe type. See header <opengl\_device\_queue> for device\_queue type.

Type in OpenCL C++	API type for application
cl::sampler	cl_sampler
cl::image[1d, 2d, 3d] cl::image1d_[buffer, array] cl::image2d_ms cl::image2d_array[ms] cl::image2d_depth[ms] cl::image2d_array_depth[ms]	cl_image
cl::pipe	cl_pipe
cl::device_queue	cl_queue

Qualifiers and Optional Attributes

Function Qualifier [2.6.1]

\_\_kernel, kernel

Type and Variable Attributes [2.8]

[[cl::aligned(X)]] [[cl::aligned]]

Specifies a minimum alignment (in bytes) for variables of the specified type.

[[cl::packed]]

Specifies that each member of the structure or union is placed to minimize the memory required.

Kernel Function Attributes [2.8.3]

[[cl::work\_group\_size\_hint(X, Y, Z)]]

A hint to the compiler to specify the value most likely to be specified by the local\_work\_size argument to clEnqueueNDRangeKernel.

[[cl::required\_work\_group\_size(X, Y, Z)]]

The work-group size that must be used as the local\_work\_size argument to clEnqueueNDRangeKernel.

half wrapper [3.6.1]

Header <opengl\_half> OpenCL C++ implements a wrapper class for the built-in half data type. The class methods perform implicit vload\_half and vstore\_half operations from Vector Data Load and Store Functions section.

fp16(const half &r) noexcept;	Constructs an object with a half built-in type.
fp16(const float &r) noexcept;	Constructs an object with a float built-in type.
fp16(const double &r) noexcept;	Constructs an object with a double built-in type.

ndrange [3.13.6]

Header <opengl\_device\_queue> The ndrange type is used to represent the size of the enqueued workload with a dimension from 1 to 3.

```
struct ndrange {
    explicit ndrange(size_t global_work_size) noexcept;
    ndrange(size_t global_work_size, size_t local_work_size
             noexcept;
    ndrange(size_t global_work_offset, size_t global_work_size,
             size_t local_work_size) noexcept;
    template <size_t N>
    ndrange(const size_t (&global_work_size)[N]) noexcept;
    template <size_t N>
    ndrange(const size_t (&global_work_size)[N],
             const size_t (&global_work_size)[N]) noexcept;
    template <size_t N>
    ndrange(const size_t (&global_work_offset)[N],
             const size_t (&global_work_size)[N],
             const size_t (&global_work_size)[N]) noexcept;
};
```

Example

```
#include <opengl_device_queue>
#include <opengl_work_item>
using namespace cl;
kernel void foo(device_queue q) {
    q.enqueue_kernel(cl::enqueue_policy::no_wait, cl::ndrange(1),
                    [](){ uint tid = get_global_id(0); });
}
```

Preprocessor Directives & Macros [2.7]

```
#pragma OPENCL FP_CONTRACT on-off-switch
on-off-switch: ON, OFF, or DEFAULT
#pragma OPENCL EXTENSION extensionname : behavior
#pragma OPENCL EXTENSION all : behavior
```

__FILE__	Current source file
__LINE__	Integer line number
__OPENCL_CPP_VERSION__	Integer version number, e.g: 100
__func__	Current function name

[[cl::required\_num\_sub\_groups(X)]]

The number of sub-groups that must be generated by a kernel launch.

[[cl::vec\_type\_hint(<type>)]]

A hint to the compiler as a representation of the computational width of the kernel.

Kernel Parameter Attribute [2.8.4]

[[cl::max\_size(n)]]

The value of the attribute specifies the maximum size in bytes of the corresponding memory object.

Loop Attributes [2.8.5]

[[cl::unroll\_hint(n)]] [[cl::unroll\_hint]]

Used to specify that a loop (for, while, and do loops) can be unrolled.

[[cl::ivdep(len)]] [[cl::ivdep]]

A hint to indicate that the compiler may assume there are no memory dependencies across loop iterations in order to autovectorize consecutive iterations of loop.

OpenCL C++ and C++ 14

The OpenCL C++ programming language is based on the ISO/IEC JTC1 SC22 WG21 N3690 language (a.k.a. C++14) specification with specific restrictions and exceptions. Section numbers denoted here with § refer to the C++ 14 specification.

- Implicit conversions for pointer types follow the rules described in the C++ 14 specification.
- Conversions between integer types follow the conversion rules specified in the C++14 specification except for specific out-of-range behavior and saturated conversions.
- The preprocessing directives defined by the C++14 specification are supported.
- Macro names defined by the C++14 specification but not currently supported by OpenCL are reserved for future use.
- OpenCL C++ standard library implements modified version of the C++ 14 numeric limits library.
- OpenCL C++ implements the following parts of the C++ 14 iterator library: Primitives, iterator operations, predefined iterators, and range access.
- The OpenCL C++ kernel language doesn't support variadic functions and variable length arrays.
- OpenCL C++ library implements most of the C++14 tuples except for allocator related traits (§ 20.4.2.8).
- OpenCL C++ supports type traits defined in the C++ 14 specification with additions and changes to the following:
  - UnaryTypeTraits (§ 3.15.1)
  - BinaryTypeTraits (§ 3.15.2)
  - TransformationTraits (§ 3.15.3)
- OpenCL C++ standard library implements most C++ 14 tuples excluding allocator related traits.
- C++14 features not supported by OpenCL C++:
  - the dynamic\_cast operator (§ 5.2.7)
  - type identification (§ 5.2.8)
  - recursive function calls (§ 5.2.2, item 9) unless they are a compile-time constant expression
  - non-placement new and delete operators (§ 5.3.4, 5.3.5)
  - goto statement (§ 6.6)
  - register and thread\_local storage qualifiers (§ 7.1.1)
  - virtual function qualifier (§ 7.1.2)
  - function pointers (§ 8.3.5, 8.5.3) unless they are a compile-time constant expression
  - virtual functions and abstract classes (§ 10.3, 10.4)
  - exception handling (§ 15)
  - the C++ standard library (§ 17 ... 30)
  - asm declaration (§ 7.4)
  - no implicit lambda to function pointer conversion (§ 5.1.2)

Conversions and Reinterpretation

Header <opengl\_convert>

Conversion types [3.2]

Conversions are available for the scalar types bool, char, uchar, short, ushort, int, uint, long, ulong, half (if cl\_khr\_fp16 extension is enabled), float, double (if cl\_khr\_fp64 is enabled), and derived vector types.

```
template <class T, rounding_mode rmode, class U>
T convert_cast(U const& arg);
```

```
template <class T, rounding_mode rmode>
T convert_cast(T const& arg);
```

// and more...

Rounding modes [3.2.3]

```
::rte to nearest even      ::rtz toward zero
::rtp toward + infinity    ::rtn toward - infinity
```

Reinterpreting types [3.3]

Header <opengl\_reinterpret>

Supported data types except bool and void may be reinterpreted as another data type of the same size using the as\_type function for scalar and vector data types.

```
template <class T, class U>
T as_type(U const& arg);
```



## Vector Component Addressing [\[2.1.2.3\]](#)

### Vector Components

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
float2 v;	v.x, v.r, v.s0	v.y, v.g, v.s1														
float3 v;	v.x, v.r, v.s0	v.y, v.g, v.s1	v.z, v.b, v.s2													
float4 v;	v.x, v.r, v.s0	v.y, v.g, v.s1	v.z, v.b, v.s2	v.w, v.a, 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*. Swizzling, duplication, and nesting are allowed, e.g.: *v.yx*, *v.xx*, *v.lo.x*

	v.lo	v.hi	v.odd	v.even
float2	v.x, v.s0	v.y, v.s1	v.y, v.s1	v.x, v.s0
float3*	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz
float4	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz
float8	v.s0123	v.s4567	v.s1357	v.s0246
float16	v.s01234567	v.s89abcdef	v.s13579bdf	v.s02468ace

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

## Address Spaces Library

Header `<opencl_memory>`

### Explicit address space storage classes [\[3.4.2\]](#)

#### global<T> class

Can only be used to declare variables at program scope, with static specifier, extern specifier, or passed as a kernel argument.

#### local<T> class

Can only be used to declare variables at kernel function scope, program scope, with static keyword, extern specifier, or passed as a kernel argument.

#### priv<T> class

Cannot be used to declare variables in the program scope, with static specifier, or extern specifier.

#### constant<T> class

Can only be used to declare variables at program scope, with static specifier, extern specifier, or passed as a kernel argument.

### Explicit address space pointer classes [\[3.4.3\]](#)

The explicit address space pointer classes can be converted to and from pointers with compatible address spaces, qualifiers, and types. Local, global, and private pointers can be converted to standard C++ pointers.

```
typedef T element_type;
typedef ptrdiff_t difference_type;
typedef add_global_t<T>& reference;
typedef const add_global_t<T>& const_reference;
typedef add_global_t<T>* pointer;
typedef const add_global_t<T>* const_pointer;
```

The following pointer classes are defined in the header file `<opencl_memory>`:

template <class T> class global_ptr
template <class T> class local_ptr
template <class T> class private_ptr
template <class T> class constant_ptr

### Non-member functions [\[3.4.3.9\]](#)

In each of the partial declarations below, the placeholder *Q* may be replaced with global, local, private, or constant. The omitted initial part of each declaration is:

**template<class T, class U>**

```
bool operator==(const Q_ptr<T> &a, const Q_ptr<U> &b)
noexcept;
bool operator!=(const OP_ptr<T> &a, const Q_ptr<U> &b)
noexcept;
bool operator<(const Q_ptr<T> &a, const Q_ptr<U> &b)
noexcept;
bool operator>(const Q_ptr<T> &a, const Q_ptr<U> &b)
noexcept;
bool operator<=(const Q_ptr<T> &a, const Q_ptr<U> &b)
noexcept;
bool operator>=(const Q_ptr<T> &a, const Q_ptr<U> &b)
noexcept;
```

In each of the partial declarations below, the omitted initial part of the declaration is:

**template<class T>**

```
bool operator==(const Q_ptr<T> &x, nullptr_t) noexcept;
bool operator==(nullptr_t, const Q_ptr<T> &x) noexcept;
bool operator!=(const Q_ptr<T> &x, nullptr_t) noexcept;
```

```
bool operator!=(nullptr_t, Q_ptr global_ptr<T> &x) noexcept;
bool operator<(const Q_ptr<T> &x, nullptr_t) noexcept;
bool operator<(nullptr_t, const Q_ptr<T> &x) noexcept;
bool operator>(const Q_ptr<T> &x, nullptr_t) noexcept;
bool operator>(nullptr_t, const Q_ptr<T> &x) noexcept;
bool operator<=(const Q_ptr<T> &x, nullptr_t) noexcept;
bool operator<=(nullptr_t, const Q_ptr<T> &x) noexcept;
bool operator>=(const Q_ptr<T> &x, nullptr_t) noexcept;
bool operator>=(nullptr_t, const Q_ptr<T> &x) noexcept;
void swap(Q_ptr<T> &a, Q_ptr<T> &b) noexcept;
```

### Pointer class constructors [\[3.4.3.5\]](#)

*Q* may be global, local, private, or constant.

constexpr Q_ptr() noexcept;	Construct an object which points to nothing
explicit Q_ptr(pointer p) noexcept;	Construct an object which points to <i>p</i>
Q_ptr(const Q_ptr &) noexcept;	Copy constructor
Q_ptr(Q_ptr &&r) noexcept;	Move constructor
constexpr Q_ptr(nullptr_t) noexcept;	Construct an object initialized with nullptr

### Pointer class assignment operators [\[3.4.3.6\]](#)

*Q* may be global, local, private, or constant.

Q_ptr &operator=(const Q_ptr &r) noexcept;	Copy assignment operator
Q_ptr &operator=(Q_ptr &&r) noexcept;	Move assignment operator
Q_ptr &operator=(pointer r) noexcept;	Assign <i>r</i> pointer to the stored pointer
Q_ptr &operator=(nullptr_t) noexcept;	Assign nullptr to the stored pointer

### Pointer class observers [\[3.4.3.7\]](#)

*Q* may be global, local, private, or constant.

add_lvalue_reference_t<add_Q<T>> operator*() const noexcept;	Return <i>*get()</i>
pointer operator->() const noexcept;	Return <i>get()</i>
reference operator[](size_t pos) const noexcept;	Return <i>get()[pos]</i>
pointer get() const noexcept;	Return the stored pointer
explicit operator bool() const noexcept;	Return <i>get()!=nullptr</i>

### Pointer class modifiers [\[3.4.3.8\]](#)

*Q* may be global, local, private, or constant.

pointer release() noexcept;	Assign nullptr to the stored pointer, returns the value that <i>get()</i> had at start
void reset(pointer p = pointer()) noexcept; void reset(pointer p) noexcept;	Assign <i>p</i> to the stored pointer
void reset(nullptr_t p = nullptr) noexcept;	Equivalent to <i>reset(pointer())</i>
void swap(Q_ptr &r) noexcept;	Invokes swap on the stored pointers.

Q_ptr &operator++() noexcept; Q_ptr &operator--() noexcept;	Prefix [in/de]crement stored pointer by one.
Q_ptr operator++(int) noexcept; Q_ptr operator--(int) noexcept;	Postfix [in/de]crement stored pointer by one.
Q_ptr &operator+=( difference_type r) noexcept;	Adds <i>r</i> to the stored pointer and returns <i>*this</i> .
Q_ptr &operator-=( difference_type r) noexcept;	Subtracts <i>r</i> to the stored pointer and returns <i>*this</i> .
Q_ptr operator+( difference_type r) noexcept;	[Adds/subtracts] <i>r</i> to the stored pointer and returns the value <i>*this</i> has at the start of the operation.
Q_ptr operator-( difference_type r) noexcept;	

### Other address space functions [\[3.4.4\]](#)

template <class T> mem_fence get_mem_fence (T *ptr);	Return the <i>mem_fence</i> value for <i>ptr</i> .
template <class T, class U> T dynamic_as_cast(U *ptr);	Returns a pointer to a region in the address space pointer class specified in <i>T</i>

## Atomic Operations Library [\[3.2.4\]](#)

Header `<opencl_atomic>`

```
template<class T> struct atomic;
template<> struct atomic<integral>;
template<class T> struct atomic<T*>;
```

### enum memory\_order

*memory\_order\_x* where *x* may be relaxed, acquire, acq\_rel, seq\_cst, release

### enum memory\_scope

*memory\_scope\_x* where *x* may be work\_item, work\_group, sub\_group, all\_svm\_devices, device

### Atomic types [\[3.2.4.4\]](#)

Combined members from struct atomic, including specializations for integers (atomic<integral>) and pointers (atomic<T\*>). For struct atomic<integral>, replace *T* with integral. For struct atomic<T\*>, replace *T* with *T\**.

The pointer specialization is available if `_INTPTR_WIDTH == 32`, or both extensions `cl_khr_int64_base_extended_atomics` are enabled and `_INTPTR_WIDTH == 64`.

bool is_lock_free() const [volatile] noexcept;
void store(T, memory_order = memory_order_seq_cst, memory_scope = memory_scope_device) [volatile] noexcept;
T load(memory_order = memory_order_seq_cst, memory_scope = memory_scope_device) const [volatile] noexcept;
operator T() const [volatile] noexcept;
T exchange(T, memory_order = memory_order_seq_cst, memory_scope = memory_scope_device) [volatile] noexcept;
bool compare_exchange_weak_strong(T&, T, memory_order, memory_order, memory_scope) [volatile] noexcept;
bool compare_exchange_weak_strong(T&, T, memory_order = memory_order_seq_cst, memory_scope = memory_scope_device) [volatile] noexcept;
atomic() noexcept = default; constexpr atomic(T) noexcept;
T operator=(T) [volatile] noexcept;

(Continued on next page >)

### Atomic Operations Library (continued)

Members available in specializations `atomic<integral>` and `atomic<T*>`. For struct `atomic<integral>`, replace `T` with `integral`, and for struct `atomic<T*>`, replace `T` with `T*`. `op` may be one of add, sub, and, or, xor, min, or max.

```
T fetch_op(T, memory_order = memory_order_seq_cst,
memory_scope = memory_scope_device) [volatile] noexcept;
Ti operator[+,-,--,](int) [volatile] noexcept;
Ti operator[+,-,&,|,^](Ti) [volatile] noexcept;
```

#### Atomic types

Pointer specializations indicated with a dot are available when these extensions are enabled:

- `cl_khr_fp64`
- both `cl_khr_int64_base_extended_atomics`

```
using atomic_u[int] = atomic<u>int;
using atomic_float = atomic<float>;
• using atomic_u[long] = atomic<u>long;
• && • using atomic_double = atomic<double>;
```

Available if `_INTPTR_WIDTH == 32`, or both extensions `cl_khr_int64_base_extended_atomics` are enabled and `_INTPTR_WIDTH == 64`.

```
using atomic_intptr_t = atomic<intptr_t>;
using atomic_uintptr_t = atomic<uintptr_t>;
```

Available if `_SIZE_WIDTH == 32`, or both extensions `cl_khr_int64_base_extended_atomics` are enabled and `_SIZE_WIDTH == 64`:

```
using atomic_size_t = atomic<size_t>;
```

Available if `_PTRDIFF_WIDTH == 32`, or both extensions `cl_khr_int64_base_extended_atomics` are enabled and `_PTRDIFF_WIDTH == 64`:

```
using atomic_ptrdiff_t = atomic<ptrdiff_t>;
```

#### Members of struct atomic\_flag:

```
atomic_flag() noexcept = default;
bool test_and_set(memory_order = memory_order_seq_cst,
memory_scope = memory_scope_device) [volatile] noexcept;
void clear(memory_order = memory_order_seq_cst,
memory_scope = memory_scope_device) [volatile] noexcept;
```

#### Non-member functions:

```
bool atomic_flag_test_and_set([volatile]atomic_flag*) noexcept;
bool atomic_flag_test_and_set_explicit([volatile]atomic_flag*,
memory_order, memory_scope) noexcept;
void atomic_flag_clear([volatile]atomic_flag*) noexcept;
void atomic_flag_clear_explicit([volatile]atomic_flag*,
memory_order, memory_scope) noexcept;
```

#### Fences [3.24.6]

```
void atomic_fence(mem_fence flags,
memory_order order, memory_scope scope) noexcept;
flags: mem_fence::global, mem_fence::local,
mem_fence::image or a combination of these values
ORed together
scope: memory_scope_x where x may be all_svm_devices,
device, work_group, sub_group, work_item
```

### Images and Samplers Library [3.11]

Header `<opencl_image>`

```
struct sampler;
template <addressing_mode A, normalized_coordinates C,
filtering_mode F> constexpr sampler make_sampler();
template <class T, image_access A, image_dim Dim,
bool Depth, bool Array, bool MS> struct image;
```

#### Image types [3.11.2]

`T` is the type of value returned when reading or sampling from given image, or the type of color used to write to image.

```
using image1d = image<T, A, image_dim::image_1d, false, false,
false>;
using image1d_buffer = image<T, A, image_dim::image_buffer,
false, false, false>;
using image1d_array = image<T, A, image_dim::image_1d, false,
true, false>;
using image2d = image<T, A, image_dim::image_2d, false, false,
false>;
using image2d_depth = image<T, A, image_dim::image_2d, true,
false, false>;
using image2d_array = image<T, A, image_dim::image_2d, false,
true, false>;
using image2d_array_depth = image<T, A,
image_dim::image_2d, true, true, false>;
using image3d = image<T, A, image_dim::image_3d, false, false,
false>;
```

The extensions `cl_khr_gl_msaas_sharing` and `cl_khr_gl_depth_images` add the following functions.

```
using image2d_ms = image<T, A, image_dim::image_2d, false,
false, true>;
using image2d_array_ms = image<T, A, image_dim::image_2d,
false, true, true>;
using image2d_depth_ms = image<T, A, image_dim::image_2d,
true, false, true>;
using image2d_array_depth_ms = image<T, A,
image_dim::image_2d, true, true, true>;
```

#### Image element types [3.11.4]

In OpenCL terminology, images are classified as depth images, which have the Depth template parameter set to true, or normal images, which have the Depth template parameter set to false. Half types are only available if `cl_khr_fp16` extension is enabled.

depth images	Non-multisample depth image types: float, half For multi-sample 2D and multi-sample 2D array images, only valid type: float
normal images	Valid types: float4, int4, uint4, and half4 For multi-sample 2D and multi-sample 2D array images, only valid types: float4, int4 and uint4

#### Image dimension [3.11.5]

```
template <image_dim Dim>
struct image_dim_num;
enum image_dim
image_1d, image_2d, image_3d, image_buffer
```

#### Members of class image

Members indicated with a dot are available when these extensions are enabled:

- `cl_khr_mipmap_image_writes`
- `cl_khr_gl_msaas_sharing` and `cl_khr_gl_depth_images`

For images specified with `image_dim::image1d` and `image_dim::buffer`

```
int width() const noexcept;
• int width(float lod) const noexcept;
```

For images specified with `image_dim::image2d`

```
int [width, height]() const noexcept;
• int [width, height](float lod) const noexcept;
• int num_samples() const noexcept;
```

For images specified with `image_dim::image3d`

```
int [width, height, depth]() const noexcept;
• int [width, height, depth](float lod) const noexcept;
```

For arrayed images

```
int array_size() const noexcept;
• int array_size(int lod) const noexcept;
```

#### Image access [3.11.6]

```
enum image_access
sample, read, write, read_write
```

#### Members of class image

The non-multisample image template class specializations present different sets of methods based on their access parameter. Members indicated with a dot are available when these extensions are enabled:

- `cl_khr_mipmap_image` • `cl_khr_mipmap_image_writes`
- `cl_khr_gl_msaas_sharing` and `cl_khr_gl_depth_images`

For images specified with `image_access::read`

```
element_type image::read(integer_coord coord) const noexcept;
pixel image::operator[](integer_coord coord) const noexcept;
element_type image::pixel::operator element_type()
const noexcept;
• element_type image::read(integer_coord coord, int sample)
noexcept;
```

For images specified with `image_access::write`

```
void image::write(integer_coord coord, element_type color)
noexcept;
image::pixel image::operator[](integer_coord coord) noexcept;
image::pixel & image::pixel::operator=(element_type color)
noexcept;
• void image::write(integer_coord coord, element_type color,
int lod) noexcept;
```

For images specified with `image_access::read_write`

```
element_type image::read(integer_coord coord) const noexcept;
void image::write(integer_coord coord, element_type color)
noexcept;
image::pixel image::operator[](integer_coord coord) noexcept;
element_type image::pixel::operator element_type(
) const noexcept;
image::pixel & image::pixel::operator=(element_type color)
noexcept;
```

For images specified with `image_access::sample`

```
element_type image::read(integer_coord coord) const noexcept;
element_type image::sample(const sampler &s,
integer_coord coord) const noexcept;
element_type image::sample(const sampler &s,
float_coord coord) const noexcept;
image::pixel image::operator[](integer_coord coord)
const noexcept;
element_type image::pixel::operator element_type(
) const noexcept;
• element_type image::sample(const sampler &s,
float_coord coord, float lod) const noexcept;
• element_type image::sample(const sampler &s,
integer_coord coord, gradient_coord gradient_x,
gradient_coord gradient_y) const noexcept;
```

#### Common image methods [3.11.7]

Each image type implements this set of common members. Member indicated with a dot is available when these extensions are enabled:

```
• cl_khr_mipmap_image_writes
image_channel_type image::data_type() const noexcept;
image_channel_order image::order() const noexcept;
• int image::miplevels() const noexcept;
```

#### enum image\_channel\_type

```
snorm_int8, snorm_int16, unorm_int8,
unorm_int16, unorm_int24,
unorm_short_565, unorm_short_555,
unorm_short_101010, unorm_short_101010_2, sint8,
sint16, sint32, uint8, uint16, uint32,
float16, float32
```

#### enum image\_channel\_order

```
a, r, rx, rg, rgb, ra, rgba, argb, bgra, intensity,
luminance, abgr, srgb, srgbx, srgba, sbgra,
depth, depth_stencil
```

(Continued on next page >)

## Images and Samplers Library (continued)

### Other image methods [3.11.8]

Members indicated with a dot are available when these extensions are enabled:

- `cl_khr_mipmap_image`
- `cl_khr_mipmap_image` or `cl_khr_mipmap_image_writes`
- `cl_khr_gl_msaas_sharing` and `cl_khr_gl_depth_images`

```
element_type image::sample(const sampler &s,
float_coord coord) const noexcept;
element_type image::sample(const sampler &s,
integer_coord coord) const noexcept;
• element_type image::sample(const sampler &s,
float_coord coord, float lod) const noexcept;
• element_type image::sample(const sampler &s,
float_coord coord, gradient_coord gradient_x,
gradient_coord gradient_y) const noexcept;
element_type image::read(integer_coord coord) const noexcept;
• void image::read(integer_coord coord, int sample) noexcept;
```

```
void image::write(integer_coord coord, element_type color)
noexcept;
• void image::write(integer_coord coord, element_type color,
int lod) noexcept;
pixel operator[](integer_coord coord) noexcept;
pixel operator[](integer_coord coord) const noexcept;
element_type pixel::operator element_type() const noexcept;
pixel & pixel::operator=(element_type color) noexcept;
int width() const noexcept;
• int width(int lod) const noexcept;
int height() const noexcept;
• int height(int lod) const noexcept;
int depth() const noexcept;
• int depth(int lod) const noexcept;
int array_size() const noexcept;
• int array_size(int lod) const noexcept;
```

```
image_channel_type image::data_type() const noexcept;
image_channel_order image::order() const noexcept;
integer_coord size() const noexcept;
• int miplevels() const noexcept;
• int num_samples() const noexcept;
```

### Sampler [3.11.9]

Acquire a sampler inside of a kernel by passing it as a kernel parameter from host using `clSetKernelArg`, or creating it using the `make_sampler` function in the kernel code.

#### enum addressing\_mode

mirrored\_repeat, repeat, clamp\_to\_edge, clamp, none

#### enum normalized\_coordinates

normalized, unnormalized

#### enum normalized\_coordinates

nearest, linear

## Pipes Library

Header `<opencl_pipe>`

Use pipe and pipe\_storage template classes as a communication channel between kernels.

```
enum class pipe_access { read, write };
template <class T, pipe_access Access = pipe_access::read> struct pipe;
template <cl::pipe_access Access = pipe_access::read, class T, size_t N> pipe<T, Access>
```

### class pipe methods [3.8.4]

When pipe_access is:	Member function	Description
read	<code>bool read(T&amp; ref) const noexcept;</code>	Read packet from pipe into <code>ref</code> .
write	<code>bool write(const T&amp; ref) noexcept;</code>	Write packet specified by <code>ref</code> to pipe.
read	<code>reservation&lt;memory_scope_work_item&gt; reserve(uint num_packets) const noexcept;</code>	Reserve <code>num_packets</code> entries for reading/writing from/to pipe.
write	<code>reservation&lt;memory_scope_work_item&gt; reserve(uint num_packets) noexcept;</code>	
read	<code>reservation&lt;memory_scope_work_group&gt; work_group_reserve(uint num_packets) const noexcept;</code>	
write	<code>reservation&lt;memory_scope_work_group&gt; work_group_reserve(uint num_packets) noexcept;</code>	
read	<code>reservation&lt;memory_scope_sub_group&gt; sub_group_reserve(uint num_packets) const noexcept;</code>	
write	<code>reservation&lt;memory_scope_sub_group&gt; sub_group_reserve(uint num_packets) noexcept;</code>	
read, write	<code>uint num_packets() const noexcept;</code>	Returns current number of packets that have been written to but not yet been read from the pipe.
read, write	<code>uint max_packets() const noexcept;</code>	Returns max. number of packets specified when pipe was created.

When pipe_access is:	Member function	Description
read	<code>bool pipe::reservation::read(uint index, T&amp; ref) const noexcept;</code>	Read packet from the reserved area of the pipe referred to by index into <code>ref</code> .
write	<code>bool pipe::reservation::write(uint index, const T&amp; ref) noexcept;</code>	Write packet specified by <code>ref</code> to the reserved area of the pipe referred to by <code>index</code> .
read	<code>void pipe::reservation::commit() const noexcept;</code>	Indicates that all reads/writes to <code>num_packets</code> associated with reservation are completed.
write	<code>bool pipe::reservation::commit() noexcept;</code>	Return true if reservation is a valid reservation ID.
read	<code>bool pipe::reservation::is_valid();</code>	
write	<code>bool pipe::reservation::is_valid() const noexcept;</code>	
read, write	<code>explicit pipe::reservation::operator bool() const noexcept;</code>	

### Non-member functions

```
template<pipe_access Access = pipe_access::read,
class T, size_t N>
pipe<T, Access> make_pipe(const pipe_storage
<T, N>& ps);
```

Constructs a read only or write only pipe from pipe\_storage object.

### pipe\_storage class [3.8.5]

N in the following declaration specifies the maximum number of packets which can be held by an object.

```
template <class T, size_t N> struct pipe_storage;
```

Members of struct pipe\_storage:

```
pipe_storage();
pipe_storage(const pipe_storage&) = default;
pipe_storage(pipe_storage&&) = default;
template<pipe_access Access = pipe_access::read>
pipe<T, Access> get() noexcept;
```

Constructs a read only or write only pipe from pipe\_storage object.

## Device Enqueue Library [3.13]

Header `<opencl_device_queue>`

Allows a kernel to independently enqueue the same device, without host interaction.

### enum enqueue\_policy

no\_wait, wait\_kernel, wait\_work\_group

### enum event\_status

submitted, complete, error

### enum enqueue\_status

success, failure, invalid\_queue, invalid\_ndrange, invalid\_event\_wait\_list, queue\_full, invalid\_arg\_size, event\_allocation\_failure, out\_of\_resources

### enum event\_profiling\_info

exec\_time

### Members of struct device\_queue [3.13.3]

```
struct device_queue: marker_type;
enqueue_status
enqueue_marker(uint num_events_in_wait_list,
const event *event_wait_list, event *event_ret) noexcept;
template <class Fun, class... Args>
enqueue_status enqueue_kernel(enqueue_policy policy,
const ndrange &ndrange, Fun fun, Args... args) noexcept;
```

Enqueues a marker to device queue after a list of events specified by `event_wait_list` completes.

Enqueue functor or lambda fun on the device with specified policy over the specified `ndrange`.

```
template <class Fun, class... Args>
enqueue_status enqueue_kernel(enqueue_policy policy,
uint num_events_in_wait_list, const event *event_wait_list,
event *event_ret, const ndrange &ndrange, Fun fun,
Args... args) noexcept;
device_queue(const device_queue&) = default;
device_queue(device_queue&&) = default;
```

Enqueues functor or lambda fun in the same way as the overload above with the exception for the passed event list.

Constructors

### Members of struct event [3.13.4]

```
struct event;
bool is_valid() const noexcept;
explicit operator bool() const noexcept;
void retain() noexcept;
void release() noexcept;
void set_status(event_status status) noexcept;
void profiling_info(event_profiling_info name,
global_ptr<long> value) noexcept;
```

Returns true if event object is a valid event.

Returns true if event object is a valid event.

Increments the event reference count.

Decrements the event reference count.

Sets the execution status of a user event.

Captures the profiling information for functions that are enqueued as commands.

(Continued on next page >)



## Device Enqueue Library (continued)

### Non-member functions [3.13.5]

<code>device_queue get_default_device_queue();</code>	Returns the default device queue.
<code>event make_user_event();</code>	Creates, returns, and sets the execution status of the user event to <code>event_status::submitted</code> .
<code>template &lt;class Fun, class... Args&gt; uint get_kernel_work_group_size( Fun fun, Args... args);</code>	Provides a mechanism to query the maximum work-group size that can be used to execute a functor
<code>template &lt;class Fun, class... Args&gt; uint get_kernel_preferred_work_group_size_ multiple(Fun fun, Args... args);</code>	Returns the preferred multiple of work-group size for launch.
<code>template &lt;class Fun, class... Args&gt; uint get_kernel_sub_group_count_for_ndrange( const ndrange &amp; ndrange, Fun fun, Args... args);</code>	Returns the number of sub-groups in each work-group of the dispatch
<code>template &lt;class Fun, class... Args&gt; uint get_kernel_max_sub_group_size_for_ndrange( const ndrange &amp; ndrange, Fun fun, Args... args);</code>	Returns the maximum sub-group size for a functor.
<code>template &lt;class Fun, class... Args&gt; uint get_kernel_local_size_for_sub_group_count( uint num_sub_groups, Fun fun, Args... args);</code>	Returns a valid local size that would produce the requested number of sub-groups such that each sub-group is complete with no partial sub-groups
<code>template &lt;class Fun, class... Args&gt; uint get_kernel_max_num_sub_groups( Fun fun, Args... args);</code>	Provides a mechanism to query the maximum number of sub-groups that can be used to execute the passed functor on the current device.

### Enqueue enums [3.13.7-11]

#### enum enqueue\_policy

`no_wait`, `wait_kernel`, `wait_work_group`

#### enum enqueue\_status

`success`, `failure`, `invalid_queue`, `invalid_ndrange`, `invalid_event_wait_list`, `queue_full`, `invalid_arg_size`, `event_allocation_failure`, `out_of_resources`

#### enum event\_status

`submitted`, `complete`, `error`

#### enum event\_profiling\_info

`exec_time`

## Synchronization Functions [3.16]

Header `<opengl_synchronization>`

`struct work_group_named_barrier: marker_type;`

### Barriers [3.16.2]

<code>void work_group_barrier(mem_fence flags, memory_scope scope = memory_scope_work_group);</code>	Work-items in a work-group must execute this before any can continue
<code>void sub_group_barrier(mem_fence flags, memory_scope scope = memory_scope_work_group);</code>	Work-items in a sub-group must execute this before any can continue

*flags*: `mem_fence::global`, `mem_fence::local`, `mem_fence::image` or a combination of these values ORed together

*scope*: `memory_scope_x` where *x* may be `all_svm_devices`, `device`, `work_group`, `sub_group`, `work_item`

### Named barriers [3.16.3]

Members from `struct work_group_named_barrier`. `work_group_named_barrier` requires the `cl_khr_sub_group_named_barrier` extension be enabled.

<code>work_group_named_barrier(uint sub_group_count);</code>	
<code>work_group_named_barrier( const work_group_named_barrier&amp;) = default;</code>	Initialize a new named barrier object to synchronize <code>sub_group_count</code> sub-groups in the current work-group.
<code>work_group_named_barrier( work_group_named_barrier&amp;&amp;) = default;</code>	
<code>wait(mem_fence flags, memory_scope scope = memory_scope_work_group) const noexcept;</code>	All work-items in a sub-group executing the kernel on a processor must execute this method before any are allowed to continue.

## Work-Item Functions [3.14]

Header `<opengl_work_item>`

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.

<code>uint get_work_dim();</code>	Number of dimensions in use
<code>size_t get_global_size(uint dimindx);</code>	Number of global work-items
<code>size_t get_global_id(uint dimindx);</code>	Global work-item ID value
<code>size_t get_local_size(uint dimindx);</code>	Number of local work-items if kernel executed with uniform work-group size
<code>size_t get_enqueued_local_size(uint dimindx);</code>	Number of local work-items
<code>size_t get_local_id(uint dimindx);</code>	Local work-item ID
<code>size_t get_num_groups(uint dimindx);</code>	Number of work-groups
<code>size_t get_group_id(uint dimindx);</code>	Work-group ID
<code>size_t get_global_offset(uint dimindx);</code>	Global offset
<code>size_t get_global_linear_id();</code>	Work-items 1-dimensional global ID
<code>size_t get_local_linear_id();</code>	Work-items 1-dimensional local ID
<code>size_t get_sub_group_size();</code>	Number of work-items in the subgroup
<code>size_t get_max_sub_group_size();</code>	Maximum size of a subgroup
<code>size_t get_num_sub_groups();</code>	Number of subgroups
<code>size_t get_enqueued_num_sub_groups();</code>	If kernel executed with a uniform work-group size, results are same as for <code>get_num_sub_groups</code> .
<code>size_t get_sub_group_id();</code>	Sub-group ID
<code>size_t get_sub_group_local_id();</code>	Unique work-item ID

## Workgroup Functions [3.15]

Header `<opengl_work_group>`

### Logical operations [3.15.2]

<code>bool work_group_all (bool predicate)</code> <code>bool work_group_any (bool predicate)</code>	Evaluates predicate for all work-items in the work-group and returns true if predicate evaluates to true for all/any work-items in the work-group.
<code>bool sub_group_all (bool predicate)</code> <code>bool sub_group_any (bool predicate)</code>	Evaluates predicate for all work-items in the sub-group and returns a non-zero value if predicate evaluates to non-zero for all/any work-items in the sub-group.

### Broadcast functions [3.15.3]

*T* is type `int`, `uint`, `long`, `ulong`, or `float`, `double` (if `cl_khr_fp64` is enabled) or `half` (if `cl_khr_fp16` is enabled).

<code>T work_group_broadcast(T a, size_t local_id);</code> <code>T work_group_broadcast(T a, size_t local_id_x, size_t local_id_y);</code> <code>T work_group_broadcast(T a, size_t local_id_x, size_t local_id_y, size_t local_id_z);</code>	Broadcast the value of <i>a</i> for work-item identified by <code>local_id</code> to all work-items in the work-group.
<code>T sub_group_broadcast(T x, size_t sub_group_local_id);</code>	Broadcast the value of <i>x</i> for work-item returned by <code>get_sub_group_local_id</code> to all work-items in the sub-group.

### Numeric operations [3.15.4]

#### enum work\_group\_op

`add`, `min`, `max`

*T* is type `int`, `uint`, `long`, `ulong`, or `float`, `double` (if `cl_khr_fp64` is enabled) or `half` (if `cl_khr_fp16` is enabled).

<code>template &lt;work_group_op op&gt; T work_group_reduce(T x);</code>	Return result of reduction operation <code>&lt;op&gt;</code> for all values of <i>x</i> specified by work-items in a work-group.
<code>template &lt;work_group_op op&gt; T work_group_scan_[ex, in]clusive(T x);</code>	Perform an exclusive/inclusive scan operation <code>&lt;op&gt;</code> of all values specified by work-items in the work-group.
<code>template &lt;work_group_op op&gt; T sub_group_reduce(T x);</code>	Return result of reduction operation <code>&lt;op&gt;</code> for all values of <i>x</i> specified by work-items in a sub-group.
<code>template &lt;work_group_op op&gt; T sub_group_scan_[ex, in]clusive(T x);</code>	Perform an exclusive/inclusive scan operation <code>&lt;op&gt;</code> of all values specified by work-items in a sub-group. The scan results are returned for each work-item.

**Math Functions** [3.19]Header `<opengl_math>`

Vector versions of the math functions operate component-wise. The description is per-component.  $T$  is `halfn` (if `cl_khr_fp16` is enabled), `floatn`, or `doublen` (if `cl_khr_fp64` is enabled), where  $n$  is 2, 3, 4, 8, or 16.  $Tf$  may only be `floatn`. All angles are in radians.

**Trigonometric functions** [3.19.2]

<code>T acos (T)</code>	Arc cosine
<code>T acosh (T)</code>	Inverse hyperbolic cosine
<code>T acospi (T x)</code>	Compute $\arccos(x) / \pi$
<code>T asin (T)</code>	Arc sine
<code>T asinh (T)</code>	Inverse hyperbolic sine
<code>T asinpi (T x)</code>	Compute $\arcsin(x) / \pi$
<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>	Compute $\arctan(x) / \pi$
<code>T atan2pi (T x, T y)</code>	Compute $\arctan2(y, x) / \pi$
<code>T cos (T x)</code> <code>Tf native_math::cos(Tf x);</code> <code>Tf half_math::cos(Tf x);</code>	Cosine, $x$ is an angle
<code>T cosh (T x)</code>	Hyperbolic cosine
<code>T cospi (T x)</code>	Compute $\cos(\pi x)$
<code>T sin (T x)</code> <code>Tf native_math::sin(Tf x);</code> <code>Tf half_math::sin(Tf x);</code>	Sine, $x$ is an angle
<code>T sincos (T x, T *cosval)</code>	Sine and cosine of $x$
<code>T sinh (T x)</code>	Hyperbolic sine
<code>T sinpi (T x)</code>	$\sin(\pi x)$
<code>T tan (T x)</code> <code>Tf native_math::tan(Tf x);</code> <code>Tf half_math::tan(Tf x);</code>	Tangent
<code>T tanh (T x)</code>	Hyperbolic tangent
<code>T tanpi (T x)</code>	$\tan(\pi x)$

**Power functions** [3.19.3]

<code>T cbrt (T)</code>	Cube root
<code>T pow (T x, T y)</code>	Compute $x$ to the power of $y$
<code>floatn pown (T x, intrn y)</code>	Compute $x^y$ , where $y$ is an integer
<code>T powr (T x, T y)</code> <code>Tf native_math::powr(Tf x, Tf y);</code> <code>Tf half_math::powr(Tf x, Tf y);</code>	Compute $x^y$ , where $x \geq 0$
<code>Tf rootn (T x, intrn y)</code>	Compute $x$ to the power of $1/y$

**Integer Built-in Functions** [3.20]Header `<opengl_integer>`

$T$  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.  $Tu$  is the unsigned version of  $T$ .  $Tsc$  is the scalar version of  $T$ .

**bitwise functions** [3.20.2]

<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 popcount (T x)</code>	Number of non-zero bits in $x$
<code>T rotate (T v, T i)</code>	$\text{result}[\text{index}] = v[\text{index} \ll i][\text{index}]$
For <code>upsample</code> , return type is scalar when the parameters are scalar.	
<code>short[n] upsample (char[n] hi, uchar[n] lo)</code>	$\text{result}[i] = (((u)\text{short})\text{hi}[i] \ll 8)   \text{lo}[i]$
<code>ushort[n] upsample (uchar[n] hi, uchar[n] lo)</code>	$\text{result}[i] = ((\text{ushort})\text{hi}[i] \ll 8)   \text{lo}[i]$
<code>int[n] upsample (short[n] hi, ushort[n] lo)</code>	$\text{result}[i] = ((\text{int})\text{hi}[i] \ll 16)   \text{lo}[i]$
<code>uint[n] upsample (ushort[n] hi, ushort[n] lo)</code>	$\text{result}[i] = ((\text{uint})\text{hi}[i] \ll 16)   \text{lo}[i]$
<code>long[n] upsample (int[n] hi, uint[n] lo)</code>	$\text{result}[i] = ((\text{long})\text{hi}[i] \ll 32)   \text{lo}[i]$
<code>ulong[n] upsample (uint[n] hi, uint[n] lo)</code>	$\text{result}[i] = ((\text{ulong})\text{hi}[i] \ll 32)   \text{lo}[i]$

<code>T rsqrt (T)</code> <code>Tf native_math::rsqrt(Tf x);</code> <code>Tf half_math::rsqrt(Tf x);</code>	Inverse square root
<code>T sqrt (T)</code> <code>Tf native_math::sqrt(Tf x);</code> <code>Tf half_math::sqrt(Tf x);</code>	Square root

**Logarithmic functions** [3.19.4]

<code>intrn ilogb (T x)</code>	Return exponent as an integer value
<code>T lgamma (T x)</code> <code>T lgamma_r (T x, intrn *signp)</code>	Log gamma function
<code>T log (T)</code> <code>Tf native_math::log(Tf x);</code> <code>Tf half_math::log(Tf x);</code>	Natural logarithm
<code>T log2 (T)</code> <code>Tf native_math::log2(Tf x);</code> <code>Tf half_math::log2(Tf x);</code>	Base 2 logarithm
<code>T log10 (T)</code> <code>Tf native_math::log10(Tf x);</code> <code>Tf half_math::log10(Tf x);</code>	Base 10 logarithm
<code>T log1p (T x)</code>	Compute $\log_e(1.0 + x)$
<code>T logb (T x)</code>	Exponent of $x$

**Exponential functions** [3.19.5]

<code>T exp (T x)</code> <code>Tf native_math::exp(Tf x);</code> <code>Tf half_math::exp(Tf x);</code>	Exponential base-e exp. of $x$
<code>T exp2 (T)</code> <code>Tf native_math::exp2(Tf x);</code> <code>Tf half_math::exp2(Tf x);</code>	Exponential base 2
<code>T exp10 (T x)</code> <code>Tf native_math::exp10(Tf x);</code> <code>Tf half_math::exp10(Tf x);</code>	Exponential base 10
<code>T expm1 (T x)</code>	Compute $\exp - 1.0$
<code>T ldexp (T x, intrn k)</code>	$x * 2^k$

**Floating point functions** [3.19.6]

<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 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 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>T frexp (T x, intrn *exp)</code>	Extract mantissa and exponent

**numeric functions** [3.20.3]

<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> <code>T clamp (T x, Tsc min, Tsc max)</code>	$\min(\max(x, \text{minval}), \text{maxval})$
<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> <code>T max (T x, Tsc y)</code>	$y$ if $x < y$ , otherwise it returns $x$
<code>T min (T x, T y)</code> <code>T min (T x, Tsc y)</code>	$y$ if $y < x$ , otherwise it returns $x$
<code>T mul_hi (T x, T y)</code>	High half of the product of $x$ and $y$
<code>T sub_sat (T x, T y)</code>	$x - y$ and saturates the result

**24-bit operations** [3.20.4]

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

<code>intrn 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 $y$

<code>T modf (T x, T *iptr)</code>	Decompose floating-point number
<code>floatn nan (uintn nancode)</code> <code>doublen nan (ulongn nancode)</code> <code>halfn nan (ushortn nancode)</code>	Quiet NaN
<code>T nextafter (T x, T y)</code>	Next representable floating-point value after $x$ in the direction of $y$
<code>T remainder (T x, T y)</code>	Floating point remainder
<code>T remquo (T x, T y, intrn *quo)</code>	Remainder and quotient
<code>T rint (T x)</code>	Round to nearest even integer
<code>T round (T x)</code>	Integral value nearest to $x$ rounding
<code>T trunc (T x)</code>	Return integral value nearest to $x$ rounding halfway cases away from zero.

**Comparison functions** [3.19.7]

<code>T fdim (T x, T y)</code>	Positive difference between $x$ and $y$
<code>T fmax (T x, T y)</code>	Return $y$ if $x < y$ , else returns $x$
<code>T fmin (T x, T y)</code>	Return $y$ if $y < x$ , else returns $x$
<code>T fmod (T x, T y)</code>	Modulus. Returns $x - y * \text{trunc}(x/y)$
<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 $y$

**Other functions** [3.19.8]

<code>Tf native_math::divide(Tf x, Tf y);</code> <code>Tf half_math::divide(Tf x, Tf y);</code>	Compute $x / y$
<code>T erfc (T)</code>	Complementary error function.
<code>T erf (T x)</code>	Calculates error function of $T$
<code>T fabs (T x)</code>	Absolute value
<code>T hypot (T x, T y)</code>	Square root of $x^2 + y^2$
<code>T mad (T a, T b, T c)</code>	Approximates $a * b + c$
<code>Tf native_math::recip(Tf x);</code> <code>Tf half_math::recip(Tf x);</code>	Reciprocal
<code>T tgamma (T x)</code>	Gamma function

**Common Functions** [3.17]Header `<opengl_common>`

These functions are implemented using the round to nearest even rounding mode. Vector versions operate component-wise.  $Ts$  is type `float`, optionally `double` (if `cl_khr_fp64` is enabled), or half if `cl_khr_fp16` is enabled.  $Tn$  is the vector form of  $Ts$ , where  $n$  is 2, 3, 4, 8, or 16.  $T$  is  $Ts$  and  $Tn$ .

<code>T clamp (T x, T min, T max)</code>	Clamp $x$ to range given by $\text{min}, \text{max}$
<code>T degrees (T radians)</code>	$\text{radians}$ to degrees
<code>T max (T x, T y)</code>	Max of $x$ and $y$
<code>T min (T x, T y)</code>	Min of $x$ and $y$
<code>T mix (T x, T y, T a)</code>	Linear blend of $x$ and $y$
<code>T radians (T degrees)</code>	$\text{degrees}$ to radians
<code>T step (T edge, T x)</code>	0.0 if $x < \text{edge}$ , else 1.0
<code>T smoothstep (T edge0, T edge1, T x)</code>	Step and interpolate
<code>T sign (T x)</code>	Sign of $x$

**Geometric Functions [3.18]**

Header `<opengl_geometric>`

These functions use the round to nearest even rounding mode. Vector versions operate component-wise. *Ts* is scalar type float, double if `cl_khr_fp64` is enabled, or half if `cl_khr_fp16` is enabled. *Tn* is the vector form of *Ts* with 2, 3, or 4 components.

<code>float{3,4} cross (float{3,4} p0, float{3,4} p1)</code> <code>double{3,4} cross (double{3,4} p0, double{3,4} p1)</code> <code>half{3,4} cross (half{3,4} p0, half{3,4} p1)</code>	Cross product
<code>Ts distance (T p0, T p1)</code>	Vector distance
<code>Ts dot (T p0, T p1)</code>	Dot product
<code>Ts length (T p)</code>	Vector length
<code>T normalize (T p)</code>	Normal vector length 1

**Vector Data Load/Store [3.22]**

Header `<opengl_vector_load_store>`

*T* is type char, uchar, short, ushort, int, uint, long, ulong, or float, optionally double (if `cl_khr_fp64` is enabled), or half (if `cl_khr_fp16` is enabled). *Tn* refers to the vector form of type *T*, where *n* is 2, 3, 4, 8, or 16.

<code>template &lt;size_t N, class T&gt; make_vector_t&lt;T, N&gt; vload(size_t offset, const T* p);</code>	Read vector data from address ( $p + (offset * n)$ )
<code>template &lt;size_t N, class T&gt; make_vector_t&lt;T, N&gt; vload(size_t offset, const constant_ptr&lt;T&gt; p);</code>	
<code>template &lt;size_t N&gt; make_vector_t&lt;float, N&gt; vload_half(size_t offset, const half* p);</code>	Read a halfn from address ( $p + (offset * n)$ )
<code>template &lt;size_t N&gt; make_vector_t&lt;float, N&gt; vload_half(size_t offset, const constant_ptr&lt;half&gt; p);</code>	
<code>template &lt;size_t N&gt; make_vector_t&lt;float, N&gt; vload_half(size_t offset, const half* p);</code>	Read half vector from ( $p + (offset * n)$ ). For half3, read from ( $p + (offset * 4)$ ).
<code>template &lt;size_t N&gt; make_vector_t&lt;float, N&gt; vload_half(size_t offset, const constant_ptr&lt;half&gt; p);</code>	
<code>template &lt;class T&gt; void vstore(T data, size_t offset, vector_element_t&lt;T&gt;* p);</code>	Write vector data to address ( $p + (offset * n)$ )
<code>template &lt;rounding_mode rmode = rounding_mode::rte, class T&gt; void vstore_half(T data, size_t offset, half* p);</code>	Write a half to address ( $p + offset$ )
<code>template &lt;rounding_mode rmode = rounding_mode::rte, class T&gt; void vstorea_half(T data, size_t offset, half* p);</code>	Write a half vector to address ( $p + (offset * n)$ )

**Array Library [3.25]**

Header `<opengl_array>`

`template<class T, size_t N> struct array;`

**Iterators from struct array**

`[const_]iterator begin()` [const] noexcept;  
`[const_]iterator end()` [const] noexcept;  
`[const_]reverse_iterator rbegin()` [const] noexcept;  
`[const_]reverse_iterator rend()` [const] noexcept;  
`const_iterator cbegin()` const noexcept;  
`const_iterator cend()` const noexcept;  
`const_reverse_iterator crbegin()` const noexcept;  
`const_reverse_iterator crend()` const noexcept;

**Capacities from struct array**

`constexpr size_type size()` const noexcept;  
`constexpr size_type max_size()` const noexcept;  
`constexpr bool empty()` const noexcept;

**Relational Built-in Functions [3.21]**

Header `<opengl_relational>`

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 (if `cl_khr_fp64` is enabled) or half or halfn (if `cl_khr_fp16` is enabled). *n* is 2, 3, 4, 8, or 16.

<code>booln isequal(floatn x, floatn y);</code> <code>booln isequal(halfn x, halfn y);</code> <code>booln isequal(doublen x, doublen y);</code>	Compare of $x == y$
<code>booln isnotequal(floatn x, floatn y);</code>	Compare of $x != y$
<code>booln isgreater(floatn x, floatn y);</code>	Compare of $x > y$
<code>booln isgreaterequal(floatn x, floatn y);</code>	Compare of $x >= y$
<code>booln isless(floatn x, floatn y);</code>	Compare of $x < y$
<code>booln islessequal(floatn x, floatn y);</code>	Compare of $x <= y$
<code>booln islessgreater(floatn x, floatn y);</code>	Compare of $(x < y)    (x > y)$
<code>booln isordered(floatn x, floatn y);</code>	Test if arguments are ordered
<code>booln isunordered(floatn x, floatn y);</code>	Test if arguments are unordered
<code>booln isfinite(floatn x, floatn y);</code>	Test for finite value

<code>booln isinf(floatn x, floatn y);</code>	Test for + or - infinity
<code>booln isnan(floatn x, floatn y);</code>	Test for a NaN
<code>booln isnormal(floatn x, floatn y);</code>	Test for a normal value
<code>booln signbit(floatn x, floatn y);</code>	Test for sign bit
<code>bool any(booln t);</code>	1 if MSB in component of <i>x</i> is set; else 0
<code>bool all(booln t);</code>	1 if MSB in all components of <i>x</i> are set; else 0
<code>T biselect(T a, T b, T c);</code>	Each bit of result is corresponding bit of <i>a</i> if corresponding bit of <i>c</i> is 0
<code>T select(T a, T b, booln c);</code>	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>

**printf Function [3.23]**

Header `<opengl_printf>`

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**

`[%[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);
uchar4 uc = uchar4(0xfA, 0xFB, 0xFC, 0xFD);
printf("f4 = %2.2v4hf\n", f);
printf("uc = %v4hhx\n", uc);
```

The above two printf calls print the following:

```
f4 = 1.00,2.00,3.00,4.00
uc = 0xfa,0xfb,0xfc,0xfd
```

**Limits [3.26]**

Header `<opengl_limits>`

Half is available if `cl_khr_fp16` is enabled, and double is available if `cl_khr_fp64` is enabled.

**Floating point limits**

OpenCL C++ Macros ( <i>x</i> is HALF, FLT, DBL)	HALF	FLT	DBL	Application Macro ( <i>x</i> is HALF, FLT, DBL)
<code>x_DIG</code>	3	6	15	<code>CL_x_DIG</code>
<code>x_MANT_DIG</code>	11	24	53	<code>CL_x_MANT_DIG</code>
<code>x_MAX_10_EXP + 4</code>	+4	+38	+308	<code>CL_x_MAX_10_EXP</code>
<code>x_MAX_EXP</code>	+16	+128	+1024	<code>CL_x_MAX_EXP</code>
<code>x_MIN_10_EXP</code>	-4	-37	-307	<code>CL_x_MIN_10_EXP</code>
<code>x_MIN_EXP</code>	-13	-125	-1021	<code>CL_x_MIN_EXP</code>
<code>x_RADIX</code>	2	2	2	<code>CL_x_RADIX</code>
<code>x_MAX</code>	0x1.ffcp15h	0x1.fffffp127f	0x1.fffffffffffffp1023	<code>CL_x_MAX</code>
<code>x_MIN</code>	0x1.0p-14h	0x1.0p-126f	0x1.0p-1022	<code>CL_x_MIN</code>
<code>x_EPSILON</code>	0x1.0p-10h	0x1.0p-23f	0x1.0p-52	<code>CL_x_EPSILON</code>

**enum float\_round\_style**

`round_indeterminate, round_toward_zero, round_to_nearest, round_toward_infinity, round_toward_neg_infinity`

**enum float\_denorm\_style**

`denorm_indeterminate, denorm_absent, denorm_present`

**Integer limits**

```
#define CHAR_BIT 8
#define CHAR_MAX SCHAR_MAX
#define CHAR_MIN SCHAR_MIN
#define INT_MAX 2147483647
#define INT_MIN (-2147483647 - 1)

#define LONG_MAX 0x7fffffffffffffffL
#define LONG_MIN (-0x7fffffffffffffffL - 1)
#define SCHAR_MAX 127
#define SCHAR_MIN (-127 - 1)
#define SHRT_MAX 32767

#define SHRT_MIN (-32767 - 1)
#define UCHAR_MAX 255
#define USHRT_MAX 65535
#define UINT_MAX 0xffffffff
#define ULONG_MAX 0xffffffffffffffffULL
```

(Continued on next page >)



## Limits (continued)

## Class numeric\_limits [3.26.2]

template<class T> class numeric\_limits;  
All the members below are declared as static constexpr.

```
bool is_signed = false;
T min() noexcept { return T(); }
T max() noexcept { return T(); }
T lowest() noexcept { return T(); }
int digits = 0;
int digits10 = 0;
int max_digits10 = 0;
bool is_signed = false;
bool is_integer = false;
bool is_exact = false;
int radix = 0;
T epsilon() noexcept { return T(); }
T round_error() noexcept { return T(); }
int min_exponent = 0;
int min_exponent10 = 0;
int max_exponent = 0;
int max_exponent10 = 0;
bool has_infinity = false;
bool has_quiet_NaN = false;

bool has_signaling_NaN = false;
float_denorm_style has_denorm = denorm_absent;
bool has_denorm_loss = false;
T infinity() noexcept { return T(); }
T quiet_NaN() noexcept { return T(); }
T signaling_NaN() noexcept { return T(); }
T denorm_min() noexcept { return T(); }
bool is_iec559 = false;
bool is_bounded = false;
bool is_modulo = false;
bool traps = false;
bool tinyness_before = false;
float_round_style round_style = round_toward_zero;
bool is_scalar = false;
bool is_vector = false;
```

## Non-members

```
template<class T> class numeric_limits<const T>;
template<class T> class numeric_limits<volatile T>;
template<class T> class numeric_limits<const volatile T>;
```

## Math Constants [3.27]

Header &lt;opengl\_math\_constants&gt;

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

Constant	Description
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

```
template<class T> class math_constants;
template<> class math_constants<halfn>;
template<> class math_constants<floatn>;
template<> class math_constants<doublen>;
```

## Tuple Library [3.28]

Header &lt;opengl\_tuple&gt;

```
template<class... Types> class tuple;
```

<b>Tuple creation functions</b>	make_tuple() tie()	forward_as_tuple() tuple_cat()
<b>Tuple helper classes</b>	class tuple_size	class tuple_element
<b>Element access</b>	get()	
<b>Relational operators</b>	operator==(const tuple<Types...>& t1, const tuple<Types...>& t2)	operator<(const tuple<Types...>& t1, const tuple<Types...>& t2)
<b>Specialized algorithms</b>	swap()	

## Constants, functions, and macros

The preprocessor macros in the table below are shown for double and are available if `cl_khr_fp64` is enabled. Append `_F` for float, or append `_H` for half if `cl_khr_fp16` is enabled.

Name of constant	FuncName	Preprocessor macros
e	e()	M_E
log2e	log2e()	M_LOG2E
log10e	log10e()	M_LOG10E
ln2	ln2()	M_LN2
ln10	ln10()	M_LN10
pi	pi()	M_PI
pi_2	pi_2()	M_PI_2
pi_4	pi_4()	M_PI_4
one_pi	one_pi()	M_1_PI
two_pi	two_pi()	M_2_PI
two_sqrtpi	two_sqrtpi()	M_2_SQRTPI
sqrt2	sqrt2()	M_SQRT2
sqrt1_2	sqrt1_2()	M_SQRT1_2

Replace the placeholders in the templates below with values from the indicated column in the table above.

```
template<class T> constexpr T Constant_v =
    math_constants<T>::FuncName;

template<class T> class math_constants;
    static constexpr T FuncName noexcept { return T(); }
```

Examples:

```
template<class T> constexpr T pi_v = math_constants<T>::pi();
template<class T> class math_constants;
    static constexpr T pi() noexcept { return T(); }
```

## Type Traits Library [3.29]

Header &lt;opengl\_type\_traits&gt;

```
template<class... Types> class tuple;
```

## Primary type categories

```
is_void is_null_pointer
is_integral is_floating_point
is_array is_pointer
is_enum is_union
is_class is_function
is_lvalue_reference
is_rvalue_reference
is_member_object_pointer
is_member_function_pointer
```

## Composite type categories

```
is_reference is_arithmetic
is_object is_fundamental
is_scalar is_compound
is_member_pointer
```

## Type property queries

```
alignment_of rank extent
```

## Type relations

```
is_same
is_base_of
is_convertible
```

## Const-volatile modifications

```
remove_const add_const
remove_volatile add_volatile
remove_cv add_cv
```

## As modifications

```
remove_as remove_attrs
add_constant remove_constant
add_local remove_local
add_global remove_global
add_private remove_private
add_generic remove_generic
```

## Reference modifications

```
remove_reference
add_lvalue_reference
add_rvalue_reference
```

## Sign modifications

```
make_signed make_unsigned
```

## Array modifications

```
remove_extent remove_all_extents
```

## Pointer modifications

```
add_pointer remove_pointer
```

## Built-in vector queries

```
vector_size is_vector_type
```

## Built-in vector modifications

```
vector_element
make_vector
```

## Other transformations

```
aligned_storage aligned_union
decay enable_if
common_type underlying_type
conditional result_of
```

## Type properties

```
is_const is_volatile
is_private is_local
is_global is_constant
is_generic is_vector
is_trivial is_trivially_copyable
is_pod is_literal_type
is_empty is_polymorphic
is_abstract is_final
is_signed is_unsigned
is_standard_layout
is_[trivially_]constructible
is_[trivially_]default_constructible
is_[trivially_]copy_constructible
is_[trivially_]move_constructible
is_[trivially_]assignable
is_[trivially_]copy_assignable
is_[trivially_]move_assignable
is_[trivially_]nothrow_destructible
is_nothrow_[default_]constructible
is_nothrow_[copy_move_]constructible
is_nothrow_[copy_move_]assignable
has_virtual_destructor
```

## Iterator Library [3.30]

Header &lt;opengl\_iterator&gt;

```
template<class Category, class T,
        class Distance = ptrdiff_t,
        class Pointer = T*,
        class Reference = T&> struct iterator;
```

## Iterator operations

```
advance() distance()
next() prev()
```

## Tags

```
input_iterator_tag
output_iterator_tag
forward_iterator_tag
bidirectional_iterator_tag
random_access_iterator_tag
```

## Range access

```
begin() cbegin() rbegin() crbegin()
end() cend() rend() crend()
```

## Predefined iterators

```
inserter() front_inserter() back_inserter()
make_reverse_iterator()
make_move_iterator()
operatorOP() where OP may be ==, !=, <, >, <=, >=, +, -
```

## Vector Wrapper Library [3.7]

Header &lt;opengl\_vec&gt;

```
template<class T, size_t Size> struct vec;
```

## struct vec members

```
vec() = default;
vec(const vec &) = default;
```

```
vec(vec &&) = default;
vec(const vector_type &r) noexcept;
vec(vector_type &&r) noexcept;
template<class... Params>
    vec(Params... params) noexcept;

operator vector_type() const noexcept;
operatorOP() where OP may be =, +=, -=, +=,
    -=, *=, /=, %=

swizzle()
```

## Simple swizzles

If preprocessor macro SIMPLE\_SWIZZLES is defined, then:

```
auto func() noexcept; where func may
    be x through zzzz
```

## Non-member operators

```
operatorOP() where OP may be ==, !=, <, >,
    <=, >=, +, -, *, /
```

## Vector Utilities [3.9]

Header &lt;opengl\_vector\_utility&gt;

```
template<size_t Channel, class Vec>
constexpr remove_attrs_t
    <vector_element_t<Vec>>
    get(Vec & vector) noexcept;

template<size_t Channel, class Vec>
constexpr void set(Vec & vector,
    remove_attrs_t<vector_element_t<Vec>>
    value) noexcept;
```

## struct channel\_ref members

```
operatorOP() where OP may be =, +=, -=, +=, -=,
    *=, /=, %=
```

OpenCL C Language Reference

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

Supported Data Types

Half vector and scalar types require *cl\_khr\_fp16*. Double types require that *CL\_DEVICE\_DOUBLE\_FP\_CONFIG* is not zero.

Built-in Scalar Data Types [6.1.1]

Table with 3 columns: OpenCL Type, API Type, Description. Lists types like bool, char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, float, double, half, size\_t, ptrdiff\_t, intptr\_t, uintptr\_t, void.

Built-in Vector Data Types [6.1.2]

n is 2, 3, 4, 8, or 16.

Table with 3 columns: OpenCL Type, API Type, Description. Lists vector types like [u]char^n, [u]short^n, [u]int^n, [u]long^n, float^n, double^n.

Other Built-in Data Types [6.1.3]

The OPTIONAL types shown below are only defined if CL\_DEVICE\_IMAGE\_SUPPORT is CL\_TRUE. API type for application shown in italics where applicable. Items in blue require the cl\_khr\_gl\_msaa\_sharing extension.

Table with 2 columns: OpenCL Type, Description. Lists image and sampler types like image2d\_[msaa\_]t, image3d\_t, image2d\_array\_[msaa\_]t, image1d\_t, image1d\_buffer\_t, image1d\_array\_t, image2d\_[msaa\_]depth\_t, image2d\_array\_[msaa\_]depth\_t, sampler\_t, queue\_t.

Table with 2 columns: OpenCL Type, Description. Lists event types like ndrange\_t, clk\_event\_t, reserve\_id\_t, event\_t, cl\_mem\_fence\_flags.

Reserved Data Types [6.1.4]

Table with 2 columns: OpenCL Type, Description. Lists reserved types like bool^n, half^n, quad, quad^n, complex half, complex half^n, imaginary half, imaginary half^n, complex float, complex float^n, imaginary float, imaginary float^n, complex double, complex double^n, imaginary double, imaginary double^n, complex quad, complex quad^n, imaginary quad, imaginary quad^n, float^m x n, double^m x n.

Vector Component Addressing [6.1.7]

Vector Components

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

Vector Addressing Equivalences

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

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

Operators and Qualifiers

Operators [6.3]

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

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

Address Space Qualifiers [6.5]

Table of address space qualifiers: \_global, global, \_local, local, \_constant, constant, \_private, private.

Function Qualifiers [6.7]

Table of function qualifiers: \_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]

Table of preprocessor directives and macros: #pragma OPENCL\_FP\_CONTRACT, \_FILE\_, \_func\_, \_LINE\_, \_OPENCL\_VERSION\_, CL\_VERSION\_1\_0, CL\_VERSION\_1\_1, CL\_VERSION\_1\_2, CL\_VERSION\_2\_0, \_OPENCL\_C\_VERSION\_, \_ENDIAN\_LITTLE\_, \_IMAGE\_SUPPORT\_, \_FAST\_RELAXED\_MATH\_, FP\_FAST\_FMA.

Table of macros: FP\_FAST\_FMAF, FP\_FAST\_FMA\_HALF, \_kernel\_exec(X, typen), \_kernel\_attribute\_\_((work\_group\_size\_hint(X, 1, 1))), \_attribute\_\_((vec\_type\_hint(typen))).

Conversions, Type Casting Examples [6.2]

T a = (T)b; // Scalar to scalar, or scalar to vector
T a = convert\_T(b);
T a = convert\_T\_R(b);
T a = as\_T(b);
T a = convert\_T\_sat\_R(b);

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

Attribute Qualifiers [6.11]

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

Table of attribute qualifiers: \_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.

Table of attribute qualifiers: \_attribute\_\_((aligned(alignment))), \_attribute\_\_((nosvm)).

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

Table of attribute qualifiers: \_attribute\_\_((attr1)) {...}

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

Table of attribute qualifiers: \_attribute\_\_((opencl\_unroll\_hint(n))), \_attribute\_\_((opencl\_unroll\_hint)).

### 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`  
`__read_write`, `read_write`

### Blocks [6.12]

A result value type with a list of parameter types: for example:

- The ^ declares variable "myBlock" is a Block.
- The return type for the Block "myBlock" is int.
- myBlock takes a single argument of type int.
- The argument is named "num."
- Multiplier captured from block's environment.

```

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

### Work-Item Built-in Functions [6.13.1]

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

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

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

### Math Built-in Functions [6.13.2]

*Ts* is type float, optionally double (if `cl_khr_fp64` is enabled), or half (if `cl_khr_fp16` is enabled). *Tn* is the vector form of *Ts*, where *n* is 2, 3, 4, 8, or 16. *T* is *Ts* and *Tn*. All angles are in radians.

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

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

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

<code>T sin (T)</code> <b>HN</b>	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> <b>HN</b>	Square root
<code>T tan (T)</code> <b>HN</b>	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]

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 <small>OPTIONAL</small>
INFINITY	Constant float expression, positive or unsigned infinity
NAN	Constant float expression, quiet NaN

When double precision is supported (if `cl_khr_fp64` is enabled) macros ending in `_F` are available in type double by removing `_F` from the macro name, and in type half (if `cl_khr_fp16` is enabled) by replacing `_F` with `_H`.

<code>M_E_F</code>	Value of $e$
<code>M_LOG2E_F</code>	Value of $\log_2 e$
<code>M_LOG10E_F</code>	Value of $\log_{10} e$
<code>M_LN2_F</code>	Value of $\log_e 2$
<code>M_LN10_F</code>	Value of $\log_e 10$
<code>M_PI_F</code>	Value of $\pi$
<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 $1 / \sqrt{2}$



**Image Read and Write Functions** [6.13.14]

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

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

**Read and write functions for 2D images**

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

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

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

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

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

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

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

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

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

float4 read_imagef (aQual image2d_t image, int2 coord)
int4 read_imagei (aQual image2d_t image, int2 coord)
uint4 read_imageui (aQual image2d_t image, int2 coord)

float4 read_imagef (aQual image2d_array_t image, int4 coord)
int4 read_imagei (aQual image2d_array_t image, int4 coord)
uint4 read_imageui (aQual image2d_array_t image, int4 coord)

float read_imagef (aQual image2d_depth_t image, int2 coord)
float read_imagef (aQual image2d_array_depth_t image,
                  int4 coord)

half4 read_imageh (read_only image2d_t image,
                  sampler_t sampler, {int2, float2} coord)
half4 read_imageh (aQual image2d_t image, int2 coord)
half4 read_imageh (read_only image2d_array_t image,
                  sampler_t sampler, {int4, float4} coord)
half4 read_imageh (aQual image2d_array_t image,
                  int4 coord)

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

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

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

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

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

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

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

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

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

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

**Read and write functions for 1D images**

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

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

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

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

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

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

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

float4 read_imagef (aQual image1d_t image, int coord)

float4 read_imagef (aQual image1d_buffer_t image, int coord)

int4 read_imagei (aQual image1d_t image, int coord)
uint4 read_imageui (aQual image1d_t image, int coord)
int4 read_imagei (aQual image1d_buffer_t image, int coord)
uint4 read_imageui (aQual image1d_buffer_t image, int coord)

float4 read_imagef (aQual image1d_array_t image, int2 coord)
int4 read_imagei (aQual image1d_array_t image, int2 coord)
uint4 read_imageui (aQual image1d_array_t image, int2 coord)

half4 read_imageh (read_only image1d_t image,
                  sampler_t sampler, {int, float} coord)
half4 read_imageh (aQual image1d_t image, int coord)
half4 read_imageh (read_only image1d_array_t image,
                  sampler_t sampler, {int2, float2} coord)
half4 read_imageh (aQual image1d_array_t image, int2 coord)
half4 read_imageh (aQual image1d_buffer_t image, int coord)

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

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

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

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

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

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

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

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

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

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

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

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

**Read and write functions for 3D images**

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

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

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

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

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

float4 read_imagef (aQual image3d_t image, int4 coord)
int4 read_imagei (aQual image3d_t image, int4 coord)
uint4 read_imageui (aQual image3d_t image, int4 coord)

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

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

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

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

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

**Extended mipmap read and write functions**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(Continued on next page &gt;)



**Integer Built-in Functions [6.13.3]**

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

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

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

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

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

**Common Built-in Functions [6.13.4]**

These functions operate component-wise and use round to nearest even rounding mode. *Ts* is type float, optionally double (if *cl\_khr\_fp64* is enabled) or half and half<sub>n</sub> (if *cl\_khr\_fp16* is enabled). *Tn* is the vector form of *Ts*, where *n* is 2, 3, 4, 8, or 16. *T* is *Ts* and *Tn*.

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

**Relational Built-in Functions [6.13.6]**

These functions can be used with built-in scalar or vector types as arguments and return a scalar or vector integer result. *T* is type float, float<sub>n</sub>, char, charn, uchar, uchar<sub>n</sub>, short, short<sub>n</sub>, ushort, ushort<sub>n</sub>, int, int<sub>n</sub>, uint, uint<sub>n</sub>, long, long<sub>n</sub>, ulong, or ulong<sub>n</sub>, or optionally double or double<sub>n</sub> (if *cl\_khr\_fp64* is enabled) or half and half<sub>n</sub> (if *cl\_khr\_fp16* is enabled). *Ti* is type char, charn, short, short<sub>n</sub>, int, int<sub>n</sub>, long, or long<sub>n</sub>. *Tu* is type uchar, uchar<sub>n</sub>, ushort, ushort<sub>n</sub>, uint, uint<sub>n</sub>, ulong, or ulong<sub>n</sub>. *n* is 2, 3, 4, 8, or 16.

int isequal (float <i>x</i> , float <i>y</i> ) intn isequal (float <sub>n</sub> <i>x</i> , float <sub>n</sub> <i>y</i> ) int isequal (double <i>x</i> , double <i>y</i> ) longn isequal (double <sub>n</sub> <i>x</i> , double <sub>n</sub> <i>y</i> ) int isequal (half <i>x</i> , half <i>y</i> ) shortn isequal (half <sub>n</sub> <i>x</i> , half <sub>n</sub> <i>y</i> )	Compare of <i>x</i> == <i>y</i>
int isnotequal (float <i>x</i> , float <i>y</i> ) intn isnotequal (float <sub>n</sub> <i>x</i> , float <sub>n</sub> <i>y</i> ) int isnotequal (double <i>x</i> , double <i>y</i> ) longn isnotequal (double <sub>n</sub> <i>x</i> , double <sub>n</sub> <i>y</i> ) int isnotequal (half <i>x</i> , half <i>y</i> ) shortn isnotequal (half <sub>n</sub> <i>x</i> , half <sub>n</sub> <i>y</i> )	Compare of <i>x</i> != <i>y</i>
int isgreater (float <i>x</i> , float <i>y</i> ) intn isgreater (float <sub>n</sub> <i>x</i> , float <sub>n</sub> <i>y</i> ) int isgreater (double <i>x</i> , double <i>y</i> ) longn isgreater (double <sub>n</sub> <i>x</i> , double <sub>n</sub> <i>y</i> ) int isgreater (half <i>x</i> , half <i>y</i> ) shortn isgreater (half <sub>n</sub> <i>x</i> , half <sub>n</sub> <i>y</i> )	Compare of <i>x</i> > <i>y</i>
int isgreaterequal (float <i>x</i> , float <i>y</i> ) intn isgreaterequal (float <sub>n</sub> <i>x</i> , float <sub>n</sub> <i>y</i> ) int isgreaterequal (double <i>x</i> , double <i>y</i> )	Compare of <i>x</i> >= <i>y</i>
longn isgreaterequal (double <sub>n</sub> <i>x</i> , double <sub>n</sub> <i>y</i> ) int isgreaterequal (half <i>x</i> , half <i>y</i> ) shortn isgreaterequal (half <sub>n</sub> <i>x</i> , half <sub>n</sub> <i>y</i> )	Compare of <i>x</i> >= <i>y</i>
int isless (float <i>x</i> , float <i>y</i> ) intn isless (float <sub>n</sub> <i>x</i> , float <sub>n</sub> <i>y</i> ) int isless (double <i>x</i> , double <i>y</i> )	Compare of <i>x</i> < <i>y</i>

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

longn isnormal (double <sub>n</sub> ) int isnormal (half) shortn isnormal (half <sub>n</sub> )	Test for a normal value
int isordered (float <i>x</i> , float <i>y</i> ) intn isordered (float <sub>n</sub> <i>x</i> , float <sub>n</sub> <i>y</i> ) int isordered (double <i>x</i> , double <i>y</i> ) longn isordered (double <sub>n</sub> <i>x</i> , double <sub>n</sub> <i>y</i> ) int isordered (half <i>x</i> , half <i>y</i> ) shortn isordered (half <sub>n</sub> <i>x</i> , half <sub>n</sub> <i>y</i> )	Test if arguments are ordered
int isunordered (float <i>x</i> , float <i>y</i> ) intn isunordered (float <sub>n</sub> <i>x</i> , float <sub>n</sub> <i>y</i> ) int isunordered (double <i>x</i> , double <i>y</i> ) longn isunordered (double <sub>n</sub> <i>x</i> , double <sub>n</sub> <i>y</i> ) int isunordered (half <i>x</i> , half <i>y</i> ) shortn isunordered (half <sub>n</sub> <i>x</i> , half <sub>n</sub> <i>y</i> )	Test if arguments are unordered
int signbit (float) intn signbit (float <sub>n</sub> ) int signbit (double) longn signbit (double <sub>n</sub> ) int signbit (half) shortn signbit (half <sub>n</sub> )	Test for sign bit
int any ( <i>Ti</i> <i>x</i> )	1 if MSB in component of <i>x</i> is set; else 0
int all ( <i>Ti</i> <i>x</i> )	1 if MSB in all components of <i>x</i> are set; else 0
<i>T</i> bitselect ( <i>Ta</i> , <i>Tb</i> , <i>Tc</i> ) half bitselect (half <i>a</i> , half <i>b</i> , half <i>c</i> ) halfn bitselect (half <sub>n</sub> <i>a</i> , half <sub>n</sub> <i>b</i> , half <sub>n</sub> <i>c</i> )	Each bit of result is corresponding bit of <i>a</i> if corresponding bit of <i>c</i> is 0
<i>T</i> select ( <i>Ta</i> , <i>Tb</i> , <i>Ti</i> <i>c</i> ) <i>T</i> select ( <i>Ta</i> , <i>Tb</i> , <i>Tu</i> <i>c</i> ) halfn select (half <sub>n</sub> <i>a</i> , half <sub>n</sub> <i>b</i> , short <sub>n</sub> <i>c</i> ) half select (half <i>a</i> , half <i>b</i> , short <i>c</i> ) halfn select (half <sub>n</sub> <i>a</i> , half <sub>n</sub> <i>b</i> , ushort <sub>n</sub> <i>c</i> ) half select (half <i>a</i> , half <i>b</i> , ushort <i>c</i> )	For each component of a vector type, result[ <i>i</i> ] = if MSB of <i>c</i> [ <i>i</i> ] is set ? <i>b</i> [ <i>i</i> ] : <i>a</i> [ <i>i</i> ] For scalar type, result = <i>c</i> ? <i>b</i> : <i>a</i>

**Geometric Built-in Functions [6.13.5]**

*Ts* is scalar type float, optionally double (if *cl\_khr\_fp64* is enabled), or half (if *cl\_khr\_fp16* is enabled). *T* is *Ts* and the 2-, 3-, or 4-component vector forms of *Ts*.

float{3,4} cross (float{3,4} <i>p0</i> , float{3,4} <i>p1</i> ) double{3,4} cross (double{3,4} <i>p0</i> , double{3,4} <i>p1</i> ) half{3,4} cross (half{3,4} <i>p0</i> , half{3,4} <i>p1</i> )	Cross product
---	---------------

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

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



**Vector Data Load/Store [6.13.7]**

*T* is type char, uchar, short, ushort, int, uint, long, ulong, or float, optionally double (if *cl\_khr\_fp64* is enabled), or half (if *cl\_khr\_fp16* is enabled). *Tn* refers to the vector form of type *T*, where *n* is 2, 3, 4, 8, or 16. *R* defaults to current rounding mode, or is one of the rounding modes listed in 6.2.3.2.

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

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

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

**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.

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

**Atomic Functions [6.13.11]**

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

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

The type atomic\_\* is a 32-bit integer. atomic\_long and atomic\_ulong require extension *cl\_khr\_int64\_base\_atomics* or *cl\_khr\_int64\_extended\_atomics*. The atomic\_double type is available if *cl\_khr\_fp64* is enabled. The default scope is work\_group for local atomics and all\_svm\_devices for global atomics. The extensions *cl\_khr\_int64\_base\_atomics* and *cl\_khr\_int64\_extended\_atomics* implement atomic operations on 64-bit signed and unsigned integers to locations in \_\_global and \_\_local memory.

See the table under Atomic Types and Enum Constants for information about parameter types memory\_order, memory\_scope, and memory\_flag.

<code>void atomic_init (volatile A *obj, C value)</code>	Initializes the atomic object pointed to by <i>obj</i> to the value <i>value</i> .
<code>void atomic_work_item_fence (cl_mem_fence_flags flags, memory_order order, memory_scope scope)</code>	Effects based on value of <i>order</i> . <i>flags</i> must be CLK_{GLOBAL, LOCAL, IMAGE}_MEM_FENCE or a combination of these.
<code>void atomic_store (volatile A *object, C desired)</code> <code>void atomic_store_explicit (volatile A *object, C desired, memory_order order [, memory_scope scope])</code>	Atomically replace the value pointed to by <i>object</i> with the value of <i>desired</i> . Memory is affected according to the value of <i>order</i> .
<code>C atomic_load (volatile A *object)</code> <code>C atomic_load_explicit (volatile A *object, memory_order order [, memory_scope scope])</code>	Atomically returns the value pointed to by <i>object</i> . Memory is affected according to the value of <i>order</i> .
<code>C atomic_exchange (volatile A *object, C desired)</code> <code>C atomic_exchange_explicit (volatile A *object, C desired, memory_order order [, memory_scope scope])</code>	Atomically replace the value pointed to by <i>object</i> with <i>desired</i> . Memory is affected according to the value of <i>order</i> .
<code>bool atomic_compare_exchange_strong (volatile A *object, C *expected, C desired)</code> <code>bool atomic_compare_exchange_strong_explicit (volatile A *object, C *expected, C desired, memory_order success, memory_order failure [, memory_scope scope])</code>	Atomically compares the value pointed to by <i>object</i> for equality with that in <i>expected</i> , and if true, replaces the value pointed to by <i>object</i> with <i>desired</i> , and if false, updates the value in <i>expected</i> with the value pointed to by <i>object</i> . These operations are atomic read-modify-write operations.
<code>bool atomic_compare_exchange_weak (volatile A *object, C *expected, C desired)</code> <code>bool atomic_compare_exchange_weak_explicit (volatile A *object, C *expected, C desired, memory_order success, memory_order failure [, memory_scope scope])</code>	Atomically compares the value pointed to by <i>object</i> for equality with that in <i>expected</i> , and if true, replaces the value pointed to by <i>object</i> with <i>desired</i> , and if false, updates the value in <i>expected</i> with the value pointed to by <i>object</i> . These operations are atomic read-modify-write operations.
<code>C atomic_fetch_and (volatile A *object, M operand)</code> <code>C atomic_fetch_and_explicit (volatile A *object, M operand, memory_order order [, memory_scope scope])</code>	Atomically replaces the value pointed to by <i>object</i> with the result of the computation applied to the value pointed to by <i>object</i> and the given <i>operand</i> .

**Async Copies and Prefetch [6.13.10]**

*T* is type char, charn, uchar, uchar\_n, short, shortn, ushort, ushortn, int, intrn, uint, uintn, long, longn, ulong, ulongn, float, floatn, optionally double or doublen (if *cl\_khr\_fp64* is enabled), or half or halfn (if *cl\_khr\_fp16* is enabled).

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

<code>bool atomic_flag_test_and_set (volatile atomic_flag *object)</code> <code>bool atomic_flag_test_and_set_explicit (volatile atomic_flag *object, memory_order order [, memory_scope scope])</code>	Atomically sets the value pointed to by <i>object</i> to true. Memory is affected according to the value of <i>order</i> . Returns atomically, the value of the object immediately before the effects.
<code>void atomic_flag_clear (volatile atomic_flag *object)</code> <code>void atomic_flag_clear_explicit (volatile atomic_flag *object, memory_order order [, memory_scope scope])</code>	Atomically sets the value pointed to by <i>object</i> to false. The order argument shall not be <i>memory_order_acquire</i> nor <i>memory_order_acq_rel</i> . Memory is affected according to the value of <i>order</i> .

**Values for key for atomic\_fetch and modify functions**

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

**Atomic Types and Enum Constants**

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

**Atomic integer and floating-point types**

† indicates types supported by a limited subset of atomic operations  
‡ indicates size depends on whether implemented on 64-bit or 32-bit architecture.  
§ indicates types supported only if extensions *cl\_khr\_int64\_base\_atomics* and *cl\_khr\_int64\_extended\_atomics* are enabled.

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

**Atomic macros**

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

**Address Space Qualifier Functions [6.13.9]**

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

[const] global <i>T</i> * <b>to_global</b> ([const] <i>T</i> * <i>ptr</i> )	global address space
[const] local <i>T</i> * <b>to_local</b> ([const] <i>T</i> * <i>ptr</i> )	local address space
[const] private <i>T</i> * <b>to_private</b> ([const] <i>T</i> * <i>ptr</i> )	private address space
[const] cl_mem_fence_flags <b>get_fence</b> ([const] <i>T</i> * <i>ptr</i> )	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);
uchar4 uc = (uchar4)(0xFA, 0xFB, 0xFC, 0xFD);
printf("f4 = %2.2v4hf\n", f);
printf("uc = %#v4hhx\n", uc);
```

```
The above two printf calls print the following:
f4 = 1.00,2.00,3.00,4.00
uc = 0xfa,0xfb,0xfc,0xfd
```

**Miscellaneous Vector Functions [6.13.12]**

*Tm* and *Tn* are type *charn*, *ucharn*, *shortn*, *ushortn*, *intn*, *uintn*, *longn*, *ulongn*, *floatn*, optionally *doublen* (if *cl\_khr\_fp64* is enabled) or *halfn* (if *cl\_khr\_fp16* is enabled), where *n* is 2,4,8, or 16 except in *vec\_step* it may also be 3. *TUn* is *ucharn*, *ushortn*, *intn*, or *ulongn*.

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

**Workgroup Functions [6.13.15]**

*T* is type int, uint, long, ulong, or float, optionally double (if *cl\_khr\_fp64* is enabled) or half (if *cl\_khr\_fp16* is enabled).

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

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

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

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

Broadcast the value of *a* to all work-items in the work-group. *local\_id* must be the same value for all workitems in the work-group.

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

Do an exclusive or inclusive scan operation specified by *<op>* of all values specified by work-items in the work-group. 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. Double or vector double types require the *cl\_khr\_fp64* extension. The macro CLK\_NULL\_RESERVE\_ID refers to an invalid reservation ID.

int <b>read_pipe</b> ( <i>__read_only pipe T p</i> , <i>T *ptr</i> )	Read packet from <i>p</i> into <i>ptr</i> .
int <b>read_pipe</b> ( <i>__read_only pipe T p</i> , <i>reserve_id_t reserve_id</i> , <i>uint index</i> , <i>T *ptr</i> )	Read packet from reserved area of the pipe <i>reserve_id</i> and <i>index</i> into <i>ptr</i> .
int <b>write_pipe</b> ( <i>__write_only pipe T p</i> , <i>const T *ptr</i> )	Write packet specified by <i>ptr</i> to <i>p</i> .
int <b>write_pipe</b> ( <i>__write_only pipe T p</i> , <i>reserve_id_t reserve_id</i> , <i>uint index</i> , <i>const T *ptr</i> )	Write packet specified by <i>ptr</i> to reserved area <i>reserve_id</i> and <i>index</i> .
bool <b>is_valid_reserve_id</b> ( <i>reserve_id_t reserve_id</i> )	Return true if <i>reserve_id</i> is a valid reservation ID and false otherwise.

<i>reserve_id_t</i> <b>reserve_read_pipe</b> ( <i>__read_only pipe T p</i> , <i>uint num_packets</i> )	Reserve <i>num_packets</i> entries for reading from or writing to <i>p</i> .
<i>reserve_id_t</i> <b>reserve_write_pipe</b> ( <i>__write_only pipe T p</i> , <i>uint num_packets</i> )	
void <b>commit_read_pipe</b> ( <i>__read_only pipe T p</i> , <i>reserve_id_t reserve_id</i> )	Indicates that all reads and writes to <i>num_packets</i> associated with reservation <i>reserve_id</i> are completed.
void <b>commit_write_pipe</b> ( <i>__write_only pipe T p</i> , <i>reserve_id_t reserve_id</i> )	
uint <b>get_pipe_max_packets</b> ( <i>pipe T p</i> )	Returns maximum number of packets specified when <i>p</i> was created.
uint <b>get_pipe_num_packets</b> ( <i>pipe T p</i> )	Returns the number of available entries in <i>p</i> .

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

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

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. The macro CLK\_NULL\_EVENT refers to an invalid device event. The macro CLK\_NULL\_QUEUE refers to an invalid device queue.

int <b>enqueue_kernel</b> ( <i>queue_t queue</i> , <i>kernel_enqueue_flags_t flags</i> , <i>const ndrange_t ndrange</i> , void ( <i>^block</i> )(void))	Allows a work-item to enqueue a block for execution to <i>queue</i> . Work-items can enqueue multiple blocks to a device queue(s). <i>flags</i> may be one of CLK_ENQUEUE_FLAGS_{NO_WAIT, WAIT_KERNEL, WAIT_WORK_GROUP}
int <b>enqueue_kernel</b> ( <i>queue_t queue</i> , <i>kernel_enqueue_flags_t flags</i> , <i>const ndrange_t ndrange</i> , <i>uint num_events_in_wait_list</i> , <i>const clk_event_t *event_wait_list</i> , <i>clk_event_t *event_ret</i> , void ( <i>^block</i> )(void))	
int <b>enqueue_kernel</b> ( <i>queue_t queue</i> , <i>kernel_enqueue_flags_t flags</i> , <i>const ndrange_t ndrange</i> , void ( <i>^block</i> )(local void *, ...), <i>uint size0</i> , ...)	
int <b>enqueue_kernel</b> ( <i>queue_t queue</i> , <i>kernel_enqueue_flags_t flags</i> , <i>const ndrange_t ndrange</i> , <i>uint num_events_in_wait_list</i> , <i>const clk_event_t *event_wait_list</i> , <i>clk_event_t *event_ret</i> , void ( <i>^block</i> )(local void *, ...), <i>uint size0</i> , ...)	

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





## OpenCL Extensions Reference

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

## Using OpenCL Extensions [9]

In this section, extensions shown in *italics* provide core features.#pragma OPENCL EXTENSION *extension\_name* : {enable | disable}To test if an extension is supported, use  
clGetPlatformInfo() or clGetDeviceInfo()To get the address of the extension function:  
clGetExtensionFunctionAddressForPlatform()

<i>cl_apple_gl_sharing</i> (see <i>cl_khr_gl_sharing</i> )
<i>cl_khr_3d_image_writes</i>
<i>cl_khr_byte_addressable_store</i>
<i>cl_khr_context_abort</i>
<i>cl_khr_d3d10_sharing</i>
<i>cl_khr_d3d11_sharing</i>

<i>cl_khr_depth_images</i>
<i>cl_khr_device_enqueue_local_arg_types</i>
<i>cl_khr_dx9_media_sharing</i>
<i>cl_khr_egl_event</i>
<i>cl_khr_egl_image</i>
<i>cl_khr_fp16</i>
<i>cl_khr_fp64</i>
<i>cl_khr_gl_depth_images</i>
<i>cl_khr_gl_event</i>
<i>cl_khr_gl_msaa_sharing</i>
<i>cl_khr_gl_sharing</i>
<i>cl_khr_global_int32_base_atomics - atomic_*</i> ()
<i>cl_khr_global_int32_extended_atomics - atomic_*</i> ()
<i>cl_khr_icd</i>

<i>cl_khr_image2d_from_buffer</i>
<i>cl_khr_initialize_memory</i>
<i>cl_khr_int64_base_atomics - atomic_*</i> ()
<i>cl_khr_int64_extended_atomics - atomic_*</i> ()
<i>cl_khr_local_int32_base_atomics - atomic_*</i> ()
<i>cl_khr_local_int32_extended_atomics - atomic_*</i> ()
<i>cl_khr_mipmap_image</i>
<i>cl_khr_mipmap_image_writes</i>
<i>cl_khr_priority_hints</i>
<i>cl_khr_srgb_image_writes</i>
<i>cl_khr_spir</i>
<i>cl_khr_subgroup_named_barrier</i>
<i>cl_khr_terminate_context</i>
<i>cl_khr_throttle_hints</i>

## OpenGL, OpenGL ES Sharing [9.3 - 9.5]

These functions require the *cl\_khr\_gl\_sharing* or  
*cl\_apple\_gl\_sharing* extension.

## CL Context &gt; GL Context, Sharegroup

```
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 &gt; GL Buffer Objects

```
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 &gt; GL Textures

```
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 [9.7]

Header &lt;cl\_dx9\_media\_sharing.h&gt;

Enable the extension *cl\_khr\_dx9\_media\_sharing*.

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

media\_adapter\_type:

```
CL_ADAPTER_{D3D9, D3D9EX, DXVA}_KHR
media_adapter_set: CL_{ALL, PREFERRED}_DEVICES_
FOR_DX9_MEDIA_ADAPTER_KHR
```

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

flags: See clCreateFromGLBuffer
adapter_type: CL_ADAPTER_{D3D9, D3D9EX, DXVA}_KHR
```

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

## EGL Interoperability [9.16, 9.17]

## Create CL Event Objects from EGL

This function requires the extension *cl\_khr\_egl\_event*.

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

## CL Image Objects &gt; GL Renderbuffers

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

flags: See clCreateFromGLBuffer
```

## Query Information

```
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

```
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 &gt; GL Sync Objects

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

Requires the cl_khr_gl_event extension.
```

## Direct3D 11 Sharing [9.8.7]

Header <cl\_d3d11.h> These functions require the  
*cl\_khr\_d3d11\_sharing* extension. For values of flags, see  
clCreateFromGLBuffer.

```
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
```

## Direct3D 10 Sharing [9.6.7]

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

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

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

```
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)
```

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

## Create CL Image Objects from EGL

These functions require the extension *cl\_khr\_egl\_image*.

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

## Example of Enqueuing Kernels

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

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

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

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

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

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

### A Complete Example [6.13.17.3]

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

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

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

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

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

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

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

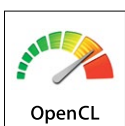




## OpenCL Reference Card Index

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

<b>A</b>		clEnqueueReadBuffer ()	2	<b>D</b>		OpenGL and OpenGL ES Sharing	Ext	21
Access Qualifiers	c	14	clEnqueueReadBufferRect ()	2	Debugging options			4
Address Space Qualifier functions	c	19	clEnqueueReadImage ()	2	depth images	C++		7
Address Space Qualifiers	c	13	clEnqueueSVM* ()	3	Device Architecture Diagram			23
Address Spaces	C++	6	clEnqueueUnmapMemobject ()	3	Device Enqueue	C++		8-9
Array library	C++	11	clEnqueueWriteBuffer ()	2	Direct3D 10 Sharing	Ext		21
Async Copies	c	18	clEnqueueWriteBufferRect ()	2	Direct3D 11 Sharing	Ext		21
Atomics	c	18	clEnqueueWriteImage ()	2	DX9 Media Surface Sharing	Ext		21
Atomics library	C++	6-7	clFinish ()	3	<b>E</b>			
Attribute Qualifiers	c	13	clFlush ()	3	EGL Interoperability	Ext		21
Attributes	C++	5	clGetCommandQueueInfo ()	1	Enqueue	C++		8-9
<b>B</b>			clGetContextInfo ()	1	Enqueue	c		19
Barrier functions	C++	9	clGetDeviceAndHostTimer ()	1	Event functions	c		20
bitwise functions	C++	10	clGetDeviceIDs ()	1	Event objects			4
Blocks	c	14	clGetDeviceInfo ()	1	Extensions	Ext		21
Broadcast functions	C++	9	clGetEventInfo ()	4	<b>F</b>			
Buffer objects		2	clGetEventProfilingInfo ()	4	fast	c		13
<b>C</b>			clGetExtensionFunctionAddress* ()	1	Fast 24-bit operations	C++		10
C Language Reference	c	13	clGetHostTimer ()	1	Fences	C++		7
C++ 14	C++	5	clGetImageInfo ()	2	Flush and Finish			3
C++ Language Reference	C++	5	clGetKernelArgInfo ()	4	Function qualifier	C++		5
channel_ref	C++	12	clGetKernelInfo ()	4	Function qualifier	c		13
Class Diagram		23	clGetKernelSubGroupInfo ()	4	<b>G</b>			
cl[Release, Retain]CommandQueue ()		1	clGetKernelWorkGroupInfo ()	4	Geometric functions	C++		11
cl[Release, Retain]Context ()		1	clGetMemobjectInfo ()	3	Geometric functions	c		17
cl[Release, Retain]Device ()		1	clGetPipeInfo ()	3	global<T>	C++		6
cl[Release, Retain]Event ()		4	clGetPlatformIDs ()	1	<b>H-I</b>			
cl[Release, Retain]Kernel ()		4	clGetPlatformInfo ()	1	half	C++		5
cl[Release, Retain]Memobject ()		3	clGetProgramBuildInfo ()	3	Image objects			2
cl[Release, Retain]Program ()		3	clGetProgramInfo ()	3	Image query functions	c		16
cl[Release, Retain]Sampler ()		3	clGetSamplerInfo ()	3	Image Read and Write functions	c		15-16
clBuildProgram ()		3	clGetSupportedImageFormats ()	2	Images	C++		7-8
clCloneKernel ()		4	clIcdGetPlatformIDsKHR ()	1	Images and Samplers library	C++		7-8
clCompileProgram ()		3	clLinkProgram ()	3	Integer functions	c		17
clCreateBuffer ()		2	clSetDefaultDeviceCommandQueue ()	1	Integer functions	C++		10
clCreateCommandQueue* ()		1	clSetEventCallback ()	4	Iterator library	C++		12
clCreateContext* ()		1	clSetKernelArg ()	4	<b>K-L</b>			
clCreateImage ()		2	clSetKernelExecInfo ()	4	Kernel objects			4
clCreateKernel ()		4	clSetMemobjectDestructorCallback ()	3	Kernel query functions	c		19
clCreateKernelsInProgram ()		4	clSetProgramReleaseCallback ()	3	Limits	C++		11-12
clCreatePipe ()		3	clSetProgramSpecializationConstant ()	3	Link			3-4
clCreateProgramWith* ()		3	clSetUserEventStatus ()	4	Linker			4
clCreateSamplerWithProperties ()		3	clSVMAlloc ()	3	local<T>	C++		6
clCreateSubBuffer ()		2	clSVMFree ()	3	<b>M</b>			
clCreateSubDevices ()		1	clTerminateContextKHR ()	1	Macros	c		14
clCreateUserEvent ()		4	clUnloadPlatformCompiler ()	3	Markers, barriers, & waiting for events			4
clEnqueueBarrierWithWaitList ()		4	clWaitForEvents ()	4	Math constants	C++		12
clEnqueueCopyBuffer ()		2	Command queues		Math constants	c		14
clEnqueueCopyBufferRect ()		2	Common functions	C++	10	Math functions	C++	10
clEnqueueCopyBufferToImage ()		2	Common functions	c	17	Math functions	c	14
clEnqueueCopyImage ()		2	Comparison functions	C++	10	Memory objects		3
clEnqueueCopyImageToBuffer ()		2	Compile	3-4	mipmap	C++		7-8
clEnqueueFillBuffer ()		2	Compiler options	4	mipmap	c		15-16
clEnqueueFillImage ()		2	constant<T>	C++	6	<b>N-O</b>		
clEnqueueMapBuffer ()		2	Constants	C++	12	Named barriers	C++	9
clEnqueueMapImage ()		2	Constants	c	14	Named Barriers for Subgroups	Ext	21
clEnqueueMarkerWithWaitList ()		4	Contexts		1	ndrange	C++	5
clEnqueueMigrateMemobjects ()		3	Conversions	C++	5	ndrange	c	20
clEnqueueNativeKernel ()		4	Conversions and Type Casting	c	13	<b>P-Q</b>		
clEnqueueNDRangeKernel ()		4				OpenGL and OpenGL ES Sharing	Ext	21
						Operators	c	13
						<b>P-Q</b>		
						Pipe functions	c	19
						Pipes		3
						Pipes	C++	8
						Platform Layer		1
						Pointer class	C++	6
						pragma	c	13
						Prefetch	c	18
						Preprocessor Directives & Macros	C++	5
						Preprocessor Directives & Macros	c	13
						printf function	C++	11
						printf function	c	19
						priv<T>	C++	6
						Profiling operations		4
						Program linking options		4
						Program objects		3-4
						Qualifiers	c	13
						<b>R</b>		
						read_image*()	c	15-16
						Reinterpreting types	C++	5
						Relational functions	c	17
						Relational functions	C++	11
						Retain and release program objects		3
						Rounding modes	C++	5
						Rounding modes	c	13
						Runtime		1
						<b>S</b>		
						Sampler	C++	7-8
						Sampler objects		3
						Shared Virtual Memory		3
						SPIR 1.2 Binaries	Ext	21
						SPIR-V specialization constants		3
						SVM		4
						SVM operations		3
						swizzles	C++	12
						Synchronization & Memory Fence functions	c	18
						Synchronization functions	C++	9
						<b>T</b>		
						Traits	C++	12
						Tuple library	C++	12
						Types	C++	5
						Types	c	13
						<b>V</b>		
						Vector Component Addressing	C++	6
						Vector Component Addressing	c	13
						Vector Data Load/Store	c	18
						Vector Data Load/Store	C++	11
						Vector functions	c	19
						Vector Utilities	C++	12
						Vector Wrapper library	C++	12
						<b>W</b>		
						Work-Item functions	c	14
						Work-Item functions	C++	9
						Workgroup functions	C++	9
						Workgroup functions	c	19
						write_image*()	c	15-16



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](http://www.khronos.org) to learn more about the Khronos Group.

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