

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

[n.n.n] refers to the section in the API Specification on available at www.khronos.org/opens.

The OpenCL Runtime

Command Queues [5.1]

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

properties: CL_QUEUE_PROFILING_ENABLE,
CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE

```
cl_int dRetainCommandQueue (cl_command_queue
    command_queue)
```

```
cl_int dReleaseCommandQueue (cl_command_queue
    command_queue)
```

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

param_name: CL_QUEUE_CONTEXT,
CL_QUEUE_DEVICE,
CL_QUEUE_REFERENCE_COUNT,
CL_QUEUE_PROPERTIES

```
cl_int dSetCommandQueueProperty (cl_command_queue
    command_queue, cl_command_queue_properties
    properties, cl_bool enable,
    cl_command_queue_properties *old_properties)
```

properties:
CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE,
CL_QUEUE_PROFILING_ENABLE

The OpenCL Platform Layer

The OpenCL platform layer which implements platform-specific features that allow applications to query OpenCL devices, device configuration information, and to create OpenCL contexts using one or more devices.

Contexts [4.3]

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

d_context_properties: CL_CONTEXT_PLATFORM,
CL_GL_CONTEXT_KHR, CL_CGL_SHAREGROUP_KHR,
CL_EGL_DISPLAY_KHR, CL_GLX_DISPLAY_KHR,
CL_WGL_HDC_KHR

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

d_context_properties: (same as for d_create_context)

```
cl_int dRetainContext (cl_context context)
```

```
cl_int dReleaseContext (cl_context context)
```

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

param_name: CL_CONTEXT_REFERENCE_COUNT,
CL_CONTEXT_DEVICES, CL_CONTEXT_PROPERTIES

Querying Platform Info and Devices [4.1, 4.2]

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

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

param_name: CL_PLATFORM_PROFILE,
CL_PLATFORM_VERSION, CL_PLATFORM_NAME,
CL_PLATFORM_VENDOR, CL_PLATFORM_EXTENSIONS

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

device_type: CL_DEVICE_TYPE_CPU, CL_DEVICE_TYPE_GPU,
CL_DEVICE_TYPE_ACCELERATOR, CL_DEVICE_TYPE_DEFAULT,
CL_DEVICE_TYPE_ALL

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

param_name: CL_DEVICE_PLATFORM, CL_DEVICE_TYPE,
CL_DEVICE_VENDOR_ID,
CL_DEVICE_MAX_COMPUTE_UNITS,
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS,
CL_DEVICE_MAX_WORK_ITEM_SIZES,
CL_DEVICE_MAX_WORK_GROUP_SIZE,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT,
CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE,
CL_DEVICE_MAX_CLOCK_FREQUENCY,
CL_DEVICE_ADDRESS_BITS,
CL_DEVICE_MAX_MEM_ALLOC_SIZE,
CL_DEVICE_IMAGE_SUPPORT,
CL_DEVICE_MAX_READ_IMAGE_ARGS,
CL_DEVICE_MAX_WRITE_IMAGE_ARGS,
CL_DEVICE_IMAGE2D_MAX_WIDTH_HEIGHT,
CL_DEVICE_IMAGE3D_MAX_WIDTH_HEIGHT_DEPTH,
CL_DEVICE_MAX_SAMPLERS,
CL_DEVICE_MAX_PARAMETER_SIZE,
CL_DEVICE_MEM_BASE_ADDR_ALIGN,
CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE,
CL_DEVICE_SINGLE_FP_CONFIG,
CL_DEVICE_GLOBAL_MEM_CACHE_TYPE,
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE,
CL_DEVICE_GLOBAL_MEM_CACHE_SIZE,
CL_DEVICE_GLOBAL_MEM_SIZE,
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE,
CL_DEVICE_MAX_CONSTANT_ARGS,
CL_DEVICE_LOCAL_MEM_TYPE,
CL_DEVICE_LOCAL_MEM_SIZE,
CL_DEVICE_ERROR_CORRECTION_SUPPORT,
CL_DEVICE_PROFILING_TIMER_RESOLUTION,
CL_DEVICE_ENDIAN_LITTLE,
CL_DEVICE_AVAILABLE, CL_DEVICE_COMPILER_AVAILABLE,
CL_DEVICE_EXECUTION_CAPABILITIES,
CL_DEVICE_QUEUE_PROPERTIES,
CL_DEVICE_NAME, CL_DEVICE_VENDOR,
CL_DRIVER_VERSION, CL_DEVICE_PROFILE,
CL_DEVICE_VERSION, CL_DEVICE_EXTENSIONS

Memory Objects

Memory objects include *buffer* objects, and *image* objects. Refer to the Graphic page for information about image objects.

A buffer object stores a one-dimensional collection of elements. Elements of a buffer object can be a scalar data type (such as int, float), vector data type, or a user-defined structure, and are stored in sequential fashion and can be accessed using a pointer by a kernel executing on a device. The data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

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

flags: CL_MEM_READ_WRITE,
CL_MEM_WRITE_ONLY,
CL_MEM_READ_ONLY,
CL_MEM_USE_HOST_PTR,
CL_MEM_ALLOC_HOST_PTR,
CL_MEM_COPY_HOST_PTR

Read, Write, Copy Buffer Objects [5.2.2 - 5.2.3]

```
cl_int dEnqueueReadBuffer (
    cl_command_queue command_queue, cl_mem buffer,
    cl_bool blocking_read, size_t ofset, size_t cb,
    void *ptr, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

```
cl_int dEnqueueWriteBuffer (
    cl_command_queue command_queue, cl_mem buffer,
    cl_bool blocking_write, size_t ofset, size_t cb,
    const void *ptr, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

```
cl_int dEnqueueCopyBuffer (
    cl_command_queue command_queue,
    cl_mem src_buffer, cl_mem dst_buffer, size_t src_ofset,
    size_t dst_ofset, size_t cb,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

```
cl_int dRetainMemObject (cl_mem memobj)
```

```
cl_int dReleaseMemObject (cl_mem memobj)
```

Map and Unmap Memory Objects [5.2.8]

```
void *dEnqueueMapBuffer (
    cl_command_queue command_queue, cl_mem buffer,
    cl_bool blocking_map, cl_mem_flags map_flags,
    size_t ofset, size_t cb, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event,
    cl_int *errcode_ret)
```

```
cl_int dEnqueueUnmapMemObject (
    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)
```

Query Buffer Object [5.2.9]

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

param_name: CL_MEM_TYPE,
CL_MEM_FLAGS, CL_MEM_SIZE,
CL_MEM_HOST_PTR, CL_MEM_MAP_COUNT,
CL_MEM_REFERENCE_COUNT, CL_MEM_CONTEXT

Program Objects

Create Program Objects [5.4.1]

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

```
cl_program dCreateProgramWithBinary (
    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_int dRetainProgram (cl_program program)
```

```
cl_int dReleaseProgram (cl_program program)
```

Build Program Executable [5.4.2]

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

Build Options [5.4.3]

Preprocessor options
(-D options processed in order listed in dBuildProgram)

-D name,
-D name=definition,
-I dir

Math Intrinsics options

-d single-precision-constant,
-d denorms-are-zero,

Optimization options

-cl-opt-disable, -cl-strict-aliasing,
-cl-mad-enable, -cl-no-signed-zeros,
-cl-finite-math-only, -cl-fast-relaxed-math,
-cl-unsafe-math-optimization

Warning request/suppress options

-W -Werror

Unload the OpenCL Compiler [5.4.4]

```
cl_int dUnloadCompiler (void)
```

Query Program Objects [5.4.5]

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

param_name: CL_PROGRAM_REFERENCE_COUNT,
CL_PROGRAM_CONTEXT,
CL_PROGRAM_NUM_DEVICES,
CL_PROGRAM_DEVICES,
CL_PROGRAM_SOURCE,
CL_PROGRAM_BINARY_SIZES,
CL_PROGRAM_BINARIES,
CL_PROGRAM_SOURCE

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

param_name: CL_PROGRAM_BUILD_STATUS,
CL_PROGRAM_BUILD_OPTIONS,
CL_PROGRAM_BUILD_LOG

Kernel and Event Objects

Create Kernel Queries [5.5.1]

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

```
cl_int clCreateKernelsInProgram (d_program program,
    cl_uint num_kernels, d_kernel *kernels,
    cl_uint *num_kernels_ret)
```

```
cl_int clRetainKernel (d_kernel kernel)
```

```
cl_int clReleaseKernel (d_kernel kernel)
```

Kernel Arguments & Object Queries [5.5.2, 5.5.3]

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

```
cl_int clGetKernelInfo (d_kernel kernel,
    d_kernel_info_param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
```

```
param_name: CL_KERNEL_FUNCTION_NAME,
CL_KERNEL_NUM_ARGS,
CL_KERNEL_REFERENCE_COUNT,
CL_KERNEL_CONTEXT, CL_KERNEL_PROGRAM
```

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

```
param_name: CL_KERNEL_WORK_GROUP_SIZE,
CL_KERNEL_COMPILE_WORK_GROUP_SIZE,
CL_KERNEL_LOCAL_MEM_SIZE
```

Execute Kernels [5.6]

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

```
cl_int clEnqueueTask (
    cl_command_queue command_queue, d_kernel kernel,
    cl_uint num_events_in_wait_list,
    const d_event *event_wait_list, cl_event *event)
```

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

Event Objects [5.7]

```
cl_int clWaitForEvents (
    cl_uint num_events, const d_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)
```

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

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

Out-of-order Execution of Kernels & Memory Object Commands [5.8]

```
cl_int clEnqueueMarker (
    cl_command_queue command_queue, cl_event *event)
```

```
cl_int clEnqueueWaitForEvents (
    cl_command_queue command_queue,
    cl_uint num_events, const d_event *event_list)
```

```
cl_int clEnqueueBarrier (
    cl_command_queue command_queue)
```

Profile Operations on Memory Objects & Kernels [5.9]

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

```
param_name: CL_PROFILING_COMMAND_QUEUED,
CL_PROFILING_COMMAND_SUBMIT,
CL_PROFILING_COMMAND_START,
CL_PROFILING_COMMAND_END
```

Flush and Finish [5.10]

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

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

```
param_name: CL_EVENT_COMMAND_QUEUE,
CL_EVENT_COMMAND_TYPE,
CL_EVENT_COMMAND_EXECUTION_STATUS,
CL_EVENT_REFERENCE_COUNT
```

Supported Data Types

Built-in Scalar Data Types [6.1.1]

OpenCL Type	API Type	Description
bool	--	true (1) or false (0)
char	d_char	8-bit signed
unsigned char, uchar	d_uchar	8-bit unsigned
short	d_short	16-bit signed
unsigned short, ushort	d_ushort	16-bit unsigned
int	cl_int	32-bit signed
unsigned int, uint	cl_uint	32-bit unsigned
long	d_long	64-bit signed
unsigned long, ulong	d_ulong	64-bit unsigned
float	d_float	32-bit float
half	d_half	16-bit float (for storage only)
size_t	--	32- or 64-bit unsigned integer
ptrdiff_t	--	32- or 64-bit signed integer
intptr_t	--	signed integer
uintptr_t	--	unsigned integer
void	--	void

Built-in Vector Data Types [6.1.2]

OpenCL Type	API Type	Description
char _n	d_char _n	8-bit signed
uchar _n	d_uchar _n	8-bit unsigned
short _n	d_short _n	16-bit signed
ushort _n	d_ushort _n	16-bit unsigned
int _n	cl_int _n	32-bit signed
uint _n	cl_uint _n	32-bit unsigned
long _n	d_long _n	64-bit signed
ulong _n	d_ulong _n	64-bit unsigned
float _n	d_float _n	32-bit float

Other Built-in Data Types [6.1.3]

OpenCL Type	Description
image2d_t	2D image handle
image3d_t	3D image handle
sampler_t	sampler handle
event_t	event handle

Reserved Data Types [6.1.4]

OpenCL Type	Description
bool _n	boolean vector
double, double _n	64-bit float, vector OPT
half _n	16-bit float, vector OPT
quad, quad _n	128-bit float, vector
complex half, complex half _n imaginary half, imaginary half _n	16-bit complex, vector
complex float, complex float _n imaginary float, imaginary float _n	32-bit complex, vector
complex double, complex double _n imaginary double, imaginary double _n	64-bit complex, vector
complex quad, complex quad _n imaginary quad, imaginary quad _n	128-bit complex, vector
float _n x _m	<i>n</i> * <i>m</i> matrix of 32-bit floats
double _n x _m	<i>n</i> * <i>m</i> matrix of 64-bit floats
long double, long double _n	64-128-bit float, vector
long long, long long _n	128-bit signed
unsigned long long, unsigned long long, ulong long _n	128-bit unsigned

Vector Component Addressing [6.1.7]

The components of a vector may be addressed as shown below or as shown in the table of equivalencies.

Vector Components

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

Vector Addressing Equivalencies

	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
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.s89abcd	v.s13579bdf	v.s02468ace

When addressing vector components by numeric indices, they must be preceded by the letter s or S, e.g.: s1.

Swizzling, duplication, and nesting are allowed, e.g.: vxy, vxx, v.lo.x

Conversions and Type Casting Examples [6.2]

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

Rounding Modes [6.2.3.2]

R can be:

- _rte Round to nearest even
- _rtz Round toward zero
- _rtp Round toward positive infinity
- _rtn Round toward negative infinity

Operators [6.3]

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

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

Address Space Qualifiers [6.5]

```
_global, global _local, local
_constant, constant _private, private
```

Function Qualifiers [6.7]

```
_kernel, kernel
_at_ribute__((vec_type_hint(int)))
_at_ribute__((work_group_size_hint(X, Y, Z)))
_at_ribute__((reqd_work_group_size(X, Y, Z)))
```

Preprocessor Directives & Macros [6.9]

```
#pragma OPENCL_FP_CONTRACT on-of -switch
on-of -switch: ON, OFF, DEFAULT
```

Predefined Macro Names

```
_FILE_ Current source file
_LINE_ Line number
_OPENCL_VERSION_ Integer version number
_ENDIAN_LITTLE_ 1 if device is little endian
_ROUNDING_MODE_ Current rounding mode (default "rte")
_kernel_exec(X, typen) Same as __kernel__ at_ribute__((work_group_size_hint(X, 1, 1))) \
_at_ribute__((vec_type_hint(typen)))
_IMAGE_SUPPORT_ 1 if images are supported,
_FAST_RELAXED_MATH_ 1 if -d-fast-relaxed-math optimization is specified
```

Work-Item Built-in Functions [6.11.1]

D is dimension index.

<code>uint_get_work_dim()</code>	Num. of dimensions in use
<code>size_t_get_global_size(uint D)</code>	Num. of global work-items
<code>size_t_get_global_id(uint D)</code>	Global work-item ID value
<code>size_t_get_local_size(uint D)</code>	Num. of local work-items
<code>size_t_get_local_id(uint D)</code>	Local work-item ID
<code>size_t_get_num_groups(uint D)</code>	Num. of work-groups
<code>size_t_get_group_id(uint D)</code>	Returns the work-group ID

Integer Built-in Functions [6.11.3]

T is type char, charn, uchar, ucharn, short, shortn, ushort, ushortn, int, intn, uint, uintn, long, longn, ulong, or ulongn. *U* refers to the unsigned version of *T*.

<code>Uabs(T x)</code>	x
<code>Uabs_dif(T x, T y)</code>	x - y without modulo overflow
<code>Tadd_sat(T x, T y)</code>	x + y and saturates the result
<code>Thadd(T x, T y)</code>	(x + y) >> 1 without modulo overflow
<code>Thradd(T x, T y)</code>	(x + y + 1) >> 1
<code>Tdz(T x)</code>	Number of leading 0-bits in x
<code>Tmad_hi(T a, T b, T c)</code>	mul_hi(a, b) + c
<code>Tmad24(T a, T b, T c)</code>	Multiply 24-bit integer values a and b and add the 32-bit integer result to 32-bit integer c
<code>Tmad_sat(T a, T b, T c)</code>	a * b + c and saturates the result
<code>Tmax(T x, T y)</code>	y if x < y, otherwise it returns x

Floating-Point Math Constants [6.11.2]

MAXFLOAT	Value of maximum non-infinite single-precision float ng-point number.
HUGE_VALF	Positive float constant expression. HUGE_VALF evaluates to +infinity. Used as an error value.
INFINITY	Constant expression of type float representing positive or unsigned infinity.
NAN	Constant expression of type float representing a quiet NaN.

<code>Tmin(T x, T y)</code>	y if y < x, otherwise it returns x
<code>Tmul_hi(T x, T y)</code>	high half of the product of x and y
<code>Tmul24(T a, T b)</code>	Multiply 24-bit integer values a and b
<code>Trotate(T v, T l)</code>	result[<i>indx</i>] = v[<i>indx</i>] << l[<i>indx</i>]
<code>Tsub_sat(T x, T y)</code>	x - y and saturates the result

For upsample, scalar types are permitted for the vector types below

<code>shortn_upsample(charn hi, ucharn lo)</code>	result[<i>i</i>] = ((short)hi[<i>i</i>] << 8) lo[<i>i</i>]
<code>ushortn_upsample(uchar n hi, uchar n lo)</code>	result[<i>i</i>] = ((ushort)hi[<i>i</i>] << 8) lo[<i>i</i>]
<code>intn_upsample(shortn hi, ushortn lo)</code>	result[<i>i</i>] = ((int)hi[<i>i</i>] << 16) lo[<i>i</i>]
<code>uintn_upsample(ushortn hi, ushortn lo)</code>	result[<i>i</i>] = ((uint)hi[<i>i</i>] << 16) lo[<i>i</i>]
<code>longn_upsample(intn hi, uintn lo)</code>	result[<i>i</i>] = ((long)hi[<i>i</i>] << 32) lo[<i>i</i>]
<code>ulongn_upsample(uintn hi, uintn lo)</code>	result[<i>i</i>] = ((ulong)hi[<i>i</i>] << 32) lo[<i>i</i>]

Common Built-in Functions [6.11.4]

T is type float or floatn (or optionally double, doublen, or halfn). Optional extensions enable double, doublen, and halfn types.

<code>Tclamp(T x, T min, T max)</code> <code>floatn_clamp(floatn x, floatn min, floatn max)</code> <code>doublen_clamp(doublen x, doublen min, doublen max)</code> <code>halfn_clamp(halfn x, halfn min, halfn max)</code>	Clamp x to range given by min, max
<code>Tdegrees(T radians)</code>	radians to degrees
<code>Tmax(T x, T y)</code> <code>floatn_max(floatn x, floatn y)</code> <code>doublen_max(doublen x, doublen y)</code> <code>halfn_max(halfn x, halfn y)</code>	Max of x and y
<code>Tmin(T x, T y)</code> <code>floatn_min(floatn x, floatn y)</code> <code>doublen_min(doublen x, doublen y)</code> <code>halfn_min(halfn x, halfn y)</code>	Min of x and y
<code>Tmix(T x, T y)</code> <code>floatn_mix(floatn x, floatn y)</code> <code>doublen_mix(doublen x, doublen y)</code> <code>halfn_mix(halfn x, halfn y)</code>	Linear blend of x and y
<code>T radians(T degrees)</code>	degrees to radians
<code>Tstep(T edge, T x)</code> <code>floatn_step(floatn edge, floatn x)</code> <code>doublen_step(doublen edge, doublen x)</code> <code>halfn_step(halfn edge, halfn x)</code>	0.0 if x < edge, else 1.0
<code>Tsmoothstep(T edge0, T edge1, T x)</code> <code>floatn_smoothstep(floatn edge0, floatn edge1, floatn x)</code> <code>doublen_smoothstep(doublen edge0, doublen edge1, doublen x)</code> <code>halfn_smoothstep(halfn edge0, halfn edge1, halfn x)</code>	Step and interpolate
<code>Tsign(T x)</code>	Sign of x

Math Built-in Functions [6.11.5]

T is type float or floatn (or optionally double, doublen, or halfn). intn, uintn, and ulongn must be scalar when *T* is scalar. The symbol HN indicates that Half and Nat ve variants are available by prepending "half_" or "nat ve_" to the function name, as in half_cos() and nat ve_cos(). Optional extensions enable double, doublen, and halfn types.

<code>Tacos(T)</code>	Arc cosine
<code>Tacosh(T)</code>	Inverse hyperbolic cosine
<code>Tacospi(T x)</code>	acos(x) / pi
<code>Tasin(T)</code>	Arc sine
<code>Tasinh(T)</code>	Inverse hyperbolic sine
<code>Tasinpi(T x)</code>	asin(x) / pi
<code>Tatan(T y_over_x)</code>	Arc tangent
<code>Tatan2(T y, T x)</code>	Arc tangent of y / x
<code>Tatanh(T)</code>	Hyperbolic arc tangent
<code>Tatanpi(T x)</code>	atan(x) / pi
<code>Tatan2pi(T x, T y)</code>	atan2(x, y) / pi
<code>Tcbrt(T)</code>	cube root
<code>Tceil(T)</code>	Round to integer toward +infinity
<code>Tcopysign(T x, T y)</code>	x with sign changed to sign of y
<code>Tcos(T)</code> HN	cosine
<code>Tcosh(T)</code>	hyperbolic cosine
<code>Tcospi(T x)</code>	cos(x) / pi
<code>T half_divide(T x, T y)</code> <code>T nat ve_divide(T x, T y)</code>	x / y
<code>Terfc(T)</code>	Complementary error function
<code>T erf(T)</code>	Calculates error function of T
<code>Texp(T x)</code> HN	Exponential base e
<code>Texp2(T)</code> HN	Exponential base 2

<code>Texp10(T)</code> HN	Exponential base 10
<code>Texpm1(T x)</code>	e^x - 1.0
<code>Tfabs(T)</code>	Absolute value
<code>Tfdim(T x, T y)</code>	"Positive difference" between x and y
<code>T floor(T)</code>	Round to integer toward -infinity
<code>Tfma(T a, T b, T c)</code>	Multiply and add, then round
<code>Tfmax(T x, T y)</code> <code>halfn_fmax(halfn x, halfn y)</code>	Return y if x < y, otherwise it returns x
<code>Tfmin(T x, T y)</code> <code>halfn_fmin(halfn x, halfn y)</code>	Return y if y < x, otherwise it returns x
<code>Tfmod(T x, T y)</code>	Modulus. Returns x - y * trunc(x/y)
<code>T fract(T x, T *iptr)</code>	Fractional value in x
<code>T 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>intn_ilogb(T x)</code>	Return exponent as an integer value
<code>T ldexp(T x, intn n)</code> <code>T ldexp(T x, int n)</code>	x * 2^n
<code>Tlgamma(T x)</code> <code>Tlgamma_r(T x, intn *signp)</code>	Log gamma function
<code>Tlog(T)</code> HN	Natural logarithm
<code>Tlog2(T)</code> HN	Base 2 logarithm
<code>Tlog10(T)</code> HN	Base 10 logarithm
<code>Tlog1p(T x)</code>	ln(1.0 + x)
<code>Tlogb(T x)</code>	exponent of x
<code>Tmad(T a, T b, T c)</code>	Approximates a * b + c
<code>T modf(T x, T *iptr)</code>	Decompose a floating-point number

<code>floatn_nan(uintn nancode)</code> <code>floatn_nan(uintn nancode)</code> <code>halfn_nan(ushortn nancode)</code> <code>doublen_nan(ulongn nancode)</code> <code>doublen_nan(uintn nancode)</code>	Quiet NaN
<code>Tnextafter(T x, T y)</code>	Next representable float ng-point value following x in the direction of y
<code>Tpow(T x, T y)</code>	Compute x to the power of y (x^y)
<code>Tpown(T x, intn y)</code>	Compute x^y, where y is an integer
<code>Tpowr(T x, T y)</code> HN	Compute x^y, where x is >= 0
<code>T half_recip(T x)</code> <code>T nat ve_recip(T x)</code>	1 / x
<code>Tremainder(T x, T y)</code>	Floating-point remainder function
<code>Tremquo(T x, T y, intn *quo)</code>	Floating-point remainder and quotient function
<code>Trint(T)</code>	Round integer to nearest even integer
<code>Trootn(T x, intn n)</code>	Compute x to the power of 1/y
<code>Tround(T x)</code>	Integral value nearest to x rounding
<code>Tsqrt(T)</code> HN	Inverse square root
<code>Tsin(T)</code> HN	sine
<code>Tsincos(T x, T *cosval)</code>	sine and cosine of x
<code>Tsinh(T)</code>	hyperbolic sine
<code>Tsinpi(T x)</code>	sin(x) / pi
<code>Tsqrt(T)</code> HN	square root
<code>Ttan(T)</code> HN	tangent
<code>Ttanh(T)</code>	hyperbolic tangent
<code>Ttanpi(T x)</code>	tan(x) / pi
<code>Tgamma(T)</code>	gamma function
<code>T trunc(T)</code>	Round to integer toward zero

Geometric Built-in Functions [6.11.5]

Vector types may have 2 or 4 components. Optional extensions enable double, doublen, and halfn types.

<code>float4_cross(float4 p0, float4 p1)</code> <code>double4_cross(double4 p0, double4 p1)</code> <code>half4_cross(half4 p0, half4 p1)</code>	Cross product
<code>float_dot(float p0, float p1)</code> <code>float_dot(floatn p0, floatn p1)</code> <code>double_dot(double p0, double p1)</code> <code>double_dot(doublen p0, doublen p1)</code> <code>half_dot(half p0, half p1)</code> <code>half_dot(halfn p0, halfn p1)</code>	Dot product

<code>float_distance(float p0, float p1)</code> <code>float_distance(floatn p0, floatn p1)</code> <code>double_distance(double p0, double p1)</code> <code>double_distance(doublen p0, doublen p1)</code> <code>half_distance(half p0, half p1)</code> <code>half_distance(halfn p0, halfn p1)</code>	Vector distance
<code>float_length(float p)</code> <code>float_length(floatn p)</code> <code>double_length(double p)</code> <code>double_length(doublen p)</code> <code>half_length(half p)</code> <code>half_length(halfn p)</code>	Vector length

<code>float_normalize(float p)</code> <code>floatn_normalize(floatn p)</code> <code>double_normalize(double p)</code> <code>doublen_normalize(doublen p)</code> <code>half_normalize(half p)</code> <code>halfn_normalize(halfn p)</code>	Normal vector length 1
<code>float_fast_distance(float p0, float p1)</code> <code>floatn_fast_distance(floatn p0, floatn p1)</code>	Vector distance
<code>float_fast_length(float p)</code> <code>floatn_fast_length(floatn p)</code>	Vector length
<code>float_fast_normalize(float p)</code> <code>floatn_fast_normalize(floatn p)</code>	Normal vector length 1

Each occurrence of *T* within a function call must be the same. In vector types, *n* is 2, 4, 8, or 16 unless otherwise specified.

HN = Half and Nat ve variants are available. **half_** and **nat ve_** variants are shown in purple.
OPT = Optional function.

More built-in functions on the reverse >

Relational Built-in Functions [6.11.6]

T is type float, floatn, char, charn, uchar, uchar_n, short, shortn, ushort, ushortn, int, intn, uint, uintn, long, longn, ulong, or ulongn (and optionally double, doublen). *S* is type char, charn, short, shortn, int, intn, long, or longn. *U* is type uchar, uchar_n, ushort, ushortn, uint, uintn, long, or longn. Optional extensions enable double, doublen, and halfn types.

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

int isninf (float) intn isninf (floatn) int isninf (double) longn isninf (doublen) int isninf (half) shortn isninf (halfn)	Test for +ve or -ve infinity
int isnan (float) intn isnan (floatn) int isnan (double) longn isnan (doublen) int isnan (half) shortn isnan (halfn)	Test for a NaN
int isnormal (float) intn isnormal (floatn) int isnormal (double) longn isnormal (doublen) int isnormal (half) shortn isnormal (halfn)	Test for a normal value
int isordered (float x, float y) intn isordered (floatn x, floatn y) int isordered (double x, double y) longn isordered (doublen x, doublen y) int isordered (half x, half y) shortn isordered (halfn x, halfn y)	Test if arguments are ordered
int isunordered (float x, float y) intn isunordered (floatn x, floatn y) int isunordered (double x, double y) longn isunordered (doublen x, doublen y) int isunordered (half x, half y) shortn isunordered (halfn x, halfn y)	Test if arguments are unordered
int signbit (float) intn signbit (floatn) int signbit (double) longn signbit (doublen) int signbit (half) shortn signbit (halfn)	Test for sign bit
int any (S x)	1 if MSB in any component of x is set; else 0
int all (S x)	1 if MSB in all components of x are set; else 0
T bitselct (T a, T b, T c) halfn bitselct (halfn a, halfn b, halfn c) doublen bitselct (doublen a, doublen b, doublen c)	Each bit of result is corresponding bit of a if corresponding bit of c is 0
T select (T a, T b, S c) T select (T a, T b, U c) doublen select (doublen, doublen, longn)	For each component of a vector type, result[i] = if MSB of c[i] is set? b[i] : a[i] For scalar type, result = c? b : a

Vector Data Load/Store Built-in Functions [6.11.7]

Q is an Address Space Qualifier listed in 6.5 unless otherwise noted. *R* defaults to the current rounding mode, or is one of the Rounding Modes listed in 6.2.3.2. *T* is type char, uchar, short, ushort, int, long, ulong, half, or float (or optionally double). *Tn* refers to the vector form of type *T*. Optional extension enables the double and doublen types.

Tn vloadn (size_t of set, const Q T* p)	Read vector data from memory
void vstoren (Tn data, size_t of set, Q T* p)	Write vector data to memory (<i>Q</i> in this function cannot be __constant)
float vload_half (size_t of set, const Q half* p)	Read a half from memory
floatn vload_half (size_t of set, const Q half* p)	Read multiple halves from memory
void vstore_half (float data, size_t of set, Q half* p) void vstore_half_R (float data, size_t of set, Q half* p) void vstore_half (double data, size_t of set, Q half* p) void vstore_half_R (double data, size_t of set, Q half* p)	Write a half to memory (<i>Q</i> in this function cannot be __constant)
void vstore_halfn (floatn data, size_t of set, Q half* p) void vstore_halfn_R (floatn data, size_t of set, Q half* p) void vstore_halfn (doublen data, size_t of set, Q half* p) void vstore_halfn_R (doublen data, size_t of set, Q half* p)	Write a half vector to memory (<i>Q</i> in this function cannot be __constant)
floatn vloada_halfn (size_t of set, const Q half* p)	sizeof (floatn) bytes of data read from locat on (p + (of set * n))
void vstorea_halfn (floatn data, size_t of set, Q half* p) void vstorea_halfn_R (floatn data, size_t of set, Q half* p) void vstorea_halfn (doublen data, size_t of set, Q half* p) void vstorea_halfn_R (doublen data, size_t of set, Q half* p)	Write a half vector to vector-aligned memory (<i>Q</i> in this function cannot be __constant)

Async Copies and Prefetch Built-in Functions [6.11.11]

T is type char, charn, uchar, uchar_n, short, shortn, ushort, ushortn, int, intn, uint, uintn, long, longn, ulong, ulongn, float, floatn, and optionally double, doublen. Optional extension enables the double and doublen types.

event_t async_work_group_copy (local T* dst, const global T* src, size_t num_elements, event_t event)	Copies <i>T</i> elements from src to dst
event_t async_work_group_copy (global T* dst, const local T* src, size_t num_elements, event_t event)	
void wait_group_events (int num_events, event_t* event_list)	Wait for events that identify the async_work_group_copy operations to complete.
void prefetch (const global T* p, size_t num_elements)	Prefetch num_elements * sizeof(T) bytes into the global cache.

Optional Extension: Atomic Functions [9.5]

Q is qualifier __global or __local. *T* is type int or unsigned int for 32-bit atomic functions. *T* is type long or ulong for 64-bit atomic functions.

To use the base or extended atomic functions, include this pragma in your application:

```
#pragma OPENCL_EXTENSION extension-name: enable
```

For base atomic functions, extension-name is one of:

```
d_khr_global_int32_base_atomics  
d_khr_local_int32_base_atomics  
d_khr_int64_base_atomics
```

For extended atomic functions, extension-name is one of:

```
d_khr_global_int32_extended_atomics  
d_khr_local_int32_extended_atomics  
d_khr_int64_extended_atomics
```

Base atomic functions

T atom_add (Q T* p, T val)	Read, add, and store
T atom_sub (Q T* p, T val)	Read, sub, and store
T atom_xchg (Q T* p, T val)	Read, swap, and store
T atom_inc (Q T* p)	Read, increment, and store
T atom_dec (Q T* p)	Read, decrement, and store
T atom_cmpxchg (Q T* p, T cmp, T val)	Read and store (*p == cmp) ? val : *p

Extended atomic functions

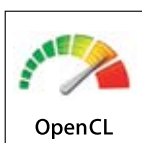
T atom_min (Q T* p, T val)	Read, store min(*p, val)
T atom_max (Q T* p, T val)	Read, store max(*p, val)
T atom_and (Q T* p, T val)	Read, store (*p & val)
T atom_or (Q T* p, T val)	Read, store (*p val)
T atom_xor (Q T* p, T val)	Read, store (*p ^ val)

Synchronization and Explicit Memory Fence Built-in Functions [6.11.9, 6.11.10]

The flags argument specifies the memory address space and can be set to a combination of CLK_LOCAL_MEM_FENCE and CLK_GLOBAL_MEM_FENCE.

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

More built-in functions on the reverse >



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

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

clCreateImage2D or clCreateImage3D. OPT

clCreateImage2D or clCreateImage3D. OPT

cl_read_imagef	U	U					
cl_read_imageh	U	U					
cl_read_imagei	U	U					
cl_read_imagej	U	U					
cl_read_imageui	U	U					
cl_read_imageui	U	U					
cl_read_imageh	U	U					
cl_read_imageh	U	U					
cl_write_imagef	U						
cl_write_imageh	U						
cl_write_imagei	U						
cl_write_imagej	U						
cl_write_imageui	U						
cl_write_imageh	U						
cl_read_imageui	U	U					
cl_read_imageui	U	U					
cl_read_imageh	U	U					
cl_read_imageh	U	U					
cl_get_image_width							
cl_get_image_width							
cl_get_image_height							
cl_get_image_height							
cl_get_image_depth							
cl_get_image_channel_data_type							
cl_get_image_channel_data_type							
cl_get_image_channel_order							
cl_get_image_channel_order							
cl_get_image_dim							
cl_get_image_dim							
cl_write_imagef	U						
cl_write_imageh	U						
cl_write_imagei	U						
cl_write_imagej	U						
cl_write_imageui	U						

OpenCL/OpenGL Sharing APIs [Appendix B]

clCreateFromGLTexture2DU or clCreateFromGLTexture3DU ensures that the

cl_mem	cl_context	context	U				
int							
CL_MEM_READ_WRITE							
cl_mem	cl_context	context	U				
GLenum							
int							
cl_mem	cl_context	context	U				
GLenum							
int							
cl_mem	cl_context	context	U				
GLenum							
int							
cl_mem	cl_context	context	U				
GLuint							
cl_int	clGetGLObjectInfo	(cl_mem u)	U				
cl_gl_object_type							
GLuint							