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 available at www.khronos.org/opencl.

The OpenCL Runtime

Command Queues [5.1]

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

properties: CL_QUEUE_PROFILING_ENABLE, CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE

```
cl_int clRetainCommandQueue (
    cl_command_queue command_queue)

cl_int clReleaseCommandQueue (
    cl_command_queue command_queue)

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

param_name: CL_QUEUE_CONTEXT, CL_QUEUE_DEVICE, CL_QUEUE_REFERENCE_COUNT, CL_QUEUE_PROPERTIES

The OpenCL Platform Layer

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

Contexts [4.3]

```
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: CL_CONTEXT_PLATFORM, CL_GL_CONTEXT_KHR, CL_CGL_SHAREGROUP_KHR, CL_{EGL, GLX}_DISPLAY_KHR, CL_WGL_HDC_KHR

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

properties: See **clCreateContext**

```
cl_int clRetainContext (cl_context context)

cl_int clReleaseContext (cl_context context)

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

param_name: CL_CONTEXT_REFERENCE_COUNT, CL_CONTEXT_{DEVICES, PROPERTIES}, CL_CONTEXT_NUM_DEVICES

Querying Platform Info and Devices [4.1, 4.2]

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

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

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

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

device_type: CL_DEVICE_TYPE_{CPU, GPU}, CL_DEVICE_TYPE_{ACCELERATOR, DEFAULT, ALL}

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

param_name: CL_DEVICE_TYPE,
CL_DEVICE_VENDOR_ID,
CL_DEVICE_MAX_COMPUTE_UNITS,
CL_DEVICE_MAX_WORK_ITEM_{DIMENSIONS, SIZES},
CL_DEVICE_MAX_WORK_GROUP_SIZE,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_CHAR,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_SHORT,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_INT,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_LONG,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_FLOAT,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_DOUBLE,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_HALF,
CL_DEVICE_MAX_CLOCK_FREQUENCY,
CL_DEVICE_ADDRESS_BITS,
CL_DEVICE_MAX_MEM_ALLOC_SIZE,
CL_DEVICE_IMAGE_SUPPORT,
CL_DEVICE_MAX_{READ, 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, SIZE},
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE,
CL_DEVICE_GLOBAL_MEM_SIZE,
CL_DEVICE_MAX_CONSTANT_{BUFFER_SIZE, ARGS}
CL_DEVICE_LOCAL_MEM_{TYPE, 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, VENDOR, PROFILE, EXTENSIONS},
CL_DEVICE_HOST_UNIFIED_MEMORY,
CL_DEVICE_OPENCL_C_VERSION,
CL_DEVICE_VERSION,
CL_DRIVER_VERSION, CL_DEVICE_PLATFORM

Buffer Objects

Elements of a buffer object can be a scalar or vector data type or a user-defined structure. Elements are stored sequentially and are accessed using a pointer by a kernel executing on a device. Data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

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

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 for **clCreateBuffer** and **clCreateSubBuffer**:
CL_MEM_READ_WRITE,
CL_MEM_{WRITE, READ}_ONLY,
CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

Read, Write, Copy Buffer Objects [5.2.2]

```
cl_int clEnqueueReadBuffer (
    cl_command_queue command_queue, cl_mem buffer,
    cl_bool blocking_read, size_t offset, size_t cb,
    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 cb,
    const 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[3],
    const size_t host_origin[3], const size_t region[3],
    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 clEnqueueWriteBufferRect (
    cl_command_queue command_queue, cl_mem buffer,
    cl_bool blocking_write, const size_t buffer_origin[3],
    const size_t host_origin[3], const size_t region[3],
    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 clEnqueueCopyBuffer (
    cl_command_queue command_queue,
    cl_mem src_buffer, cl_mem dst_buffer, size_t src_offset,
    size_t dst_offset, size_t cb,
    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[3], const size_t dst_origin[3],
    const size_t region[3], 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 [5.2.2]

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

Map Buffer Objects [5.4.1-2]

```
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)
```

Query Buffer Object [5.4.3]

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

param_name: CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR}, CL_MEM_{MAP, REFERENCE}_COUNT, CL_MEM_OFFSET, CL_MEM_CONTEXT, CL_MEM_ASSOCIATED_MEMOBJECT

Program Objects

Create Program Objects [5.6.1]

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

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

cl_int clRetainProgram (cl_program program)

cl_int clReleaseProgram (cl_program program)
```

Build Program Executable [5.6.2]

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

Build Options [5.6.3]

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

```
-D name        -D name=definition        -I dir
```

Optimization options:

```
-cl-opt-disable                    -cl-mad-enable
-cl-no-signed-zeros                -cl-finite-math-only
-cl-fast-relaxed-math
-cl-unsafe-math-optimizations
```

Math Intrinsics:

```
-cl-single-precision-constant        -cl-denorms-are-zero
```

Warning request/suppress:

```
-w                 -Werror
```

Control OpenCL C language version:

```
-cl-std=CL1.1      // OpenCL 1.1 specification.
```

Query Program Objects [5.6.5]

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

param_name: CL_PROGRAM_{REFERENCE_COUNT}, CL_PROGRAM_{CONTEXT, NUM_DEVICES, DEVICES}, CL_PROGRAM_{SOURCE, BINARY_SIZES, BINARIES}

(Program Objects Continue >)

Program Objects (continued)

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

param_name: CL_PROGRAM_BUILD_{STATUS, OPTIONS, LOG}

Unload the OpenCL Compiler [5.6.4]

cl_int **clUnloadCompiler** (void)

Supported Data Types

Built-in Scalar Data Types [6.1.1]

OpenCL Type	API Type	Description
bool	--	true (1) or false (0)
char	cl_char	8-bit signed
unsigned char, uchar	cl_uchar	8-bit unsigned
short	cl_short	16-bit signed
unsigned short, ushort	cl_ushort	16-bit unsigned
int	cl_int	32-bit signed
unsigned int, uint	cl_uint	32-bit unsigned
long	cl_long	64-bit signed
unsigned long, ulong	cl_ulong	64-bit unsigned
float	cl_float	32-bit float
half	cl_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	void

Built-in Vector Data Types [6.1.2]

OpenCL Type	API Type	Description
charn	cl_charn	8-bit signed
ucharn	cl_ucharn	8-bit unsigned
shortn	cl_shortn	16-bit signed
ushortn	cl_ushortn	16-bit unsigned
intn	cl_intn	32-bit signed
uintn	cl_uintn	32-bit unsigned
longn	cl_longn	64-bit signed
ulongn	cl_ulongn	64-bt unsigned
floatn	cl_floatn	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
booln	boolean vector
double, doublen OPTIONAL	64-bit float, vector
halfn	16-bit, vector
quad, quadn	128-bit float, vector
complex half, complex halfn imaginary half, imaginary halfn	16-bit complex, vector
complex float, complex floatn imaginary float, imaginary floatn	32-bit complex, vector
complex double, complex doublen imaginary double, imaginary doublen	64-bit complex, vector
complex quad, complex quadn imaginary quad, imaginary quadn	128-bit complex, vector
floatnxm	n*m matrix of 32-bit floats
doublenxm	n*m matrix of 64-bit floats
long double, long doublen	64 - 128-bit float, vector
long long, long longnb	128-bit signed
unsigned long long, ulong long, ulong longn	128-bit unsigned

Kernel and Event Objects

Create Kernel Objects [5.7.1]

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

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

cl_int **clRetainKernel** (cl_kernel kernel)

cl_int **clReleaseKernel** (cl_kernel kernel)

Kernel Args. & Object Queries [5.7.2, 5.7.3]

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

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

param_name: CL_KERNEL_FUNCTION_NAME,
  CL_KERNEL_NUM_ARGS, CL_KERNEL_REFERENCE_COUNT,
  CL_KERNEL_CONTEXT, CL_KERNEL_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_WORK_GROUP_SIZE,
  CL_KERNEL_COMPILE_WORK_GROUP_SIZE,
  CL_KERNEL_{LOCAL, PRIVATE}_MEM_SIZE,
  CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_MULTIPLE

Execute Kernels [5.8]

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

cl_int **clEnqueueTask** (
  cl_command_queue command_queue, cl_kernel
  kernel, cl_uint num_events_in_wait_list,
  const cl_event *event_wait_list, cl_event *event)

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

Event Objects [5.9]

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

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

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

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

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

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

cl_int **clRetainEvent** (cl_event event)

cl_int **clReleaseEvent** (cl_event event)

Out-of-order Execution of Kernels & Memory Object Commands [5.10]

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

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

cl_int **clEnqueueBarrier** (
  cl_command_queue command_queue)

Profiling Operations [5.11]

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

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

Flush and Finish [5.12]

cl_int **clFlush** (cl_command_queue command_queue)

cl_int **clFinish** (cl_command_queue command_queue)

Vector Component Addressing [6.1.7]

Vector Components

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

Vector Addressing Equivalencies

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

	v.lo	v.hi	v.odd	v.even
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

	v.lo	v.hi	v.odd	v.even
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.

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 is rounding mode

R can be one of the following rounding modes:

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

Each occurrence of *T* within a function call must be the same. *n* is 2, 3, 4, 8, or 16 unless otherwise specified.

Preprocessor Directives & Macros [6.9]

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

__FILE__	Current source file
__LINE__	Integer line number
__OPENCL_VERSION__	Integer version number
CL_VERSION_1_0__	Substitutes integer 100 for version 1.0
CL_VERSION_1_1__	Substitutes integer 110 for version 1.1
__ENDIAN_LITTLE__	1 if device is little endian
__kernel_exec(X, type*n*)	Same as: __kernel __attribute__((work_group_size_hint(*X*, 1, 1))) \ __attribute__((vec_type_hint(*typen*)))
__IMAGE_SUPPORT__	1 if images are supported
__FAST_RELAXED_MATH__	1 if –cl-fast-relaxed-math optimization option is specified

Specify Type Attributes [6.10.1]

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

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

Math Constants [6.11.2]

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

MAXFLOAT	Value of max. non-infinite single-precision floating-point number.
HUGE_VALF	Positive float expression, evaluates to +infinity. Used as error value.
HUGE_VAL	Positive double expression, evals. to +infinity. Used as error value. OPTIONAL
INFINITY	Constant float expression, positive or unsigned infinity.
NAN	Constant float expression, quiet NaN.
M_E_F	Value of e
M_LOG2E_F	Value of log2e
M_LOG10E_F	Value of log10e

M_LN2_F	Value of loge2
M_LN10_F	Value of loge10
M_PI_F	Value of π
M_PI_2_F	Value of π / 2
M_PI_4_F	Value of π / 4
M_1_PI_F	Value of 1 / π
M_2_PI_F	Value of 2 / π
M_2_SQRTPI_F	Value of 2 / √π
M_SQRT2_F	Value of √2
M_SQRT1_2_F	Value of 1 / √2

Work-Item Built-in Functions [6.11.1]

D is dimension index.

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

Common Built-in Functions [6.11.4]

T is type float or float*n* (or optionally double, double*n*, or half*n*).
Optional extensions enable double, double*n*, and half*n* types.

T **clamp** (*T x*, *T min*, *T max*) float*n* **clamp** (float*n x*, float *min*, float *max*) double*n* **clamp** (double*n x*, double *min*, double *max*) half*n* **clamp** (half*n x*, half *min*, half *max*)	Clamp *x* to range given by *min*, *max*
T **degrees** (*T radians*)	*radians* to degrees
T **max** (*T x*, *T y*) float*n* **max** (float*n x*, float *y*) double*n* **max** (double*n x*, double *y*) half*n* **max** (half*n x*, half *y*)	Max of *x* and *y*
T **min** (*T x*, *T y*) float*n* **min** (float*n x*, float *y*) double*n* **min** (double*n x*, double *y*) half*n* **min** (half*n x*, half *y*)	Min of *x* and *y*
T **mix** (*T x*, *T y*, *T a*) float*n* **mix** (float*n x*, float *y*, float *a*) double*n* **mix** (double*n x*, double *y*, double *a*) half*n* **mix** (half*n x*, half *y*, half *a*)	Linear blend of *x* and *y*
T **radians** (*T degrees*)	*degrees* to radians
T **step** (*T edge*, *T x*) float*n* **step** (float *edge*, float*n x*) double*n* **step** (double *edge*, double*n x*) half*n* **step** (half *edge*, half*n x*)	0.0 if *x* < *edge*, else 1.0
T **smoothstep** (*T edge0*, *T edge1*, *T x*) float*n* **smoothstep** (float *edge0*, float *edge1*, float*n x*) double*n* **smoothstep** (double *edge0*, double *edge1*, double*n x*) half*n* **smoothstep** (half *edge0*, half *edge1*, half*n x*)	Step and interpolate
T **sign** (*T x*)	Sign of *x*

Integer Built-in Functions [6.11.3]

T is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, or ulong*n*.
U is the unsigned version of ***T***. ***S*** is the scalar version of ***T***.

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

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

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

The following fast integer functions optimize the performance of kernels. In these functions, ***T*** is type int, int2, int3, int4, int8, int16, uint, uint2, uint4, uint8 or uint16.

T **mad24** (*T a*, *T b*, *T c*)	Multiply 24-bit int. values *a*, *b*, add 32-bit int. result to 32-bit int. *c*
T **mul24** (*T a*, *T b*)	Multiply 24-bit int. values *a* and *b*

Math Built-in Functions [6.11.2]

T is type float or float*n* (or optionally double, double*n*, or half*n*). int*n*, uint*n*, and ulong*n* must be scalar when *T* is scalar. ***Q*** is qualifier __global, __local, or __private. **HN** indicates that Half and Native variants are available by prepending "half_" or "native_" to function name. Prototypes shown in purple are half_ and native_ only. **Optional extensions enable double, double*n*, half, and half*n* types.**

T **acos** (*T*)	Arc cosine
T **acosh** (*T*)	Inverse hyperbolic cosine
T **acospi** (*T x*)	acos (*x*) / π
T **asin** (*T*)	Arc sine
T **asinh** (*T*)	Inverse hyperbolic sine
T **asinpi** (*T x*)	asin (*x*) / π
T **atan** (*T y_over_x*)	Arc tangent
T **atan2** (*T y*, *T x*)	Arc tangent of *y* / *x*
T **atanh** (*T*)	Hyperbolic arc tangent
T **atanpi** (*T x*)	atan (*x*) / π
T **atan2pi** (*T x*, *T y*)	atan2 (*x*, *y*) / π
T **cbrt** (*T*)	Cube root
T **ceil** (*T*)	Round to integer toward + infinity
T **copysign** (*T x*, *T y*)	*x* with sign changed to sign of *y*
T **cos** (*T*) **HN**	Cosine
T **cosh** (*T*)	Hyperbolic consine
T **cospi** (*T x*)	cos (π *x*)
T **half_divide** (*T x*, *T y*) *T* **native_divide** (*T x*, *T y*)	*x* / *y* (*T* may be float or float*n*)
T **erfc** (*T*)	Complementary error function
T **erf** (*T*)	Calculates error function of *T*
T **exp** (*T x*) **HN**	Exponential base e
T **exp2** (*T*) **HN**	Exponential base 2
T **exp10** (*T*) **HN**	Exponential base 10
T **expm1** (*T x*)	e^*x* -1.0
T **fabs** (*T*)	Absolute value
T **fdim** (*T x*, *T y*)	"Positive difference" between *x* and *y*
T **floor** (*T*)	Round to integer toward - infinity
T **fma** (*T a*, *T b*, *T c*)	Multiply and add, then round
T **fmax** (*T x*, *T y*) half*n* **fmax** (half*n x*, half *y*) float*n* **fmax**(float*n x*, float *y*) double*n* **fmax**(double*n x*, double *y*)	Return *y* if *x* < *y*, otherwise it returns *x*
T **fmin** (*T x*, *T y*) half*n* **fmin** (half*n x*, half *y*) float*n* **fmin**(float*n x*, float *y*) double*n* **fmin**(double*n x*, double *y*)	Return *y* if *y* < *x*, otherwise it returns *x*
T **fmod** (*T x*, *T y*)	Modulus. Returns *x* – *y* * trunc (*x*/*y*)
T **fract** (*T x*, *Q T* **iptr*)	Fractional value in *x*
T **frexp** (*T x*, *Q* int*n* **exp*)	Extract mantissa and exponent
T **hypot** (*T x*, *T y*)	Square root of *x*^2+ *y*^2
int*n* **ilogb** (*T x*)	Return exponent as an integer value
T **ldexp** (*T x*, int*n n*) *T* **ldexp** (*T x*, int *n*)	*x* * 2^*n*
T **lgamma** (*T x*) *T* **lgamma_r** (*T x*, *Q* int*n* **signp*)	Log gamma function
T **log** (*T*) **HN**	Natural logarithm
T **log2** (*T*) **HN**	Base 2 logarithm
T **log10** (*T*) **HN**	Base 10 logarithm
T **log1p** (*T x*)	ln (1.0 + *x*)
T **logb** (*T x*)	Exponent of *x*
T **mad** (*T a*, *T b*, *T c*)	Approximates *a* * *b* + *c*
T **maxmag** (*T x*, *T y*)	Maximum magnitude of *x* and *y*
T **minmag** (*T x*, *T y*)	Minimum magnitude of *x* and *y*
T **modf** (*T x*, *Q T* **iptr*)	Decompose a floating-point number
float **nan** (uint*n nancode*) float*n* **nan** (uint*n nancode*) half*n* **nan** (ushort*n nancode*) double*n* **nan** (ulong*n nancode*)	Quiet NaN
T **nextafter** (*T x*, *T y*)	Next representable floating-point value following *x* in direction of *y*
T **pow** (*T x*, *T y*)	Compute *x* to the power of *y* (*x*^*y*)
T **pown** (*T x*, int*n y*)	Compute *x*^*y*, where *y* is an integer
T **powr** (*T x*, *T y*) **HN**	Compute *x*^*y*, where *x* is >= 0
T **half_recip** (*T x*) *T* **native_recip** (*T x*)	1 / *x* (*T* may be float or float*n*)
T **remainder** (*T x*, *T y*)	Floating point remainder
T **remquo** (*T x*, *T y*, *Q* int*n* **quo*)	Floating point remainder and quotient
T **rint** (*T*)	Round to nearest even integer
T **rootn** (*T x*, int*n y*)	Compute *x* to the power of 1/*y*
T **round** (*T x*)	Integral value nearest to *x* rounding
T **rsqrt** (*T*) **HN**	Inverse square root
T **sin** (*T*) **HN**	Sine
T **sincos** (*T x*, *Q T* **cosval*)	Sine and cosine of *x*
T **sinh** (*T*)	Hyperbolic sine
T **sinpi** (*T x*)	sin (π *x*)
T **sqrt** (*T*) **HN**	Square root
T **tan** (*T*) **HN**	Tangent
T **tanh** (*T*)	Hyperbolic tangent
T **tanpi** (*T x*)	tan (π *x*)
T **tgamma** (*T*)	Gamma function
T **trunc** (*T*)	Round to integer toward zero

Geometric Built-in Functions [6.11.5]

Vector types may have 2, 3, or 4 components. **Optional extensions enable double, double*n*, and half*n* types.**

float **dot** (float *p0*, float *p1*)	Dot product
float **dot** (float*n p0*, float*n p1*)	
double **dot** (double *p0*, double *p1*)	
double **dot** (double*n p0*, double*n p1*)	
half **dot** (half *p0*, half *p1*)	
half **dot** (half*n p0*, half*n p1*)	
float{3,4} **cross** (float{3,4} *p0*, float{3,4} *p1*)	Cross product
double{3,4} **cross** (double{3,4} *p0*, double{3,4} *p1*)	
half{3,4} **cross** (half{3,4} *p0*, half{3,4} *p1*)	

float **distance** (float *p0*, float *p1*)	Vector distance
float **distance** (float*n p0*, float*n p1*)	
double **distance** (double *p0*, double *p1*)	
double **distance** (double*n p0*, double*n p1*)	
half **distance** (half *p0*, half *p1*)	
half **distance** (half*n p0*, half*n p1*)	
float **length** (float *p*)	Vector length
float **length** (float*n p*)	
double **length** (double *p*)	
double **length** (double*n p*)	
half **length** (half *p*)	
half **length** (half*n p*)	

float **normalize** (float *p*)	Normal vector length 1
float*n* **normalize** (float*n p*)	
double **normalize** (double *p*)	
double*n* **normalize** (double*n p*)	
half **normalize** (half *p*)	
half*n* **normalize** (half*n p*)	
float **fast_distance** (float *p0*, float *p1*)	Vector distance
float **fast_distance** (float*n p0*, float*n p1*)	
float **fast_length** (float *p*)	Vector length
float **fast_length** (float*n p*)	
float **fast_normalize** (float *p*)	Normal vector length 1
float*n* **fast_normalize** (float*n p*)	

Relational Built-in Functions [6.11.6]

T is type float, float*n*, char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, or ulong*n* (and optionally double, double*n*). ***S*** is type char, char*n*, short, short*n*, int, int*n*, long, or long*n*. ***U*** is type uchar, uchar*n*, ushort, ushort*n*, uint, uint*n*, ulong, or ulong*n*. **Optional extensions enable double, double*n*, and half*n* types.**

int **isequal** (float *x*, float *y*)	Compare of *x* == *y*
int*n* **isequal** (float*n x*, float*n y*)	
int **isequal** (double *x*, double *y*)	
long*n* **isequal** (double*n x*, double*n y*)	
int **isequal** (half *x*, half *y*)	
short*n* **isequal** (half*n x*, half*n y*)	
int **isnotequal** (float *x*, float *y*)	Compare of *x* != *y*
int*n* **isnotequal** (float*n x*, float*n y*)	
int **isnotequal** (double *x*, double *y*)	
long*n* **isnotequal** (double*n x*, double*n y*)	
int **isnotequal** (half *x*, half *y*)	
short*n* **isnotequal** (half*n x*, half*n y*)	
int **isgreater** (float *x*, float *y*)	Compare of *x* > *y*
int*n* **isgreater** (float*n x*, float*n y*)	
int **isgreater** (double *x*, double *y*)	
long*n* **isgreater** (double*n x*, double*n y*)	
int **isgreater** (half *x*, half *y*)	
short*n* **isgreater** (half*n x*, half*n y*)	
int **isgreaterequal** (float *x*, float *y*)	Compare of *x* >= *y*
int*n* **isgreaterequal** (float*n x*, float*n y*)	
int **isgreaterequal** (double *x*, double *y*)	
long*n* **isgreaterequal** (double*n x*, double*n y*)	
int **isgreaterequal** (half *x*, half *y*)	
short*n* **isgreaterequal** (half*n x*, half*n y*)	
int **isless** (float *x*, float *y*)	Compare of *x* < *y*
int*n* **isless** (float*n x*, float*n y*)	
int **isless** (double *x*, double *y*)	
long*n* **isless** (double*n x*, double*n y*)	
int **isless** (half *x*, half *y*)	
short*n* **isless** (half*n x*, half*n y*)	
int **islessequal** (float *x*, float *y*)	Compare of *x* <= *y*
int*n* **islessequal** (float*n x*, float*n y*)	
int **islessequal** (double *x*, double *y*)	
long*n* **islessequal** (double*n x*, double*n y*)	
int **islessequal** (half *x*, half *y*)	
short*n* **islessequal** (half*n x*, half*n y*)	
int **islessgreater** (float *x*, float *y*)	Compare of (*x* < *y*) || (*x* > *y*)
int*n* **islessgreater** (float*n x*, float*n y*)	
int **islessgreater** (double *x*, double *y*)	
long*n* **islessgreater** (double*n x*, double*n y*)	
int **islessgreater** (half *x*, half *y*)	
short*n* **islessgreater** (half*n x*, half*n y*)	
int **isfinite** (float)	Test for finite value
int*n* **isfinite** (float*n*)	
int **isfinite** (double)	
long*n* **isfinite** (double*n*)	
int **isfinite** (half)	
short*n* **isfinite** (half*n*)	

int **isinf** (float)	Test for +ve or –ve infinity
int*n* **isinf** (float*n*)	
int **isinf** (double)	
long*n* **isinf** (double*n*)	
int **isinf** (half)	
short*n* **isinf** (half*n*)	
int **isnan** (float)	Test for a NaN
int*n* **isnan** (float*n*)	
int **isnan** (double)	
long*n* **isnan** (double*n*)	
int **isnan** (half)	
short*n* **isnan** (half*n*)	
int **isnormal** (float)	Test for a normal value
int*n* **isnormal** (float*n*)	
int **isnormal** (double)	
long*n* **isnormal** (double*n*)	
int **isnormal** (half)	
short*n* **isnormal** (half*n*)	
int **isordered** (float *x*, float *y*)	Test if arguments are ordered
int*n* **isordered** (float*n x*, float*n y*)	
int **isordered** (double *x*, double *y*)	
long*n* **isordered** (double*n x*, double*n y*)	
int **isordered** (half *x*, half *y*)	
short*n* **isordered** (half*n x*, half*n y*)	
int **isunordered** (float *x*, float *y*)	Test if arguments are unordered
int*n* **isunordered** (float*n x*, float*n y*)	
int **isunordered** (double *x*, double *y*)	
long*n* **isunordered** (double*n x*, double*n y*)	
int **isunordered** (half *x*, half *y*)	
short*n* **isunordered** (half*n x*, half*n y*)	
int **signbit** (float)	Test for sign bit
int*n* **signbit** (float*n*)	
int **signbit** (double)	
long*n* **signbit** (double*n*)	
int **signbit** (half)	
short*n* **signbit** (half*n*)	
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 **bitselect** (*T a*, *T b*, *T c*)	Each bit of result is corresponding bit of *a* if corresponding bit of *c* is 0
half*n* **bitselect** (half*n a*, half*n b*, half*n c*)	
double*n* **bitselect** (double*n a*, double*n b*, double*n c*)	
T **select** (*T a*, *T b*, *S c*)	For each component of a vector type, result[*i*] = if MSB of *c*[*i*] is set ? *b*[*i*] : *a*[*i*] For scalar type, result = *c* ? *b* : *a*
T **select** (*T a*, *T b*, *U c*)	
double*n* **select** (double*n*, double*n*, long*n*)	
double*n* **select** (double*n*, double*n*, ulong*n*)	
half*n* **select** (half*n*, half*n*, short*n*)	
half*n* **select** (half*n*, half*n*, ushort*n*)	

Vector Data Load/Store 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, uint, long, ulong, half, or float (or optionally double). ***Tn*** refers to the vector form of type *T*. **Optional extensions enable the double, double*n*, half, and half*n* types.**

Tn **vload*n*** (size_t *offset*, const *Q T* **p*)	Read vector data from memory
void **vstore*n*** (*Tn data*, size_t *offset*, *Q T* **p*)	Write vector data to memory *(Q in this function cannot be __constant)*
float **vload_half** (size_t *offset*, const *Q* half **p*)	Read a half from memory
float*n* **vload_half*n*** (size_t *offset*, const *Q* half **p*)	Read multiple halfs from memory
void **vstore_half** (float *data*, size_t *offset*, *Q* half **p*)	Write a half to memory *(Q in this function cannot be __constant)*
void **vstore_half**_*R* (float *data*, size_t *offset*, *Q* half **p*)	
void **vstore_half** (double *data*, size_t *offset*, *Q* half **p*)	
void **vstore_half**_*R* (double *data*, size_t *offset*, *Q* half **p*)	
void **vstore_half*n*** (float*n data*, size_t *offset*, *Q* half **p*)	Write a half vector to memory *(Q in this function cannot be __constant)*
void **vstore_half*n***_*R* (float*n data*, size_t *offset*, *Q* half **p*)	
void **vstore_half*n*** (double*n data*, size_t *offset*, *Q* half **p*)	
void **vstore_half*n***_*R* (double*n data*, size_t *offset*, *Q* half **p*)	
float*n* **vloada_half*n*** (size_t *offset*, const *Q* half **p*)	sizeof (float*n*) bytes of data read from location (*p* + (*offset* * *n*))
void **vstorea_half*n*** (float*n data*, size_t *offset*, *Q* half **p*)	Write a half vector to vector-aligned memory *(Q in this function cannot be __constant)*
void **vstorea_half*n***_*R* (float*n data*, size_t *offset*, *Q* half **p*)	
void **vstorea_half*n*** (double*n data*, size_t *offset*, *Q* half **p*)	
void **vstorea_half*n***_*R* (double*n data*, size_t *offset*, *Q* half **p*)	

Async Copies and Prefetch Functions [6.11.10]

T is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, ulong*n*, float, float*n*, and optionally half*n* double, double*n*. **Optional extensions enable the half*n*, double, and double*n* types.**

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

Atomic Functions [6.11.11, 9.4]

T is type int or unsigned int. ***T*** may also be type float for **atomic_xchg**, and type long or ulong for extended 64-bit atomic functions. ***Q*** is volatile __global or volatile __local, except ***Q*** must be volatile __global for **atomic_xchg** when ***T*** is float.

The built-in atomic functions for 32-bit values begin with atomic_ while the extended 64-bit atomic functions begin with atom_. For example:

Built-in atomic function	Extended atomic function
atomic_add ()	**atom_add** ()

Extended 64-bit atomic functions are enabled by the following pragma; *extension-name* is one of cl_khr_int64_{base, extended}_atomics:

```
#pragma OPENCL EXTENSION extension-name : enable
```

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

Miscellaneous Vector Built-In Functions [6.11.12]

Tn and *Tm* mean the 2,4,8, or 16-component vectors of char, uchar, short, ushort, half, int, uint, long, ulong, float, double. *Un* means the built-in unsigned integer data types. For vec_step(), *Tn* also includes char3, uchar3, short3, ushort3, half3, int3, uint3, long3, ulong3, float3, and double3. Half and double types are enabled by **cl_khr_fp16** and **cl_khr_fp64** respectively.

int **vec_step** (*Tn a*) int **vec_step** (*typename*)	Takes a built-in scalar or vector data type argument and returns an integer value representing the number of elements in the scalar or vector.	*Tn* **shuffle** (*Tm x*, *Un mask*) *Tn* **shuffle2** (*Tm x*, *Tm y*, *Un mask*)	Construct permutation of elements from one or two input vectors, return a vector with same element type as input & length that is the same as the shuffle mask.

Synchronization, Explicit Mem. Fence [6.11.9-10]

flags argument is the memory address space, 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

OpenCL Graphics: Following is a subset of the OpenCL API specification that pertains to graphics.

Image Read and Write Built-in Functions [6.11.13, 9.5, 9.6.8]

The built-in functions defined in this section can only be used with image memory objects created with **clCreateImage2D** or **clCreateImage3D**. *sampler* specifies the addressing and filtering mode to use. **H** = To enable **read_imageh** and **write_imageh**, enable extension cl_khr_fp16. **3D** = To enable type image3d_t in **write_image{f, i, ui}**, enable extension cl_khr_3d_image_writes.

float4 **read_imagef** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) float4 **read_imagef** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) int4 **read_imagei** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) int4 **read_imagei** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) uint4 **read_imageui** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) uint4 **read_imageui** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) half4 **read_imageh** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) H half4 **read_imageh** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) H	Read an element from a 2D image
void **write_imagef** (image2d_t *image*, int2 *coord*, float4 *color*) void **write_imagei** (image2d_t *image*, int2 *coord*, int4 *color*) void **write_imageui** (image2d_t *image*, int2 *coord*, uint4 *color*) void **write_imageh** (image2d_t *image*, int2 *coord*, half4 *color*) H	Write *color* value to (*x*, *y*) location specified by *coord* in the 2D image
float4 **read_imagef** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) float4 **read_imagef** (image3d_t *image*, sampler_t *sampler*, float4 *coord*) int4 **read_imagei** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) int4 **read_imagei** (image3d_t *image*, sampler_t *sampler*, float4 *coord*)	Read an element from a 3D image

uint4 **read_imageui** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) uint4 **read_imageui** (image3d_t *image*, sampler_t *sampler*, float4 *coord*)	Read an element from a 3D image
int **get_image_width** (image2d_t *image*) int **get_image_width** (image3d_t *image*)	Image width in pixels
int **get_image_height** (image2d_t *image*) int **get_image_height** (image3d_t *image*)	Image height in pixels
int **get_image_depth** (image3d_t *image*)	Image depth in pixels
int **get_image_channel_data_type** (image2d_t *image*) int **get_image_channel_data_type** (image3d_t *image*)	Image channel data type
int **get_image_channel_order** (image2d_t *image*) int **get_image_channel_order** (image3d_t *image*)	Image channel order
int2 **get_image_dim** (image2d_t *image*)	Image width, height
int4 **get_image_dim** (image3d_t *image*)	Image width, height, and depth
Use this pragma to enable type image3d_t in **write_image{f, i, ui}**: #pragma OPENCL EXTENSION cl_khr_3d_image_writes : enable void **write_imagef** (image3d_t *image*, int4 *coord*, float4 *color*) 3D void **write_imagei** (image3d_t *image*, int4 *coord*, int4 *color*) 3D void **write_imageui** (image3d_t *image*, int4 *coord*, uint4 *color*) 3D	Writes *color* at *coord* in the 3D image

Image Objects

Create Image Objects [5.3.1]

cl_mem **clCreateImage2D** (cl_context *context*,
cl_mem_flags *flags*, const cl_image_format **image_format*,
size_t *image_width*, size_t *image_height*,
size_t *image_row_pitch*, void **host_ptr*, cl_int **errcode_ret*)

flags: (also for **clCreateImage3D**, **clGetSupportedImageFormats**)
CL_MEM_READ_WRITE, CL_MEM_{WRITE, READ}_ONLY,
CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

cl_mem **clCreateImage3D** (cl_context *context*,
cl_mem_flags *flags*, const cl_image_format **image_format*,
size_t *image_width*, size_t *image_height*, size_t *image_depth*, size_t *image_row_pitch*, size_t *image_slice_pitch*,
void **host_ptr*, cl_int **errcode_ret*)

flags: See **clCreateImage2D**

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 **clCreateImage2D**

Copy Between Image, Buffer Objects [5.3.4]

cl_int **clEnqueueCopyImageToBuffer** (
cl_command_queue *command_queue*, cl_mem *src_image*,
cl_mem *dst_buffer*, const size_t *src_origin*[3],
const size_t *region*[3], 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*[3], const size_t *region*[3],
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

Map and Unmap Image Objects [5.3.5]

void * **clEnqueueMapImage** (
cl_command_queue *command_queue*, cl_mem *image*,
cl_bool *blocking_map*, cl_map_flags *map_flags*,
const size_t *origin*[3], const size_t *region*[3],
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*)

Read, Write, Copy Image Objects [5.3.3]

cl_int **clEnqueueReadImage** (
cl_command_queue *command_queue*, cl_mem *image*,
cl_bool *blocking_read*, const size_t *origin*[3],
const size_t *region*[3], 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*[3], const size_t *region*[3],
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 **clEnqueueCopyImage** (
cl_command_queue *command_queue*,
cl_mem *src_image*, cl_mem *dst_image*,
const size_t *src_origin*[3], const size_t *dst_origin*[3],
const size_t *region*[3], cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

Query Image Objects [5.3.6]

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

param_name: CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR},
CL_MEM_{MAP, REFERENCE}_COUNT,
CL_MEM_{CONTEXT, OFFSET},
CL_MEM_ASSOCIATED_MEMOBJECT

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

param_name: CL_IMAGE_{FORMAT, ELEMENT_SIZE},
CL_IMAGE_{ROW, SLICE}_PITCH,
CL_IMAGE_{HEIGHT, WIDTH, DEPTH},
CL_IMAGE_D3D10_SUBRESOURCE_KHR,
CL_MEM_D3D10_RESOURCE_KHR

Access Qualifiers [6.6]

Apply to image image2d_t and image3d_t types to declare if the image memory object is being read or written by a kernel. The default qualifier is __read_only.

__read_only, read_only
__write_only, write_only

Image Formats [5.3.1.1, 9.5]

Supported image formats: image_channel_order with image_channel_data_type.

Built-in support: [Table 5.7]

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

Optional support: [Table 5.5]

CL_R, CL_A: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16,32} , CL_UNSIGNED_INT{8,16,32} , CL_SNORM_INT{8,16}
CL_INTENSITY: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SNORM_INT{8|16}
CL_LUMINANCE: CL_UNORM_INT{8,16}, CL_HALF_FLOAT, CL_FLOAT, CL_SNORM_INT{8,16}
CL_RG, CL_RA: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16, 32} , CL_UNSIGNED_INT{8,16,32} , CL_SNORM_INT{8,16}
CL_RGB: CL_UNORM_SHORT_{555,565} , CL_UNORM_INT_101010
CL_ARGB: CL_UNORM_INT8, CL_SIGNED_INT8, CL_UNSIGNED_INT8, CL_SNORM_INT8
CL_BGRA: CL_SIGNED_INT8, CL_UNSIGNED_INT8, CL_SNORM_INT8

Sampler Objects [5.5]

cl_sampler **clCreateSampler** (
cl_context *context*, cl_bool *normalized_coords*,
cl_addressing_mode *addressing_mode*,
cl_filter_mode *filter_mode*, cl_int **errcode_ret*)

cl_int **clRetainSampler** (cl_sampler *sampler*)

cl_int **clReleaseSampler** (cl_sampler *sampler*)

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

param_name: CL_SAMPLER_REFERENCE_COUNT,
CL_SAMPLER_{CONTEXT, FILTER_MODE},
CL_SAMPLER_ADDRESSING_MODE,
CL_SAMPLER_NORMALIZED_COORDS

Sampler Declaration Fields [6.11.13.1]

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

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

normalized-mode:
    CLK_NORMALIZED_COORDS_{TRUE, FALSE}

address-mode:
    CLK_ADDRESS_{REPEAT, CLAMP, NONE},
    CLK_ADDRESS_{CLAMP_TO_EDGE, MIRRORED_REPEAT}

filter-mode:
    CLK_FILTER_NEAREST, CLK_FILTER_LINEAR

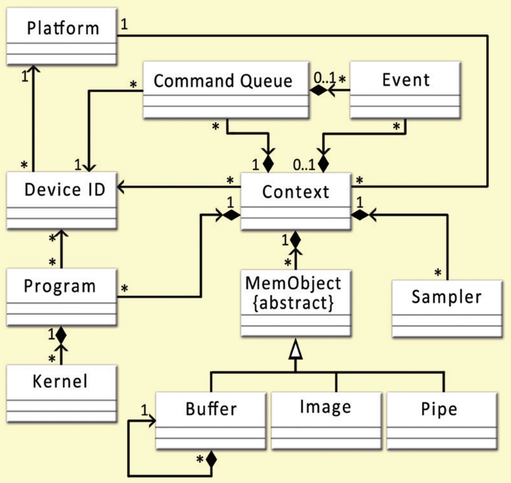
OpenCL Class Diagram [5.13]

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

Annotations

Relationships	
abstract classes	{abstract}
aggregations	◆
inheritance	△
relationship navigability	^

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



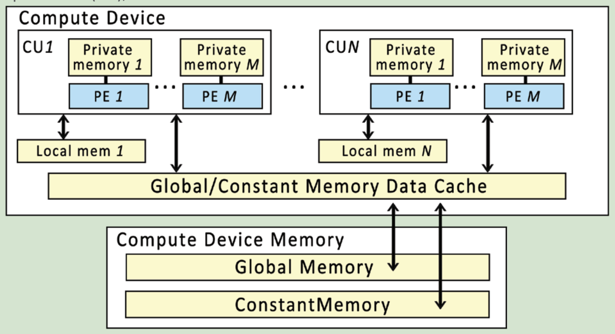
[1] Unified Modeling Language (http://www.uml.org/) is a trademark of Object Management Group (OMG).

OpenCL Device Architecture Diagram [3.3]

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

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

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



OpenCL/OpenGL Sharing APIs

Creating OpenCL memory objects from OpenGL objects using **clCreateFromGLBuffer**, **clCreateFromGLTexture2D**, **clCreateFromGLTexture3D**, and **clCreateFromGLRenderbuffer** ensure that the storage of the OpenGL object will not be deleted while the corresponding OpenCL memory object exists.

CL Buffer Objects > GL Buffer Objects [9.8.2]

cl_mem **clCreateFromGLBuffer** (cl_context *context*,
    cl_mem_flags *flags*, GLuint *bufobj*, int **errcode_ret*)

flags: CL_MEM_{READ, WRITE}_ONLY, CL_MEM_READ_WRITE

CL Image Objects > GL Textures [9.8.3]

cl_mem **clCreateFromGLTexture2D** (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_{2D, RECTANGLE},
    GL_TEXTURE_CUBE_MAP_POSITIVE_{X, Y, Z},
    GL_TEXTURE_CUBE_MAP_NEGATIVE_{X, Y, Z}

cl_mem **clCreateFromGLTexture3D** (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_3D

CL Image Objects > GL Renderbuffers [9.8.4]

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

flags: **clCreateFromGLBuffer**

Query Information [9.8.5]

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

**gl_object_type* returns: CL_GL_OBJECT_BUFFER,
    CL_GL_OBJECT_{TEXTURE2D, TEXTURE3D},
    CL_GL_OBJECT_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,
    CL_GL_MIPMAP_LEVEL

Share Objects [9.8.6]

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

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

CL Event Objects > GL Sync Objects [9.9]

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

CL Context > GL Context, Sharegroup [9.7]

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

OpenCL/Direct3D 10 Sharing APIs [9.10]

Creating OpenCL memory objects from OpenGL objects using **clCreateFromGLBuffer**, **clCreateFromGLTexture2D**, **clCreateFromGLTexture3D**, or **clCreateFromGLRenderbuffer** ensures that the storage of that OpenGL object will not be deleted while the corresponding OpenCL memory object exists.

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_KHR,
    CL_D3D10_DXGI_ADAPTER_KHR

d3d_object: ID3D10Device, IDXGIAdapter

d3d_device_set: CL_ALL_DEVICES_FOR_D3D10_KHR,
    CL_PREFERRED_DEVICES_FOR_D3D10_KHR

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

flags: CL_MEM_{READ, WRITE}_ONLY, CL_MEM_READ_WRITE

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

flags: See **clCreateFromD3D10BufferKHR**

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

flags: See **clCreateFromD3D10BufferKHR**

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

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

Notes

Notes