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. In blue type: When the appropriate extension is supported.

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

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

**Partitioning a Device** [4.3]

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

**Contexts** [4.4]

- `cl_int clCreateContext (const cl_context_properties *properties, cl_uint num_devices, cl_device_id *device, void *private_info)`

**Get CL Extension Function Pointers** [9.2]

- `void *eglGetExtensionFunctionAddressForPlatform (cl_platform_id platform, const char *funcname)`

---

Buffer Objects

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

**Create Buffer Objects** [5.2.1]

- `cl_mem clCreateBuffer(cl_mem_flags flags, size_t size, void *host_ptr, cl_int *errcode_ret` flags:
  - [Table 5.3] CL_MEM_READ_WRITE
  - CL_MEM_WRITE_ONLY
  - CL_MEM_USE_HOST_PTR
  - CL_MEM_MAP_HOST_PTR

- `void *buffer_create_info, cl_int *errcode_ret`

**Write, Read, Copy Buffer Objects** [5.2.2]

- `void clEnqueueReadBuffer(cl_isa_queue queue, void *host_ptr, size_t offset, size_t size, void *ptr, cl_uint num_events_in_wait_list, cl_event *event_wait_list)`

- `void clEnqueueReadBufferRect(cl_isa_queue queue, cl_mem src_buffer, size_t row_size, size_t slice_size, void *host_ptr, size_t offset, size_t size, void *ptr, cl_uint num_events_in_wait_list, cl_event *event_wait_list)`

(Continued on next page >)
OpenCL Class Diagram

The figure below describes the OpenCL specification as a class diagram using the Unified Modeling Language (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
aggregations
inheritance
relationship
Cardinality

many
one and only one
optionally one
one or more

Modeling Language
The figure below describes the OpenCL specification as a class diagram using the Unified Modeling Language (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.

```

OpenCL 2.0 Reference Card

Conversions and Type Casting Examples

\[ T = (T_1) \]  
\[ T = (T_2) \]

T \( \rightarrow \) T:

- Scalar to scalar,
- Vector to scalar

T \( \rightarrow \) T:

- Scalar to scalar
- Vector to scalar

T \( \rightarrow \) T:

- Scalar to scalar
- Vector to scalar

T \( \rightarrow \) T:

- Scalar to scalar
- Vector to scalar

OpenCL Device Architecture Diagram

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

<table>
<thead>
<tr>
<th>Region</th>
<th>Allocation</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>No allocation</td>
<td>R/W access</td>
</tr>
<tr>
<td>Global</td>
<td>Static allocation</td>
<td>R/W access</td>
</tr>
<tr>
<td>Local</td>
<td>Static allocation</td>
<td>R/W access</td>
</tr>
</tbody>
</table>

Memory Objects

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

```
Memory Objects [5.5.1, 5.5.2]

- cl_int clRetainMemObject (cl_mem memobj)
- cl_int clReleaseMemObject (cl_mem memobj)
- cl_int clSetMemObjectDestructorCallback (cl_mem memobj, void *callback_func, void *user_data)

Migrate Memory Objects

- cl_int clEnqueueMigrateMemObject (cl_context context, cl_mem memobj, cl_image_mem_object_t image, cl_int flags, cl_mem_object_flags_t memobjflags, cl_int num_events_in_wait_list, const cl_event *event_wait_list)

Query Memory Object

- cl_int clGetMemObjectInfo (cl_mem memobj, cl_mem_info info, param name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)

Shared Virtual Memory

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

```
SVM Sharing Granularity

- void *dsvmalloc (cl_context context, cl_mem_flags flags, size_t size, unsigned int alignment)

Enqueuing SVM Operations

- cl_int clEnqueueSVMFree (cl_command_queue command_queue, cl_int num_svm_pointers, cl_int svm_free_func)

Pipe Object Queries

- cl_int clGetPipeInfo (cl_mem pipe, cl_pipe_info info, size_t param_value_size, void *param_value, size_t *param_value_size_ret)

Create Pipe Objects

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

Pipes

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

Copyright © Khronos Group - Rev. 1119

www.khronos.org/opencl
Program Objects

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

Create Program Objects [5.8.3]

cl_program clCreateProgramWithSource (cl_context context, cl_uint count, const char *strings, cl_int *errorcode_ret)

cl_program clCreateProgramWithBinary (cl_context context, cl_uint num_devices, const cl_device_id *device_list, size_t *lengths, cl_int *binary_status, cl_int *errorcode_ret)

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

cl_int clRetainProgram (cl_program program)

cl_int clReleaseProgram (cl_program program)

Building Program Executables [5.8.2]

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

Separate Compilation and Linking [5.8.3]

cl_int clCompileProgram (cl_program program, cl_uint num_devices, const cl_device_id *device_list, cl_int *input_value_size, cl_int *output_value_size, cl_int *input_value_size_ret, cl_int *output_value_size_ret)

cl_int clGetKernelInfo (cl_kernel kernel, const char *name, size_t *value, size_t *value_size, size_t *value_size_ret)

cl_int clGetKernelSubGroupInfoKHR (cl_kernel kernel, cl_device_id device_id, cl_kernel_sub_group_info *kernel_name, size_t *input_value_size, size_t *param_value_size, void *input_value, void *param_value, size_t *param_value_size)

Execute Kernels [5.10]

cl_int clEnqueueNDRangeKernel (cl_command_queue command_queue, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event, const void **ait_list, size_t global_work_offset, size_t global_work_size, size_t local_work_size, size_t *global_work_size, size_t *local_work_size, size_t *global_work_offset)

cl_int clEnqueueNDRangeKernel (cl_queue command_queue, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueNativeKernel (cl_command_queue command_queue, void *CL_CALLBACK pfn_notify, cl_kernel kernel, cl_uint num_kernels, size_t *kernels_num, cl_uint *kernels, cl_int *errorcode_ret)

cl_int clFlush (cl_command_queue command_queue)

cl_int clFinish (cl_command_queue command_queue)

Flush and Finish [5.15]

cl_int clSetUserStatus (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 info_type, size_t *value, size_t *value_size, void *user_data)

cl_int clCreateUserEvent (cl_context context, cl_int *errorcode_ret)

Event Objects

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

Event Objects [5.11]

cl_int clCreateUserEvent (cl_context context, cl_int *errorcode_ret)
OpenCL 2.0 Reference Card

Markers, Barriers, Waiting for Events [5.12]

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

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

Profiling Operations [5.14]

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

param_name: [Table 5.23]

CL_PROFILING_COMMAND_QUEUED, CL_PROFILING_COMMAND_COMPLETE,
CL_PROFILING_COMMAND_SUBMIT, START, END)

OpenCL C Language Reference

Supported Data Types

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

Built-in Scalar Data Types [6.1.1]

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td></td>
<td>true (1) or false (0)</td>
</tr>
<tr>
<td>char</td>
<td></td>
<td>8-bit signed</td>
</tr>
<tr>
<td>uchar</td>
<td></td>
<td>8-bit unsigned</td>
</tr>
<tr>
<td>short</td>
<td></td>
<td>16-bit signed</td>
</tr>
<tr>
<td>ushort</td>
<td></td>
<td>16-bit unsigned</td>
</tr>
<tr>
<td>int</td>
<td></td>
<td>32-bit signed</td>
</tr>
<tr>
<td>uint</td>
<td></td>
<td>32-bit unsigned</td>
</tr>
<tr>
<td>long</td>
<td></td>
<td>64-bit signed</td>
</tr>
<tr>
<td>ulong</td>
<td></td>
<td>64-bit unsigned</td>
</tr>
<tr>
<td>float</td>
<td></td>
<td>32-bit float</td>
</tr>
<tr>
<td>double</td>
<td></td>
<td>64-bit float</td>
</tr>
<tr>
<td>half</td>
<td></td>
<td>32-bit IEEE 754</td>
</tr>
<tr>
<td>size_t</td>
<td></td>
<td>port size</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td>integer version number, e.g: 200</td>
</tr>
</tbody>
</table>

Reserved Data Types [6.1.4]

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>function type. In this example:</td>
</tr>
<tr>
<td>void *</td>
<td>Index into block's environment.</td>
</tr>
<tr>
<td>event_t</td>
<td>Function handle.</td>
</tr>
<tr>
<td>pfn_event_notify</td>
<td>Event handle.</td>
</tr>
<tr>
<td>cl_command_queue</td>
<td>Command queue handle.</td>
</tr>
</tbody>
</table>

Other Built-in Data Types [6.1.3]

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

Built-in Vector Data Types [6.1.2]

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>cl_char</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>uchar</td>
<td>cl_uchar</td>
<td>8-bit unsigned</td>
</tr>
<tr>
<td>short</td>
<td>cl_short</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>ushort</td>
<td>cl_ushort</td>
<td>16-bit unsigned</td>
</tr>
<tr>
<td>int</td>
<td>cl_int</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>uint</td>
<td>cl_uint</td>
<td>32-bit unsigned</td>
</tr>
<tr>
<td>long</td>
<td>cl_long</td>
<td>64-bit signed</td>
</tr>
<tr>
<td>ulong</td>
<td>cl_ulong</td>
<td>64-bit unsigned</td>
</tr>
<tr>
<td>float</td>
<td>cl_float</td>
<td>32-bit float</td>
</tr>
<tr>
<td>double</td>
<td>cl_double</td>
<td>64-bit float</td>
</tr>
<tr>
<td>half</td>
<td>cl_half</td>
<td>32-bit float</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td>integer version number, e.g: 200</td>
</tr>
</tbody>
</table>

Reserved Data Types [6.1.4]

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>function type. In this example:</td>
</tr>
<tr>
<td>void *</td>
<td>Index into block's environment.</td>
</tr>
<tr>
<td>event_t</td>
<td>Function handle.</td>
</tr>
<tr>
<td>pfn_event_notify</td>
<td>Event handle.</td>
</tr>
<tr>
<td>cl_command_queue</td>
<td>Command queue handle.</td>
</tr>
</tbody>
</table>

Vector Components Addressing [6.1.7]

Vector Components

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>float2</td>
<td>v.x, v.x</td>
<td>v.y, v.y</td>
<td>v.z, v.z</td>
<td>v.w, v.w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float3</td>
<td>v.x, v.x, v.x</td>
<td>v.y, v.y, v.y</td>
<td>v.z, v.z, v.z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float4</td>
<td>v.x, v.x, v.x, v.x</td>
<td>v.y, v.y, v.y, v.y</td>
<td>v.z, v.z, v.z, v.z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reserved Data Types [6.1.4]

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>function type. In this example:</td>
</tr>
<tr>
<td>void *</td>
<td>Index into block's environment.</td>
</tr>
<tr>
<td>event_t</td>
<td>Function handle.</td>
</tr>
<tr>
<td>pfn_event_notify</td>
<td>Event handle.</td>
</tr>
<tr>
<td>cl_command_queue</td>
<td>Command queue handle.</td>
</tr>
</tbody>
</table>

Operators and Qualifiers [6.5]

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

+ - * /  % ~ & ^ | << >> |

Address Space Qualifiers [6.5]

| global | local | constant | private |

Function Qualifiers [6.7]

| kernel | _attribute_((vec_type_hint(type))) |

Blocks [6.12]

A result value type with a list of parameter types, similar to a function type. In this example:

1. The * declares variable “myBlock” is a Block.
2. The return type for the Block “myBlock” is int.
3. myBlock takes a single argument of type int.
4. The argument is named “num.”
5. myBlock captured from block’s environment.

OpenCL Type       | Description       |
|-------------------|-------------------|
### Math Built-in Functions [6.13.2] [9.4.2]

- **acos (T)**: Arc cosine
- **acosh (T)**: Inverse hyperbolic cosine
- **acospi (T x)**: acos (x) / x
- **asin (T)**: Arc sine
- **asin (π)**: Inverse hyperbolic sine
- **asinpi (T x)**: asin (x) / x
- **atan (T y, x)**: Arc tangent
- **atan2 (T y, x)**: Arc tangent of y / x
- **atanh (T x)**: Hyperbolic arc tangent
- **atanpi (T x)**: atan (x) / x
- **atan2pi (T y, x)**: atan2 (y, x) / x
- **cbrt (T x)**: Cube root
- **ceil (T x)**: Round to integer toward +infinity
- **copyright (T x, T y)**: x with sign changed to sign of y
- **cos (T x)**: Cosine
- **cosh (T x)**: Hyperbolic cosine
- **cospi (T x)**: cos (x) / x
- **half_divide (T x, T y)**: x / y (if may only be float or float)
- **native_divide (T x, T y)**: (T x) / (T y)
- **erf (T x)**: Complex error function of T
- **erf (T x)**: Calculates error function of T
- **exp (T x)**: Exponential function e^x
- **exp2 (T x)**: Exponential function 2^x
- **exp10 (T x)**: Exponential function 10^x
- **expm1 (T x)**: Exponential function e^x - 1
- **fabs (T x)**: Absolute value
- **fdim (T x, T y)**: Positive difference between x and y
- **floor (T x)**: Round to integer toward infinity
- **fma (T a, T b, T c)**: Multiply and add, then round
- **fmax (T x, T y)**: Return if x < y, otherwise it returns x
- **fmin (T x, T y)**: Return if y < x, otherwise it returns x
- **fmod (T x, T y)**: Modulus. Returns x - y * trunc (x/y)
- **fract (T x, T y)**: Fractional value in x
- **frexp (T x, int */exp)**: Extract mantissa and exponent
- **frac (T x, T y)**: Round to integer value
- **intn (T llog (T x)**: Log gamma function
- **hypot (T x, T y)**: Square root of x^2 + y^2
- **ilogb (T x)**: Return exponent as an integer value
- **ldexp (T x, int n)**: x * 2^n
- **lgamma (T x)**: Natural logarithm
- **lgamma (T x, T y)**: Base 2 logarithm
- **log10 (T x)**: Base 10 logarithm
- **log1p (T x)**: In (1 + x)
- **log (T x)**: Exponent of x
- **logb (T x)**: Logarithm of (Continued on next page >)
- **mad (T x, T y, T c)**: Approximates a + b + c
- **max (T x, T y)**: Maximum magnitude of x and y
- **min (T x, T y)**: Minimum magnitude of x and y
- **mod (T x, T y)**: Decompose floating-point number
- **round (T x)**: Round to integer toward zero
- **sign (T x)**: Round to integer toward infinity
- **signp (T x)**: Round to integer toward zero
- **sin (T x)**: Hyperbolic sine
- **sincos (T x, T *cosval)**: Sine and cosine of x
- **sqrt (T x)**: Square root
- **tan (T x)**: Tangent
- **tanh (T x)**: Hyperbolic tangent
- **tanpi (T x)**: tan (x * π)
- **tgamma (T x)**: Gamma function
- **trunc (T x)**: Truncate to integer towards zero

### Math Constants [6.13.2] [9.4.2]

- **MAXFLOAT**: Value of maximum non-infinite single-precision floating-point number
- **HUGE_VALF**: Positive float evaluation, evaluates to +infinity
- **HUGE_VAL**: Positive double evaluation, evaluates to +infinity
- **INFINITY**: Constant float evaluation, positive or unsigned infinity
- **NAN**: Constant float evaluation, quiet NaN

When double precision is supported, macros ending in F are available in type double by replacing _F from the macro name, and in type half when the cl_khr_fp16 extension is enabled by replacing _F with _H.

### Integer Built-in Functions [6.13.3]

- **abs (T x)**: |x|
- **abs (T x, T y, T z)**: |x| − |y| without modulo overflow
- **add (T x, T y, T z)**: x + y and saturates the result
- **add saturated (T x, T y, T z)**: x + y and saturates the result
- ** clamp (T x, T y, T z)**: min(max(x, z), max(y, z))
- **clamp (T x, T min, T max)**: min(max(x, min), max(y, max))
- **ceil (T x)**: number of leading 0-bits in x
- **ceil (T x)**: number of trailing 0-bits in x
- **char (T x)**: number of trailing 0-bits in x
- **mul (T x, T y, T z)**: a + b + c and saturates the result
- **max (T x, T y)**: if x < y, otherwise it returns x
- **max (T x, T y)**: if y < x, otherwise it returns x
- **mul (T x, T y)**: high half of the product of x and y
- **rotate (T x, T y)**: result = (result | result shifted b bits) << a
- **sub (T x, T y)**: x − y and saturates the result
- **sub saturated (T x, T y)**: x − y and saturates the result
- **popcount (T x)**: Number of non-zero bits in x

For unsigned, return type is scalar when the parameters are scalar.

### Common Built-in Functions [6.13.4] [9.4.3]

- **max (T x, T y)**: Max of x and y
- **min (T x, T y)**: Min of x and y

(Continued on next page >)
**Common Functions (continued)**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>radd()</td>
<td>Radial addition</td>
<td>radians, operand</td>
<td>scalar</td>
</tr>
<tr>
<td>rsub()</td>
<td>Radial subtraction</td>
<td>radians, operand</td>
<td>vector</td>
</tr>
<tr>
<td>rdiv()</td>
<td>Radial division</td>
<td>radians, operand</td>
<td>scalar</td>
</tr>
<tr>
<td>rneg()</td>
<td>Radial negation</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>rabs()</td>
<td>Radial absolute value</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>rmax()</td>
<td>Radial maximum</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>rmin()</td>
<td>Radial minimum</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>rsum()</td>
<td>Radial sum</td>
<td>operand</td>
<td>scalar</td>
</tr>
<tr>
<td>rdot()</td>
<td>Radial dot product</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>rnorm()</td>
<td>Radial norm</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>rproj()</td>
<td>Radial projection</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>rdist()</td>
<td>Radial distance</td>
<td>operand</td>
<td>scalar</td>
</tr>
<tr>
<td>rangle()</td>
<td>Radial angle</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>rnormalize()</td>
<td>Radial normalize</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>rstep()</td>
<td>Radial step</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>rsmoothstep()</td>
<td>Radial smoothstep</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>rsign()</td>
<td>Radial sign</td>
<td>operand</td>
<td>vector</td>
</tr>
</tbody>
</table>

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

These functions can be used with built-in scalar or vector types as arguments and return a scalar or vector integer result.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>iseq()</td>
<td>Integer equality</td>
<td>operand</td>
<td>scalar</td>
</tr>
<tr>
<td>igreater()</td>
<td>Integer greater</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>ile()</td>
<td>Integer less</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>islesseq()</td>
<td>Integer less or equal</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>isnot()</td>
<td>Integer not</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>isis()</td>
<td>Integer is</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>isnot()</td>
<td>Integer not</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>islesseq()</td>
<td>Integer not less or equal</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>islesseq()</td>
<td>Integer is less or equal</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>islesseq()</td>
<td>Integer is not less or equal</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>islesseq()</td>
<td>Integer is not equal</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>islesseq()</td>
<td>Integer is not less or equal</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>islesseq()</td>
<td>Integer is not equal</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>islesseq()</td>
<td>Integer is not equal</td>
<td>operand</td>
<td>vector</td>
</tr>
</tbody>
</table>

**Vector Data Load/Store [6.13.7] [9.4.6]**

These functions allow reading and writing of data from memory locations.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>vload()</td>
<td>Data load</td>
<td>offset, data</td>
<td>vector</td>
</tr>
<tr>
<td>vstore()</td>
<td>Data store</td>
<td>offset, data</td>
<td>vector</td>
</tr>
</tbody>
</table>

**Synchronization & Memory Fence Functions [6.13.8]**

These functions allow atomic access to shared memory locations.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>work_group_barrier()</td>
<td>Work item barrier</td>
<td>fence, workgroup</td>
<td>void</td>
</tr>
<tr>
<td>atomic_work_item_fence()</td>
<td>Atomic work item barrier</td>
<td>fence, workgroup</td>
<td>void</td>
</tr>
<tr>
<td>sub_group_barrier()</td>
<td>Sub group barrier</td>
<td>fence, workgroup</td>
<td>void</td>
</tr>
</tbody>
</table>

**Asynchronous Copies and Prefetch [6.13.10] [9.4.7]**

These functions allow asynchronous copies and prefetching of data.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>vcopy()</td>
<td>Data copy</td>
<td>offset, data</td>
<td>vector</td>
</tr>
</tbody>
</table>

**Geometric Built-in Functions [6.13.5] [9.4.4]**

These functions are used with built-in vector types to perform geometric operations.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>fadd()</td>
<td>Float add</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>fsub()</td>
<td>Float subtract</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>fmul()</td>
<td>Float multiply</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>fdiv()</td>
<td>Float divide</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>fmod()</td>
<td>Float mod</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>fmax()</td>
<td>Float maximum</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>fmin()</td>
<td>Float minimum</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>fsum()</td>
<td>Float sum</td>
<td>operand</td>
<td>scalar</td>
</tr>
<tr>
<td>fdot()</td>
<td>Float dot</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>fnorm()</td>
<td>Float norm</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>fproj()</td>
<td>Float projection</td>
<td>operand</td>
<td>vector</td>
</tr>
<tr>
<td>fdist()</td>
<td>Float distance</td>
<td>operand</td>
<td>scalar</td>
</tr>
<tr>
<td>fangle()</td>
<td>Float angle</td>
<td>operand</td>
<td>vector</td>
</tr>
</tbody>
</table>

**Async Copies and Prefetch [6.13.10] [9.4.7]**

These functions allow asynchronous copies and prefetching of data.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>vcopy()</td>
<td>Data copy</td>
<td>offset, data</td>
<td>vector</td>
</tr>
</tbody>
</table>

**Syncopation & Memory Fence Functions [6.13.8]**

These functions allow atomic access to shared memory locations.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>work_group_barrier()</td>
<td>Work item barrier</td>
<td>fence, workgroup</td>
<td>void</td>
</tr>
<tr>
<td>atomic_work_item_fence()</td>
<td>Atomic work item barrier</td>
<td>fence, workgroup</td>
<td>void</td>
</tr>
<tr>
<td>sub_group_barrier()</td>
<td>Sub group barrier</td>
<td>fence, workgroup</td>
<td>void</td>
</tr>
</tbody>
</table>

**Asynchronous Copies and Prefetch [6.13.10] [9.4.7]**

These functions allow asynchronous copies and prefetching of data.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>vcopy()</td>
<td>Data copy</td>
<td>offset, data</td>
<td>vector</td>
</tr>
</tbody>
</table>

**Syncopation & Memory Fence Functions [6.13.8]**

These functions allow atomic access to shared memory locations.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>work_group_barrier()</td>
<td>Work item barrier</td>
<td>fence, workgroup</td>
<td>void</td>
</tr>
<tr>
<td>atomic_work_item_fence()</td>
<td>Atomic work item barrier</td>
<td>fence, workgroup</td>
<td>void</td>
</tr>
<tr>
<td>sub_group_barrier()</td>
<td>Sub group barrier</td>
<td>fence, workgroup</td>
<td>void</td>
</tr>
</tbody>
</table>
**Atomic Functions**

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

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory_order</td>
<td>memory_order_acquire</td>
<td>Enum which identifies memory ordering constraints.</td>
</tr>
<tr>
<td>memory_scope</td>
<td>memory_scope_work_item</td>
<td>Enum which identifies scope of memory ordering constraints. memory_scope_sub_group requires the cl_khr_subgroups extension.</td>
</tr>
</tbody>
</table>

**Atomic integer and floating-point types**

† indicates types supported by a limited subset of atomic operations.
‡ indicates size depends on whether implemented on 64-bit or 32-bit architecture.
§ indicates types supported only if both 64-bit extensions are supported.

**Atomic Macros**

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

**64-bit Atomics**

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

The cl_khr_int64_extended_atomics extension enables 64-bit versions of the following functions: `atom_min`, `atom_max`, `atom_and`, `atom_or`, `atom_xor`
Pipe Built-in Functions [6.13.16-2.4]

T represents the built-in OpenCL C scalar or vector integer or floating-point data types or any user defined type built from these scalar and vector data types. Half scalar and vector types require the cl_khr_subgroups extension. The macro CLK_NULL_EVENT refers to an invalid event ID.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int read_pipe (pipe T p, T *ptr)</td>
<td>Read packet from p into ptr.</td>
</tr>
<tr>
<td>int read_pipe (pipe T p, reserve_id_t reserve_id, uint index, T *ptr)</td>
<td>Read packet from reserved area of the pipe reserved_id and index into ptr.</td>
</tr>
<tr>
<td>void * write_pipe (pipe T p, T val)</td>
<td>Write packet specified by ptr to p.</td>
</tr>
<tr>
<td>void * write_pipe (ptr to ptr to reserved area reserved_id and index.</td>
<td></td>
</tr>
<tr>
<td>bool isValid_reserved_id (reserve_id_t reserve_id)</td>
<td>Return true if reserved_id is a valid reservation ID and false otherwise.</td>
</tr>
<tr>
<td>void * get_pipe_max_packets (pipe T p)</td>
<td>Returns maximum number of packets specified when p was created.</td>
</tr>
<tr>
<td>void * get_pipe_num_packets (pipe T p, uint index)</td>
<td>Reserve num_packets entries for reading from or writing to p.</td>
</tr>
<tr>
<td>size_t get_pipe_nbytes (pipe T p, uint index)</td>
<td>Indicates that all reads and writes to num_packets associated with reservation reserved_id are completed.</td>
</tr>
<tr>
<td>void commit_read_pipe (pipe T p, reserve_id_t reserve_id)</td>
<td>Takes a built-in scalar or vector data type argument. Returns 1 for scalar, 4 for 3-component vector, else number of elements in the specified type.</td>
</tr>
<tr>
<td>T * get_kernel_sub_group_count_for_ndrange (const ndrange_t * p)</td>
<td>Used to execute a block. Work-items can enqueue execution to queue(s).</td>
</tr>
<tr>
<td>void work_group_commit_read_pipe (pipe T p, reserve_id_t reserve_id, T m x, T m y, T m z)</td>
<td>Allows a work-item to enqueue a block for execution to queue. Work-items can enqueue multiple blocks to a device queue() (this may be one of CL_ENQUEUE_NO_WAIT_KERNEL, WAIT_WORK_GROUP).</td>
</tr>
<tr>
<td>void set_user_event_status (clk_event_t event, int status)</td>
<td>Sets the execution status of a user event. Status: CL_COMPLETE or a negative error value.</td>
</tr>
<tr>
<td>void capture_event_profiling_info (clk_event_t event, info, global void *value)</td>
<td>Captures profiling information for command associated with event in value.</td>
</tr>
</tbody>
</table>

Enqueuing and Kernel Query Built-in Functions [6.13.17][9.17.3.6]

A kernel may enqueue code represented by Block syntax, and control execution order with event dependencies including user events and markers. There are several advantages to using the Block syntax: it is more compact; it does not require a cl_kernel object; and enqueuing can be done as a single semantic step. Sub-groups require the cl_khr_subgroups extension. The macro CLK_NULL_EVENT refers to an invalid device event. The macro CLK_NULL_QUEUE refers to an invalid device queue.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int enqueue_kernel (queue t queue, kernel enqueue flags T flags, const ndrange_t ndrange, void *[block][void])</td>
<td>Allows a work-item to enqueue a block for execution to queue.</td>
</tr>
<tr>
<td>int enqueue_kernel (queue t queue, kernel enqueue flags T flags, const ndrange_t ndrange, uint num_events_in_wait_list, const clk_event_t *event_wait_list, cl_event_t *event_ret, void *[block][void])</td>
<td>Enables a work-item to enqueue multiple blocks to a device queue() (this may be one of CL_ENQUEUE_NO_WAIT_KERNEL, WAIT_WORK_GROUP).</td>
</tr>
<tr>
<td>void set_kernel_work_group_size (void *[block][void])</td>
<td>Query the maximum work-group size that can be used to execute a block.</td>
</tr>
<tr>
<td>void set_kernel_preferred_work_group_size_multiple (void *[block][void])</td>
<td>Returns the preferred multiple of work-group size for launch.</td>
</tr>
<tr>
<td>void enqueue_marker (queue t queue, uint num_events_in_wait_list, cl_event_t *event_wait_list, cl_event_t *event_ret)</td>
<td>Enqueue a marker command to queue.</td>
</tr>
<tr>
<td>void enqueue_kernel_to_group (const ndrange_t ndrange, void *block)</td>
<td>Returns number of subgroups in each workgroup of the dispatch.</td>
</tr>
<tr>
<td>void enqueue_kernel_to_group (const ndrange_t ndrange, void *block)</td>
<td>Returns the maximum sub-group size for a block.</td>
</tr>
</tbody>
</table>

Event Built-in Functions [6.13.17.8]

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void retain_event (clk_event_t event)</td>
<td>Increments event reference count.</td>
</tr>
<tr>
<td>void release_event (clk_event_t event)</td>
<td>Decrements event reference count.</td>
</tr>
<tr>
<td>void is_valid_event (clk_event_t event)</td>
<td>Return true for a valid event.</td>
</tr>
<tr>
<td>void set_user_event_status (clk_event_t event, int status)</td>
<td>Sets the execution status of a user event. Status: CL_COMPLETE or a negative error value.</td>
</tr>
<tr>
<td>void capture_event_profiling_info (clk_event_t event, info, global void *value)</td>
<td>Captures profiling information for command associated with event in value.</td>
</tr>
</tbody>
</table>

Helper Built-in Functions [6.13.17.9]

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void get_default_queue (void)</td>
<td>Default queue or CLK_NULL_QUEUE.</td>
</tr>
<tr>
<td>nrange_t get_nrange_1D (size_t global_work_size)</td>
<td>Builds a 1D ND-range descriptor.</td>
</tr>
<tr>
<td>nrange_t get_nrange_1D (size_t global_work_offset, size_t global_work_size)</td>
<td>Builds a 2D ND-range descriptor. n may be 2 or 3.</td>
</tr>
</tbody>
</table>

OpenCL Image Processing Reference

A subset of the OpenCL API and C Language specifications pertaining to image processing and graphics.
Image Formats [5.3.1.1]
Supported image formats: image_channel_order with image_channel_data_type.

Optional support: [Table 5.6]

CL_R, CL_A: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT8(8,16),
CL_SIGNED_INT8(16,32), CL_UNSIGNED_INT8(8,16),
CL_SIGNED_INT8(8,16), CL_SNORM_INT8(16)

CL_INTENSITY: CL_HALF_FLOAT, CL_FLOAT,
CL_UNORM_INT8(8,16), CL_SIGNED_INT8(16,32),
CL_UNORM_INT8(16), CL_SNORM_INT8(16)

CL_DEPTH_STENCIL: Only used if extension
cl_khr_gl_msaa_sharing is enabled and
channel data type = CL_UNORM_INT8 or CL_FLOAT

CL_LUMINANCE: CL_UNORM_INT8(8,16), CL_HALF_FLOAT,
CL_FLOAT, CL_SIGNED_INT8(16,32), CL_SNORM_INT8(16)

CL_RGBA: CL_UNORM_SHORT(555,565),
CL_UNORM_INT8(101010)

CL_RGB: CL_UNORM_INT8(555555), CL_SIGNED_INT8,
CL_SNORM_INT8

CL_BGR: CL_SIGNED, UNSIGNED_INT8, CL_SNORM_INT8

OpenCL Image Processing

Read and write functions for 2D images (continued)

half4 read_imagef(image2d_t image, int2 coord, float4 color)
void write_imagef(image2d_t image, int2 coord, half4 color)
void write_imagef(image2d_t image, int2 coord, int4 color)
void write_imagef(image2d_t image, int2 coord, float4 color)
void write_imagef(image2d_t image, int2 coord, uint4 color)
void write_imagef(image2d_t image, int2 coord, half4 color)

Read and write functions for 3D images

float4 read_imagef(image3d_t image, int3 coord, float4 color)
void write_imagef(image3d_t image, int3 coord, float4 color)
void write_imagef(image3d_t image, int3 coord, int4 color)
void write_imagef(image3d_t image, int3 coord, float4 color)
void write_imagef(image3d_t image, int3 coord, uint4 color)
void write_imagef(image3d_t image, int3 coord, half4 color)
void write_imagef(image3d_t image, int3 coord, half4 color)

Read and write functions for 4D images

half4 read_imagef(image4d_t image, int4 coord, float4 color)
void write_imagef(image4d_t image, int4 coord, float4 color)
void write_imagef(image4d_t image, int4 coord, int4 color)
void write_imagef(image4d_t image, int4 coord, float4 color)
void write_imagef(image4d_t image, int4 coord, uint4 color)
void write_imagef(image4d_t image, int4 coord, half4 color)
void write_imagef(image4d_t image, int4 coord, half4 color)

(Continued on next page >)
Image Read and Write (continued)

Extended mipmap read and write functions [9.18.2.1]

These functions require the cl_khr_mipmap_image and cl_khr_mipmap_write_extensions.

```c
int4 read_image( image2d_t image, sampler_t sampler, float2 coord, float2 lod )
int4 read_image( image2d_t image, sampler_t sampler, float4 coord, float4 lod )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, coord lod )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, uint4 lod )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, int4 lod )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, float2 gradient_x, float2 gradient_y )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, uint2 gradient_x, uint2 gradient_y )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, int2 gradient_x, int2 gradient_y )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, float4 gradient )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, uint4 gradient )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, int4 gradient )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, float4 gradient_x, float4 gradient_y )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, uint4 gradient_x, uint4 gradient_y )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, int4 gradient_x, int4 gradient_y )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, float4 gradient, float4 gradient )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, uint4 gradient, uint4 gradient )
int4 read_image( image2d_t image, sampler_t sampler, coord coord, int4 gradient, int4 gradient )
```
OpenCL Extensions Reference

Using OpenCL Extensions [9]
The following extensions extend the OpenCL API.
Extensions shown in italics provide core features.
To control an extension: #pragma OPENCL EXTENSION extension_name : [enable | disable]
To test if an extension is supported: clGetPlatformInfo() or clGetDeviceInfo()
To get the address of the extension function: clGetExtensionFunctionAddressForPlatform()

cl_apple_gl_sharing (see cl_khr_gl_sharing)
cl_khr_3d_image_writes
cl_khr_byte_addressable_store

OpenGL Sharing [8.5 - 8.7]
These functions require the cl_khr_gl_sharing or cl_apple_gl_sharing extension.

CL Context > GL Context, Sharegroup [9.5.5]
c_int clGetGLContextInfoKHR (cl_context context, void *param_value_size, size_t param_value_size, void *param_value, void *param_value_size_ret, size_t param_value_size_ret)
c_int clCreateFromGLContextKHR (cl_context context, cl_device_type type, cl_context *context)
c_int clAcquireGLContextKHR (cl_context context, cl_context *context)
c_int clReleaseGLContextKHR (cl_context context)

CL Buffer Objects > GL Buffer Objects [9.6.2]
c_mem clCreateFromGLBuffer (cl_context context, cl_buffer_flags flags, cl_uint bufobj, cl_int *errcode_ret)
cl_mem clCreateFromGLBufferKHR (cl_context context, cl_buffer_flags flags, GLuint bufobj, cl_int *errcode_ret)
flags: See clCreateFromGLBuffer

CL Image Objects > GL Textures [9.6.3]
c_mem clCreateFromGLTexture (cl_context context, cl_image_flags flags, GLenum texture_target, GLint mipmaplevel, cl_uint *param_value_size, void *param_value, void *param_value_size_ret)
flags: See clCreateFromGLBuffer

d3d9_media sharing [9.9]
These functions require the extension cl_khr_d3d9_media_sharing.
c_int clGetDeviceIDsFromD3D9KHR (cl_int num_devices, cl_device_type type, cl_device_id *devices)
c_int clCreateFromD3D9TextureKHR (cl_context context, cl_mem_flags flags, cl_int num_objects, const cl_mem *mem_objects, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *errcode_ret)

dx9媒体共享 [9.9]
These functions require the extension cl_khr_dx9_media_sharing.
c_int clCreateFromDX9MediaAdapterKHR (const cl_device_id *devices, const cl_device_type *device_types, void *media_adapters, cl_host_ptr *media_adapter_set_khr_media_adapter_set, cl_uint *num_entries, cl_int *param_value_size, void *param_value, cl_int *param_value_size_ret)
flags: See clCreateFromGLBuffer

Direct3D 10 Sharing [8.7]
These functions require the cl_khr_d3d10_sharing extension. The associated header file is cl_d3d10.h.
c_int clGetDeviceIDsFromD3D10KHR (cl_platform_id platform, cl_array_size max_memory_objects, cl_device_type *device_types, void *device_memory_objects, cl_int *errcode_ret)
c_int clCreateFromD3D10Texture2DKHR (cl_context context, cl_mem_flags flags, cl_int num_objects, const cl_mem *mem_objects, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)

direct3d 11 sharing [9.10.7.3 - 9.10.7.6]
These functions require the cl_khr_d3d11_sharing extension. Associated header file is cl_d3d11.h.
c_int clGetDeviceIDsFromD3D11KHR (cl_context context, cl_platform_id platform, cl_array_size max_objects, cl_mem_flags *object_types, void *device_objects, cl_int *param_value_size, void *param_value, cl_int *param_value_size_ret)
flags: See clCreateFromGLBuffer

egl interoperability [9.19, 9.20]

Create CL Image Objects from EGL [9.19]
These functions require the extension cl_khr_eegl_image.
c_mem clCreateFromEGLImageKHR (cl_context context, cl_mem_flags flags, cl_int *errcode_ret)
cl_mem clCreateFromEGLImageKHR (cl_context context, cl_mem_flags flags, cl_int *errcode_ret)
cl_mem clCreateFromEGLImagePropertiesKHR (cl_int *param_value_size, void *param_value)

Create CL Event Objects from EGL [9.20]
This function requires the extension cl_khr_eegl_event.
c_event clCreateFromEGLSyncKHR (cl_context context, cl_device_id device_id, cl_int *errcode_ret)

EGL Interoperability [9.19, 9.20]

Direct3D 10 Sharing [8.7]
These functions require the cl_khr_d3d10_sharing extension. The associated header file is cl_d3d10.h.
c_int clGetDeviceIDsFromD3D10KHR (cl_platform_id platform, cl_array_size max_memory_objects, cl_device_type *device_types, void *device_memory_objects, cl_int *errcode_ret)
c_int clCreateFromD3D10Texture2DKHR (cl_context context, cl_mem_flags flags, cl_int num_objects, const cl_mem *mem_objects, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)

direct3d 11 sharing [9.10.7.3 - 9.10.7.6]
These functions require the cl_khr_d3d11_sharing extension. Associated header file is cl_d3d11.h.
c_int clGetDeviceIDsFromD3D11KHR (cl_context context, cl_platform_id platform, cl_array_size max_objects, cl_mem_flags *object_types, void *device_objects, cl_int *param_value_size, void *param_value, cl_int *param_value_size_ret)
flags: See clCreateFromGLBuffer

egl interoperability [9.19, 9.20]

Create CL Image Objects from EGL [9.19]
These functions require the extension cl_khr_eegl_image.
c_mem clCreateFromEGLImageKHR (cl_context context, cl_mem_flags flags, cl_int *errcode_ret)
cl_mem clCreateFromEGLImageKHR (cl_context context, cl_mem_flags flags, cl_int *errcode_ret)
cl_mem clCreateFromEGLImagePropertiesKHR (cl_int *param_value_size, void *param_value)

Create CL Event Objects from EGL [9.20]
This function requires the extension cl_khr_eegl_event.
c_event clCreateFromEGLSyncKHR (cl_context context, cl_device_id device_id, cl_int *errcode_ret)
OpenCL Reference Card Index

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

A
Access Qualifiers 10
Address Space Qualifier Functions 7
Aligned attribute qualifiers 5
Async Copies and Prefetch 6
Atomic Functions 7
Attribute Qualifiers 5

B
Barriers 4
Blocks 4
Buffer Objects 1-2

C
clKHR 11
clBuildProgram 3
clCompileProgram 3
clCreateBuffer 1
clCreateCommandQueueWithProperties 1
clCreateContext 1
clCreateContextFromType 1
clCreateEventFromEGISyncKHR 11
clCreateEventFromGLSyncKHR 11
clCreateFromD3D10BufferKHR 11
clCreateFromD3D10Texture3DKHR 11
clCreateFromD3D11BufferKHR 11
clCreateFromD3D11Texture3DKHR 11
clCreateFromD3D11Texture3DKHR 11
clCreateFromDX9MediaSurfaceKHR 11
clCreateFromEGImageKHR 11
clCreateFromGLBuffer 11
clCreateFromGLRenderbuffer 11
clCreateFromGLTexture 11
clCreateImage 8
clCreateKernel 3
clCreateKernelsInProgram 3
clCreatePipe 2
clCreateProgramWithBinary 3
clCreateProgramWithBuiltInKernels 3
clCreateProgramWithSource 3
clCreateSamplerWithProperties 10
clCreateSubBuffer 1
clCreateSubDevices 1
clCreateUserEvent 3
clEnqueueAcquireD3D10ObjectsKHR 11
clEnqueueAcquireD3D11ObjectsKHR 11
clEnqueueAcquireDX9MediaSurfacesKHR 11
clEnqueueAcquireEGObjectsKHR 11
clEnqueueAcquireGLObjects 11
clEnqueueBarrierWithWaitList 4
clEnqueueCopyBuffer 2
clEnqueueCopyBufferToImage 8
clEnqueueCopyImage 8
clEnqueueCopyImageToBuffer 8
clEnqueueFillBuffer 2
clEnqueueFillImage 8
clEnqueueMapBuffer 2
clEnqueueMapImage 9
clEnqueueMarkerWithWaitList 4
clEnqueueMigrateMemObjects 2
clEnqueueNativeKernel 3
clEnqueueNDRangeKernel 3
cEnqueueReadBuffer 1
cEnqueueReadBufferRect 1
cEnqueueReadImage 8
cEnqueueReleaseD3D10ObjectsKHR 11
cEnqueueReleaseD3D11ObjectsKHR 11
cEnqueueReleaseDX9MediaSurfacesKHR 11
cEnqueueReleaseEGObjectsKHR 11
cEnqueueReleaseGLObjects 11
cEnqueueSVMMap(m,p, f1l) 3
clEnqueueUnmapMemObject 2
clEnqueueWriteBuffer 2
cEnqueueWriteBufferRect 2
cEnqueueWriteImage 8
cFinish 3
cFlush 3
clGetCommandQueueInfo 1
clGetContextInfo 1
clGetDeviceIDs 1
clGetDeviceIdsFromD3D10KHR 11
clGetDeviceIdsFromD3D11KHR 11
clGetDeviceIdsFromDX9MediaAdapterKHR 11
clGetDeviceInfo 1
cGetEventInfo 3
cGetEventProfilingInfo 4
cGetExtensionFunctionAddressKHR 1
cGetGLContextInfoKHR 11
clGetGLObjectinfoKHR 11
cGetGLTextureInfo 11
cGetKernelArgInfo 3
cGetKernelInfo 3
cGetKernelSubGroupInfoKHR 3
cGetKernelWorkGroupInfo 3
cGetMemObjectInfo 2
cGetMemObjectInfo 9
cGetPipeInfo 2
cGetPlatformIDs 1
clGetPlatformInfo 3
cGetProgramBuildInfo 3
cGetProgramInfo 3
cGetSamplerInfo 10
clGetSupportedImageFormats 8
cIclGetPlatformIDsKHR 1
cIclLinkProgram 3
cReleaseCommandQueue 1
cReleaseContext 1
cReleaseDevice 1
cReleaseEvent 1
cReleaseKernel 3
cReleaseMemObject 2
cReleaseProgram 3
cReleaseSampler 10
cRetainCommandQueue 1
cRetainContext 1
cRetainDevice 1
cRetainEvent 3
cRetainKernel 3
cRetainMemObject 2
clRetainProgram 3
cRetainSampler 10
cSetEventCallback 4
cSetKernelArg 4
cSetKernelArgSVMPointer 3
cSetKernelExecinfo 3
cSetMemObjectDestructorCallback 2
cSetUserEventStatus 3
clSVMAloc 2
clSVMFree 2
cTermImageSurfaceKHR 3
cUnloadPlatformCompiler 3
cWaitForEvents 3
Command Queues 1
Common Built-in Functions 5-6
Compiler Options 3
cost sampler_t 10
Contexts 1
Conversions and Type Casting 2
Copy Between Image, Buffer Objects 8
D
Data Types 4
Debugging options 3
Device Architecture Diagram 2
Direct3D 10 Sharing 11
Direct3D 11 Sharing 11
DX9 Media Surface Sharing 11
E - F
EGL Interoperability 11
Enqueue & Kernel Query Built-in Functions 8
Event Built-in Functions 8
Event Objects 3
Execute Kernels 3
Extension Function Pointers 1
Extensions 11
Fence Functions 6
G - H
Geometric Built-in Functions 6
Helper Built-in Functions 8
I
Image Formats 9
Image Objects 8-9
Image Query Functions 10
Image Read and Write Functions 9-10
Integer Built-in Functions 5
K
Kernel Arguments and Queries 3
Kernel Objects 3
Kernel Query Built-in Functions 8
L
Library linking options 3
Linker Options 3
M
Map and Unmap Image Objects 9
Map Buffer Objects 2
Markers, Barriers, Waiting for Events 4
Math Built-in Functions 5
Math Constants 5
Memory Fence Functions 6
Memory Objects 2
Migrate Memory Objects 2
O
OpenCL Class Diagram 2
OpenCL Extensions 11
OpenGL Sharing 11
Operators 4
Optimization options 3
P
Partitioning a Device 1
Pipe Built-in Functions 8
Pipes 2
Prefetch 6
Preprocessor 3
Preprocessor Directives & Macros 4
printf Function 7
Profiling Operations 4
Program linking options 3
Program Objects 3
Q
Qualifiers 4
Query Image Objects 9
Query Image Functions 10
Query List of Supported Image Formats 8
Query Memory Object 2
Query Program Objects 3
Querying Platform Info & Devices 1
R
Read, Write, Copy Buffer Objects 1-2
Read, Write, Copy, Fill Image Objects 8
Read and Write Image Objects 9-10
Relational Built-in Functions 6
S - T
Sampler Objects, Declaration Fields 10
Scalar Data Types 4
Separate Compilation and Linking 3
Share Objects 11
Shared Virtual Memory 2-3
SPIR compiler options 3
Supported Data Types 4
SVM Sharing Granularity 2
Synchronization & Memory Fence Functions 6
Type Casting Examples 2
Types 4
U - V
Unload the OpenCL Compiler 3
Unroll the OpenCL Compiler 3
Unroll attribute qualifiers 5
Vector Component Addressing 4
Vector Data Load/Store 6
Vector Functions 8
Vector Data Types 3
Version 3
W
Waiting for Events 3
Warning request/suppress 3
Workgroup Functions 7
Work-Item Built-in Functions 4-5

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.