OpenCL API Reference

The OpenCL Platform Layer

The OpenCL platform layer implements platform-specific features that allow applications to query OpenCL devices, device configuration information, and to create OpenCL contexts using one or more devices. Items in blue are taken from sections and text in the OpenCL Extension Spec. Items in purple are taken from sections and text in the OpenCL API Spec.

Querying Platform Info & Devices [4.1-4.2] [5.1-5.2]

- `cl_int cGetPlatformIDs` (cl_uint num_entries, cl_platform_id *platforms, cl_uint *num_platforms)
- `cl_int cGetPlatformInfo` (cl_platform_id platform, cl_platform_info param_name, size_t *param_value_size, void *param_value, size_t *param_value_size_ret)
- `cl_int cGetDeviceIDs` (cl_platform_id platform, cl_device_type device_type, cl_uint num_entries, cl_device_id *devices, cl_uint *num_devices)
- `cl_int cGetDeviceInfo` (cl_device_id device, cl_device_info device_info, size_t *param_value_size, void *param_value, size_t *param_value_size_ret)
- `cl_int cGetEnqueueProperties` (cl_command_queue command_queue, cl_command_queue_properties *properties)

Partitioning a Device [4.3]

- `cl_int cCreateSubDevices` (cl_device_id device_id, cl_device_partition_property *properties, cl_uint num_devices, cl_device_id *devices, cl_uint *num_devices)
- `cl_int cRetainDevice` (cl_device_id device_id)
- `cl_int cReleaseDevice` (cl_device_id device_id)

Contexts [4.4]

- `cl_context cCreateContext` (const cl_context_properties *properties, cl_uint num_devices, cl_device_id *device, cl_device_partition_property *properties)
- `cl_int cGetContextInfo` (cl_context context, cl_context_info param_name, size_t *param_value_size, void *param_value, size_t *param_value_size_ret)
- `cl_int cGetExtensionFunctionPointer` (cl_context context, const char *extension_name, void **extension_function)

Get CL Extension Function Pointers [9.2]

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

Buffer Objects

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

Create Buffer Objects [5.2.1]

- `cl_mem cCreateBuffer` (cl_context context, cl_mem_flags flags, size_t size, void *host_ptr, cl_int *errcode_ret)
- `cl_command_queue cCreateCommandQueueWithProperties` (cl_context context, cl_device_id device, cl_device_partition_property *properties, cl_int *errcode_ret)
- `cl_int cGetCommandQueueInfo` (cl_command_queue command_queue, cl_command_queue_properties *properties)
- `cl_int cGetBufferImageSubBuffer` (cl_mem buffer, size_t offset, size_t size, void *host_ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)
- `cl_int cEnqueueReadBufferRect` (cl_command_queue command_queue, cl_mem buffer, size_t origin, size_t size, size_t offset, void *host_ptr, size_t event_wait_list, cl_event *event)
- `cl_int cEnqueueReadImage` (cl_command_queue command_queue, cl_mem image, cl_image_format format, cl_image_absize origin, cl_image_absize region, void *host_ptr, size_t event_wait_list, cl_event *event)

The OpenCL Runtime

API calls that manage OpenCL objects such as command-queues, memory objects, program objects, kernel objects for __kernel functions in a program and calls that allow you to enqueue commands to a command-queue such as executing a kernel, reading, or writing a memory object.

Command Queues [5.1]

- `cl_int cCreateCommandQueueWithProperties` (cl_context context, cl_device_id device, cl_device_partition_property *properties, cl_int *errcode_ret)
- `cl_int cGetCommandQueueInfo` (cl_command_queue command_queue, cl_command_queue_properties *properties)
- `cl_int cGetBufferImageSubBuffer` (cl_mem buffer, size_t offset, size_t size, void *host_ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)
- `cl_int cEnqueueReadBufferRect` (cl_command_queue command_queue, cl_mem buffer, size_t origin, size_t size, size_t offset, void *host_ptr, size_t event_wait_list, cl_event *event)
- `cl_int cEnqueueReadImage` (cl_command_queue command_queue, cl_mem image, cl_image_format format, cl_image_absize origin, cl_image_absize region, void *host_ptr, size_t event_wait_list, cl_event *event)

(Continued on next page)
**OpenCL Reference Card**

**Kernel**

This conceptual OpenCL device architecture diagram

---

**OpenCL API**

- cl_int clEnqueueWriteBuffer
- cl_int clEnqueueWriteBufferRect
- cl_int clEnqueueFillBuffer

**Memory Objects**

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

- cl_int clRetainMemObject
- cl_int clReleaseMemObject
- cl_int clSetMemObjectDestructorCallback

**Map Buffer Objects**

- void * clEnqueueMapBuffer
- void * clEnqueueMapBufferRect

**Conversions and Type Casting Examples**

<table>
<thead>
<tr>
<th>T a (T b); // Scalar to scalar</th>
<th>_rte to nearest even</th>
</tr>
</thead>
<tbody>
<tr>
<td>T a = convert_T(b);</td>
<td>_rtz toward zero</td>
</tr>
<tr>
<td>T a = convert_T_R(b);</td>
<td>_rtp toward + infinity</td>
</tr>
<tr>
<td>T a = as_T(b);</td>
<td>_rtn toward - infinity</td>
</tr>
<tr>
<td>T a = convert_T_sat_R(b);</td>
<td>R: one of the following rounding modes</td>
</tr>
</tbody>
</table>

---

**Create Pipe Objects**

- cl_mem clCreatePipe
- cl_int clGetPipeInfo

**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.

**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 * clSVMAlloc
- cl_int clSVMFree

**Enqueuing SVM Operations**

- cl_int clEnqueueSVMFree

---

(Continued on next page >)
**Shared Virtual Memory (continued)**

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

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

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

cl_int clEnqueueSVMMunmap(
    cl_command_queue command_queue,
    void *src_ptr, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list,
    cl_event *event)

**Program Objects**

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

**Create Program Objects**

- cl_program clCreateProgramWithSource()
  - cl_program clCreateProgramWithBinary()
  - cl_program clCreateProgramWithBuiltInKernels()
  - cl_program clCreateProgramWithSVMKernels()
  - cl_program clCreateProgramWithBinaryFromBuffer()
  - cl_program clCreateProgramWithFile()
  - cl_program clCreateProgramWithCache()
  - cl_program clCreateProgramWithBuiltInKernelsFromBuffer()
  - cl_program clCreateProgramWithSVMKernelsFromBuffer()

**Create Kernel Objects**

- cl_kernel clCreateKernel()
  - cl_kernel clCreateKernelsInProgram()
  - cl_kernel clCreateKernelInProgram()
  - cl_kernel clCreateKernelFromCache()

**Kernel Arguments and Queries**

- cl_int clSetKernelArg()
  - cl_int clSetKernelArgSVMPointer()

**Build Program Executables**

- cl_int clBuildProgram()
  - cl_int clBuildProgramWithDebugInfo()

**Separate Compilation and Linking**

- cl_int clCompileProgram()
  - cl_int clLinkProgram()

**Execute Kernels**

- cl_int clEnqueueNDRangeKernel()
  - cl_int clEnqueueNativeKernel()

**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.

**Optimization options**

- -cl-opt-disable
- -cl-no-disable
- -cl-until-ndef
- -cl-until-ndef

**Debugging options**

- -D
- -fPIC

**Compiler options**

- -D
- -I dir

**Linker options**

- -create-library
- -enable-link-options

**Flush and Finish**

- cl_int clFlush(cl_command_queue command_queue)

**Event Objects**

- cl_event clCreateUserEvent(cl_context context, cl_int errcode_ret)
Markers, Barriers, Waiting for Events [5.12]

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

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

Profile Operations [5.14]

```c
cl_int clGetEventProfilingInfo(
    cl_event event,  
    cl_profiling_info param_name,  
    size_t *param_value,  
    size_t *param_value_size,  
    const void *user_data  
)
```

Preprocessor Directives & Macros [6.10]

```c
#pragma OPENCL_FP_CONTRACT on-off-switch
```

Reserved Data Types [6.1.4]

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>boolean vector</td>
</tr>
<tr>
<td>char</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>short</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>unsigned char</td>
<td>8-bit unsigned</td>
</tr>
<tr>
<td>wchar_t</td>
<td>16-bit unsigned, vector</td>
</tr>
<tr>
<td>int</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>unsigned int</td>
<td>32-bit unsigned</td>
</tr>
<tr>
<td>float</td>
<td>32-bit float</td>
</tr>
<tr>
<td>double</td>
<td>64-bit float</td>
</tr>
<tr>
<td>complex float</td>
<td>128-bit float, vector</td>
</tr>
<tr>
<td>complex double</td>
<td>128-bit complex, vector</td>
</tr>
<tr>
<td>long</td>
<td>64-bit signed</td>
</tr>
<tr>
<td>unsigned long</td>
<td>64-bit unsigned</td>
</tr>
<tr>
<td>long double</td>
<td>80-bit floating</td>
</tr>
</tbody>
</table>

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>float4</td>
<td>cl_float4</td>
<td>4-component float</td>
</tr>
<tr>
<td>float8</td>
<td>cl_float8</td>
<td>8-component float</td>
</tr>
<tr>
<td>float16</td>
<td>cl_float16</td>
<td>16-component float</td>
</tr>
<tr>
<td>char16</td>
<td>cl_char16</td>
<td>16-bit character</td>
</tr>
<tr>
<td>cl_double</td>
<td>64-bit float</td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td>cl_bool</td>
<td>boolean vector</td>
</tr>
<tr>
<td>cl_int8</td>
<td>8-bit signed</td>
<td></td>
</tr>
<tr>
<td>cl_int16</td>
<td>16-bit signed</td>
<td></td>
</tr>
<tr>
<td>cl_int32</td>
<td>32-bit signed</td>
<td></td>
</tr>
<tr>
<td>cl_int64</td>
<td>64-bit signed</td>
<td></td>
</tr>
<tr>
<td>cl_uint8</td>
<td>8-bit unsigned</td>
<td></td>
</tr>
<tr>
<td>cl_uint16</td>
<td>16-bit unsigned</td>
<td></td>
</tr>
<tr>
<td>cl_uint32</td>
<td>32-bit unsigned</td>
<td></td>
</tr>
<tr>
<td>cl_uint64</td>
<td>64-bit unsigned</td>
<td></td>
</tr>
<tr>
<td>cl_uchar</td>
<td>8-bit unsigned</td>
<td></td>
</tr>
<tr>
<td>cl_ushort</td>
<td>16-bit unsigned</td>
<td></td>
</tr>
<tr>
<td>cl_uint</td>
<td>32-bit unsigned</td>
<td></td>
</tr>
<tr>
<td>cl_ulong</td>
<td>64-bit unsigned</td>
<td></td>
</tr>
<tr>
<td>cl_ulonglong</td>
<td>64-bit signed</td>
<td></td>
</tr>
</tbody>
</table>

Vector Component Addressing [6.1.7]

<table>
<thead>
<tr>
<th>Vector Components</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>float2 x, y</td>
<td>x, y</td>
</tr>
<tr>
<td>float3 x, y, z</td>
<td>x, y, z</td>
</tr>
<tr>
<td>float4 x, y, z, w</td>
<td>x, y, z, w</td>
</tr>
<tr>
<td>float8 x, y, z, s</td>
<td>x, y, z, s</td>
</tr>
<tr>
<td>float16 x, y, z, s, v</td>
<td>x, y, z, s, v</td>
</tr>
</tbody>
</table>

Vector Addressing Equivalences

<table>
<thead>
<tr>
<th>v.lo</th>
<th>v.hi</th>
<th>v.oddd</th>
<th>v.even</th>
</tr>
</thead>
<tbody>
<tr>
<td>v.x0</td>
<td>v.x1</td>
<td>v.x2</td>
<td>v.x3</td>
</tr>
<tr>
<td>v.x4</td>
<td>v.x5</td>
<td>v.x6</td>
<td>v.x7</td>
</tr>
<tr>
<td>v.x8</td>
<td>v.x9</td>
<td>v.xa</td>
<td>v.xb</td>
</tr>
<tr>
<td>v.xc</td>
<td>v.xd</td>
<td>v.xe</td>
<td>v.xf</td>
</tr>
</tbody>
</table>

Operators and Qualifiers

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

```
+ - % / & ^ sizeof
< > <= => & & | || != ! == == =
\`
```

Address Space Qualifiers [6.5]

| _global, local | local, local |
| constant, constant | private, private |

Function Qualifiers [6.7]

```
__kernel __attribute__((vec_type_hint(vec_type)))
```

```
__kernel __attribute__((work_group_size(work_group_size)))
```

```
__kernel __attribute__((reqd_work_group_size(work_group_size)))
```

Work-Item Built-in Functions [6.13.1]

```
uint get_global_dim ()
```

```
uint get_local_dim ()
```

```
int get_global_id (unit dimindex)
```

```
int get_local_id (unit dimindex)
```

```
__kernel __attribute__((lvec_type_hint(lvec_type)))
```

```c
int (*myBlock)(int) =
    ^(int num) (return num * multiplier; )
```

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. Multiplier captured from block's environment.
### Work-Item Functions (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>size_t get_enqueued_local_size (uint dimind)</code></td>
<td>Number of local work-items in a work-item group</td>
</tr>
<tr>
<td><code>size_t get_local_id (uint dimind)</code></td>
<td>Local work-item ID</td>
</tr>
<tr>
<td><code>size_t get_num_groups (uint dimind)</code></td>
<td>Number of work-groups</td>
</tr>
<tr>
<td><code>size_t get_group_id (uint dimind)</code></td>
<td>Work-group ID</td>
</tr>
<tr>
<td><code>size_t get_global_offset (uint dimind)</code></td>
<td>Global offset</td>
</tr>
<tr>
<td><code>size_t get_global_linear_id ()</code></td>
<td>1-dimensional global ID</td>
</tr>
<tr>
<td><code>size_t get_local_linear_id ()</code></td>
<td>1-dimensional local ID</td>
</tr>
<tr>
<td><code>uint get_sub_group_id ()</code></td>
<td>Number of work-items in the subgroup</td>
</tr>
<tr>
<td><code>uint get_max_sub_group_size ()</code></td>
<td>Maximum size of a subgroup</td>
</tr>
<tr>
<td><code>uint get_num_sub_groups ()</code></td>
<td>Number of subgroups</td>
</tr>
<tr>
<td><code>uint get_enqueued_num_sub_groups ()</code></td>
<td>Unique work-item ID</td>
</tr>
</tbody>
</table>

### Math Built-in Functions

#### [6.13.2] [9.4.2]

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MAXFLOAT</code></td>
<td>Value of maximum non-infinite single-precision floating-point number</td>
</tr>
<tr>
<td><code>HUGE_VAL</code></td>
<td>Positive float expression, evaluates to +infinity</td>
</tr>
<tr>
<td><code>HUGE_VALF</code></td>
<td>Positive double expression, evaluates to +infinity</td>
</tr>
<tr>
<td><code>INFINITY</code></td>
<td>Constant float expression, positive or unsigned infinity</td>
</tr>
<tr>
<td><code>NAN</code></td>
<td>Constant float expression, quiet NaN</td>
</tr>
</tbody>
</table>

When double precision is supported, macros ending in _F are available in type double by removing _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]

The functions operate component-wise and use round to nearest even rounding mode. 7 is type integer, int, intn, intn_t, uint, uintn, uintn_t, long, lorn, lorn_t, ulong, ulorn, ulorn_t, or int8, int16, int32, int64, long8, long16, long32, long64, or uint8, uint16, uint32, uint64, ulong8, ulong16, ulong32, ulong64.

### Attribute Qualifiers

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

- `attribute ((aligned))[n]` attribute ((endianness))
- `attribute ((aligned))[n]` attribute ((endianness))
- `attribute ((packed))[n]` attribute ((endianness))

Use to specify special attributes of variables or structure fields.

- `attribute ((aligned)(alignment))` attribute ((nomask))
- `attribute ((aligned)(alignment))` attribute ((nomask))

Use to specify basic blocks and control-flow statements.

- `attribute ((attribution))` attribute ((...)}

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

- `attribute ((opencl_unroll_hint))` attribute ((opencl_unroll_hint))
- `attribute ((opencl_unroll_hint))` attribute ((opencl_unroll_hint))

The following fast integer functions optimize the performance of kernels. In these functions, T is type int, intn, intn_t, uint, uintn, uintn_t, long, lorn, lorn_t, ulong, ulorn, ulorn_t, or int16, int32, int64, long16, long32, long64, uint16, uint32, uint64, ulong16, ulong32, or ulong64.

### Common Built-in Functions

These functions operate component-wise and use round to nearest even rounding mode. 7 is type float, optionally double, or half if cl_khr_fp16 is enabled. 7 is the vector form of 7, where n is 2, 3, 4, 8, or 16. T is 7 and Tn.
Atomic Functions [6.13.11]
OpenCL C implements a subset of the C11 atomics (see section 7.17 of the C11 specification) and synchronization operations.

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 int*. For atomic pointer types, M is ptrdiff_t.
The type atomic_* is a 32-bit integer. atomic_long and atomic_ulong require extension cl_khr_int64_base_atomics or cl_khr_int64_extended_atomics. The atomic_double type requires double precision support. The default scope is work_group for local objects and all _svm_ devices for global objects.

See the table below for Atomic Types and Enum Constants for information about parameter types memory_order, memory_scope, and memory_flag.

### Atomic Types and Enum Constants

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory_order</td>
<td>memory_order_relaxed</td>
<td>Enum which identifies memory ordering constraints.</td>
</tr>
<tr>
<td></td>
<td>memory_order_release</td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory_order_acquire</td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory_order_acquire_rel</td>
<td></td>
</tr>
<tr>
<td>memory_scope</td>
<td>memory_scope_work_item</td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory_scope_work_group</td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory_scope_sub_group</td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory_scope_device</td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory_scope_all_devices</td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory_order_acq_rel</td>
<td></td>
</tr>
<tr>
<td>atomic_flag</td>
<td>32-bit int representing a lock-free, primitive atomic flag; and several atomic analogs of integer types.</td>
<td></td>
</tr>
</tbody>
</table>

### Atomic integer and floating-point types

- atomic_int
- atomic_uint
- atomic_long
- atomic_ulong
- atomic_float
- atomic_double
- atomic_intptr_t
- atomic_uintptr_t
- atomic_size_t
- atomic_ptrdiff_t

### Atomic Macros

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

```
#define ATOMIC_FLAG_INIT                     
  Initialize an atomic_flag to the clear state.
```

### 64-bit Atomics [9.3]
The cl_khr_int64_base_atomics extension enables 64-bit versions of the following functions: atom_add, atom_sub, atom_inc, atom_dec, atom_ashr, atom_cmpxchg
The cl_khr_int64_extended_atomics extension enables 64-bit versions of the following functions: atom_min, atom_max, atomic_and, atomic_or, atomic_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. Subgroups require Double or vector double types require double precision support. The macro CLK_NULL_RESERVED_ID refers to an invalid reservation ID.

```c
int read_pipe (pipe T p, *ptr)
Read packet from p into ptr.

int read_pipe (pipe T p, reserve_id_t reserve_id, uint index, *ptr)
Read packet from reserved area of the pipe reserve_id and index into ptr.

int write_pipe (pipe T p, *ptr)
Write packet specified by ptr to p.

int write_pipe (pipe T p, reserve_id_t reserve_id, uint index, *ptr)
Write packet specified by ptr to reserved area reserve_id and index.

bool is_valid_reserve_id (reserve_id_t reserve_id)
Return true if reserve_id is a valid reservation ID and false otherwise.

void get_pipe_max_packets (pipe T p)
Returns maximum number of packets specified when p was created.

reserve_id_t reserve_id (pipe T p, uint num_packets)
Reserve num_packets entries for reading from or writing to p.

reserve_id_t reserve_id (pipe T p, uint num_packets)
Reserve num_packets entries for reading from or writing to p.

void commit_read_pipe (pipe T p, reserve_id_t reserve_id)
Indicates that all reads and writes to num_packets associated with reservation reserve_id are completed.

void commit_write_pipe (pipe T p, reserve_id_t reserve_id)
Indicates that all reads and writes to num_packets associated with reservation reserve_id are completed.

uint get_pipe_num_packets (pipe T p)
Returns the number of available entries in p.
```

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 is more efficient to execute a block. 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.

```c
int enqueue_kernel (queue_t queue, kernel_enqueue_flag_t flags, const ndrange_t t, *ptr)
Allows a work-item to enqueue a block of device code.

int enqueue_kernel (queue_t queue, kernel_enqueue_flag_t flags, const ndrange_t t, num_events_in_wait_list, const clk_event_t *event_wait_list, clk_event_t *event_ret, *ptr)
Indicates that a work-item has been executed and execution status of the work-item is returned.

int enqueue_kernel (queue_t queue, kernel_enqueue_flag_t flags, const ndrange_t t, num_events_in_wait_list, clk_event_t *event_wait_list, clk_event_t *event_ret, void (*block)(void))
Enqueues a kernel to execute a block.

void get_kernel_max_sub_group_size_for_ndrange (ndrange_t ndrange, uint *size)
Returns number of sub-group size for the ndrange.

void get_kernel_sub_group_count_for_ndrange (const ndrange_t t, reserve_id_t reserve_id, uint *count)
Returns number of subgroups in each subgroup of the dispatch.

void get_kernel_max_sub_group_size_for_ndrange (const ndrange_t t, uint *size)
Returns the maximum sub-group size for a block.

void get_kernel_work_group_size (void (*block)(void))
Queries the maximum work-group size that can be used to execute a block.

void get_kernel_preferred_work_group_size (void (*block)(void))
Returns the preferred multiple of work-group size for launch.

void get_kernel_sub_group_count_for_ndrange (const ndrange_t t, reserve_id_t reserve_id, void (*block)(void))
Returns number of subgroups in each subgroup of the dispatch.

void get_kernel_max_sub_group_size_for_ndrange (const ndrange_t t, void (*block)(void))
Returns the maximum sub-group size for a block.
```

Miscellaneous Vector Functions [6.13.12]

Tm and Tn are type char, uchar, short, ushort, int, uint, long, ulong, float, optionally double, or half if the cl_khr_fp16 extension is supported, where n is 2, 4, 8, or 16 except in vec_step it may also be 3. Tn is uchar, ushort, uint, or ulong.

```c
int vec_step (T a)
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.

int vec_step (typename)
Construct permutation of elements from one or two input vectors, return a vector with same element type as input and length that is the same as the shuffle mask.
```

Event Built-in Functions [6.13.17.8]

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

```c
void retain_event (clk_event_t event)
Increments event reference count.

void release_event (clk_event_t event)
Decrements event reference count.

void set_user_event_status (clk_event_t event, int status)
Sets the execution status of a user event. status: CL_COMPLETE or a negative error value.

void capture_event_profiling_info (clk_event_t event, void (*fn)(void), void (*fn)(void))
Captures profiling information for command associated with event value.
```

Helper Built-in Functions [6.13.17.9]

```c
queue_t get_default_queue (void)
Default queue or CLK_NULL_QUEUE.

ndrange_t ndrange_1D (size_t global_work_size, size_t local_work_size)
Builds a 1D ND-range descriptor.

ndrange_t ndrange_1D (size_t global_work_size, size_t local_work_size)
Builds a 2D ND-range descriptor. n may be 2 or 3.
```

Image Objects

Items in blue apply when the appropriate extension is supported.

Create Image Objects [5.3.1]

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

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

Query List of Supported Image Formats [5.3.2]

```c
cl_int clGetSupportedImageFormats (cl_context context, cl_mem_flags flags, cl_mem_object_type image_type, cl_uint num_formats, cl_image_format *format_list, cl_int *num_image_formats)
flags: See clCreateImage
image_type: CL_MEM_OBJECT_IMAGE1D, CL_MEM_OBJECT_IMAGE2D, CL_MEM_OBJECT_IMAGE3D, CL_MEM_OBJECT_IMAGE2D_ARRAY
```

Read, Write, Copy, Fill Image Objects [5.3.4]

```c
cl_int clEnqueueReadImage (cl_command_queue command_queue, const cl_mem image, void *host_ptr, size_t origin, size_t region, size_t row_pitch, size_t slice_pitch, void *ptr, cl_uint num_events_in_wait_list, cl_event *event, cl_event *event_ret)

cl_int clEnqueueWriteImage (cl_command_queue command_queue, const cl_mem image, void *host_ptr, size_t origin, size_t region, size_t row_pitch, size_t slice_pitch, void *ptr, cl_uint num_events_in_wait_list, cl_event *event, cl_event *event_ret)

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

Copy Between Image, Buffer Objects [5.3.5]

```c
cl_int clEnqueueCopyImageToBuffer (cl_command_queue command_queue, cl_mem src_image, cl_mem dst_buffer, size_t src_origin, size_t dst_origin, size_t region, cl_uint num_events_in_wait_list, cl_event *event, cl_event *event_ret)

cl_int clEnqueueCopyBufferToImage (cl_command_queue command_queue, cl_mem src_buffer, cl_mem dst_image, size_t src_offset, conv_size_t *dst_origin, conv_size_t *dst_region, cl_uint num_events_in_wait_list, cl_event *event, cl_event *event_ret)
```

(Continued on next page >)
Image Read and Write Functions [6.13.14]
The built-in functions defined in this section can only be used with image memory objects created with clCreateImage. sampler specifies the addressing and filtering mode to use. Writing to sRGB images from a kernel requires the cl_khr_srgb_image_writes extension.

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

```c
float4 read_imagef(image1d_t image, sampler_t sampler, int, float4) coord
```

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

```c
half4 read_imageh(image1d_t image, int) coord
```

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

```c
uint4 read_imageui(image1d_t image, sampler_t sampler, int, float4) coord
```

**write_imagef**
Write an element to a 1D image, or read a color value from a location in a 1D image.

```c
void write_imagef(image1d_t image, int coord, float4 color)
```

**write_imageh**
Write an element to a 1D image, or read a color value from a location in a 1D image.

```c
void write_imageh(image1d_t image, int coord, half4 color)
```

**write_imageui**
Write an element to a 1D image, or read a color value from a location in a 1D image.

```c
void write_imageui(image1d_t image, int coord, uint4 color)
```

**write_image**
Write an element to a 1D image, or read a color value from a location in a 1D image.

```c
void write_image(image1d_t image, int coord, int4 color)
```

**write_image**
Write an element to a 1D image, or read a color value from a location in a 1D image.

```c
void write_image(image1d_t image, int coord, int4 color)
```

**write_image**
Write an element to a 1D image, or read a color value from a location in a 1D image.

```c
void write_image(image1d_t image, int coord, uint4 color)
```

**write_image**
Write an element to a 1D image, or read a color value from a location in a 1D image.

```c
void write_image(image1d_t image, int coord, void)
```

Read and write functions for 2D images

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

```c
float4 read_imagef(image2d_t image, sampler_t sampler, int2, float4) coord
```

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

```c
half4 read_imageh(image2d_t image, int2) coord
```

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

```c
uint4 read_imageui(image2d_t image, int, float4) coord
```

**write_imagef**
Write an element to a 2D image, or read a color value from a location in a 2D image.

```c
void write_imagef(image2d_t image, int, float4 color)
```

**write_imageh**
Write an element to a 2D image, or read a color value from a location in a 2D image.

```c
void write_imageh(image2d_t image, int coord, float4 color)
```

**write_imageui**
Write an element to a 2D image, or read a color value from a location in a 2D image.

```c
void write_imageui(image2d_t image, int, float4 color)
```

**write_image**
Write an element to a 2D image, or read a color value from a location in a 2D image.

```c
void write_image(image2d_t image, int coord, int4 color)
```

**write_image**
Write an element to a 2D image, or read a color value from a location in a 2D image.

```c
void write_image(image2d_t image, int coord, int4 color)
```

**write_image**
Write an element to a 2D image, or read a color value from a location in a 2D image.

```c
void write_image(image2d_t image, int coord, uint4 color)
```

**write_image**
Write an element to a 2D image, or read a color value from a location in a 2D image.

```c
void write_image(image2d_t image, int coord, void)
```

Read and write functions for 3D images

**read_imagef**
Read an element from a 3D image, or write a color value to a location in a 3D image.

```c
float4 read_imagef(image3d_t image, sampler_t sampler, int4, float4) coord
```

**read_imageh**
Read an element from a 3D image, or write a color value to a location in a 3D image.

```c
half4 read_imageh(image3d_t image, int4) coord
```

**read_imageui**
Read an element from a 3D image, or write a color value to a location in a 3D image.

```c
uint4 read_imageui(image3d_t image, int4, float4) coord
```

**write_imagef**
Write an element to a 3D image, or read a color value from a location in a 3D image.

```c
void write_imagef(image3d_t image, int, float4 color)
```

**write_imageh**
Write an element to a 3D image, or read a color value from a location in a 3D image.

```c
void write_imageh(image3d_t image, int coord, float4 color)
```

**write_imageui**
Write an element to a 3D image, or read a color value from a location in a 3D image.

```c
void write_imageui(image3d_t image, int coord, void)
```

**write_image**
Write an element to a 3D image, or read a color value from a location in a 3D image.

```c
void write_image(image3d_t image, int coord, int4 color)
```

**write_image**
Write an element to a 3D image, or read a color value from a location in a 3D image.

```c
void write_image(image3d_t image, int coord, int4 color)
```

**write_image**
Write an element to a 3D image, or read a color value from a location in a 3D image.

```c
void write_image(image3d_t image, int coord, uint4 color)
```

**write_image**
Write an element to a 3D image, or read a color value from a location in a 3D image.

```c
void write_image(image3d_t image, int coord, void)
```

**write_image**
Write an element to a 3D image, or read a color value from a location in a 3D image.

```c
void write_image(image3d_t image, int coord, void)
```

(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_gl_mipmap_image` write extensions.

#### Sampler Objects [5.7]

Items in blue require the `cl_khr_mipmap_image` extension.

```c
cl_sampler cCreateSamplerWithProperties ( cl_context context, const cl_sampler_properties * sampler_properties, cl_int * errorcode, ret_t);

cl_sampler_properties: (Table 5.14)
   CL_SAMPLER_NORMALIZED_COORDS, CL_SAMPLER_ADDRESSING, FILTER_MODE, CL_SAMPLER_LOAD_MIN, MAX

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

cl_sglamplerInfo (cl_sampler sampler, cl_sampler_info * sampler_info, size_t param_name, void * param_value, size_t * param_value_size, ret_t);
```


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

```c
const sampler_t * sampler-name = sampler-normalized-mode | <address-mode> | <filter-mode> normalization-mode:
   CLK_NORMALIZED_COORDS_TRUE, FALSE
   address-mode:
   CLK_ADDRESS (REPEAT, CLAMP, NONE)
   CLK_ADDRESS (CLAMP_TO_EDGE)
   CLK_ADDRESS (MIRRORED_REPEAT)
   filter-mode:
   CLK_FILTER_NEAREST, CLK_FILTER_LINEAR
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>float read_image()</code></td>
<td>Read a 1D texture image from the sampler</td>
</tr>
<tr>
<td><code>uint4 read_image()</code></td>
<td>Read a 2D texture image from the sampler</td>
</tr>
<tr>
<td><code>uint4 read_image()</code></td>
<td>Read a 3D texture image from the sampler</td>
</tr>
<tr>
<td><code>uint4 read_image()</code></td>
<td>Read a 2D array texture image from the sampler</td>
</tr>
<tr>
<td><code>uint4 read_image()</code></td>
<td>Read a 3D array texture image from the sampler</td>
</tr>
<tr>
<td><code>uint4 read_image()</code></td>
<td>Read a 1D array texture image from the sampler</td>
</tr>
<tr>
<td><code>uint4 read_image()</code></td>
<td>Read a 2D array texture image from the sampler</td>
</tr>
<tr>
<td><code>uint4 read_image()</code></td>
<td>Read a 3D array texture image from the sampler</td>
</tr>
</tbody>
</table>

#### Access Qualifiers [6.6]

Apply to 2D and 3D image types to declare if the image memory object is being read or written by a kernel.

```c
<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>read only</code>, <code>read only</code></td>
<td>Read-only access to the image</td>
</tr>
<tr>
<td><code>write only</code>, <code>write only</code></td>
<td>Write-only access to the image</td>
</tr>
</tbody>
</table>
```

A C++ wrapper is available for developing OpenCL applications in C++.

See www.khronos.org/registry/cl/

©2013 Khronos Group - Rev. 1113

www.khronos.org/opencl
Using OpenCL Extensions

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: clGetExtensionFunctionAddressForPlatform()

To get the address of the extension function: clGetExtensionFunctionAddress()

OpenCL 2.0 Reference Card

These functions require the extension cl_khr_gl_sharing or cl_apple_gl_sharing extension.

CL Image Objects > GL Renderbuffers

These functions require the extension cl_khr_fromGLRenderbuffer().

CL Buffer Objects > GL Buffer Objects

These functions require the extension cl_khr_gl_msaa_sharing.

Query Information

These functions require the extension cl_khr_d3d10_sharing.

Direct3D 10 Sharing

These functions require the extension cl_khr_d3d10_sharing.

Direct3D 11 Sharing

These functions require the extension cl_khr_d3d11_sharing.

Create CL Image Objects from EGL

These functions require the extension cl_khr_egl_image.

Create CL Event Objects from D3D10

These functions require the extension cl_khr_d3d10_sharing.

Create CL Event Objects from D3D11

These functions require the extension cl_khr_d3d11_sharing.

EGL Interoperability

These functions require the extension cl_khr_egl_image.
The Krakenos 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 Krakenos Group.

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

©2013 Khronos Group - Rev. 1113
Reference card production by Miller & Mattsson www.miller&mattsson.com www.khronos.org/opencl