The OpenCL™ Extension Specification

Khronos® OpenCL Working Group

Version v3.0.15, Thu, 14 Dec 2023 20:00:00 +0000: from git branch: main commit: 3c1a38cbdbbe6586f44c718e87585de4b2924a3f

Table of Contents

1. Extensions Overview 2
  1.1. Naming Convention for Optional Extensions 2
  1.2. Compiler Directives for Optional Extensions 2
  1.3. Getting OpenCL API Extension Function Pointers 4
2. Installable Client Drivers 6
  2.1. Overview 6
  2.2. General Information 6
  2.3. Inferring Vendors from Function Call Arguments 6
  2.4. ICD Data 7
  2.5. ICD Loader Vendor Enumeration on Windows 7
  2.6. ICD Loader Vendor Enumeration on Linux 8
  2.7. ICD Loader Vendor Enumeration on Android 8
  2.8. Adding a Vendor Library 9
  2.9. New Procedures and Functions 9
  2.10. New Tokens 9
  2.11. Additions to Chapter 4 of the OpenCL 2.2 Specification 10
  2.12. Source Code 11
  2.13. Issues 11
3. Byte Addressable Stores 12
  3.1. General Information 12
4. Writing to 3D Image Objects 13
  4.1. General Information 13
5. Half Precision Floating-Point 15
  5.1. General Information 15
  5.2. Additions to Chapter 6 of the OpenCL 2.0 C Specification 15
  5.3. Additions to Chapter 8 of the OpenCL 2.0 C Specification 48
6. Double Precision Floating-Point 51
  6.1. General Information 51
  6.2. Additions to Chapter 6 51
7. 32-bit Atomics 73
  7.1. General Information 73
  7.2. Global Atomics for 32-bit Integers 73
  7.3. Local Atomics for 32-bit Integers 75
8. 64-bit Atomics 77
  8.1. General Information 77
9. Selecting the Rounding Mode **(DEPRECATED)** 80
  9.1. General Information 80
  9.2. Changes to OpenCL C specification 80

10. Creating an OpenCL Context from an OpenGL Context or Share Group . . . 82
10.1. Overview . . . 82
10.2. General Information . . . 82
10.3. New Procedures and Functions . . . 82
10.4. New Tokens . . . 82
10.5. Additions to Chapter 4 of the OpenCL 2.2 Specification . . . 83
10.6. Additions to Chapter 5 of the OpenCL 2.2 Specification . . . 85
10.7. Issues . . . 87
11. Creating OpenCL Memory Objects from OpenGL Objects . . . 90
11.1. General Information . . . 90
11.2. Lifetime of Shared Objects . . . 90
11.3. OpenCL Buffer Objects from OpenGL Buffer Objects . . . 90
11.4. OpenCL Image Objects from OpenGL Textures . . . 91
11.5. OpenCL Image Objects from OpenGL Renderbuffers . . . 95
11.6. Querying OpenGL object information from an OpenCL memory object . . . 96
11.7. Sharing memory objects that map to GL objects between GL and CL contexts . . . 98
12. Creating OpenCL Event Objects from OpenGL Sync Objects . . . 102
12.1. Overview . . . 102
12.2. General Information . . . 102
12.3. New Procedures and Functions . . . 102
12.4. New Tokens . . . 102
12.5. Additions to Chapter 5 of the OpenCL 2.2 Specification . . . 102
12.6. Additions to the OpenCL Extension Specification . . . 104
12.7. Issues . . . 105
13. Creating OpenCL Memory Objects from Direct3D 10 Buffers and Textures . . . 107
13.1. Overview . . . 107
13.2. General Information . . . 107
13.3. New Procedures and Functions . . . 107
13.4. New Tokens . . . 108
13.5. Additions to Chapter 4 of the OpenCL 2.2 Specification . . . 109
13.6. Additions to Chapter 5 of the OpenCL 2.2 Specification . . . 110
13.7. Sharing Memory Objects with Direct3D 10 Resources . . . 111
13.8. Issues . . . 121
14. Creating OpenCL Memory Objects from Direct3D 11 Buffers and Textures . . . 123
14.1. Overview . . . 123
14.2. General Information . . . 123
14.3. New Procedures and Functions . . . 123
14.4. New Tokens . . . 124
14.5. Additions to Chapter 4 of the OpenCL 2.2 Specification . . . 125
14.6. Additions to Chapter 5 of the OpenCL 2.2 Specification . . . 126
14.7. Sharing Memory Objects with Direct3D 11 Resources . . . 127

15. Creating OpenCL Memory Objects from DirectX 9 Media Surfaces . . . 138
15.1. Overview . . . 138
15.2. General Information . . . 138
15.3. New Procedures and Functions . . . 138
15.4. New Tokens . . . 139
15.5. Additions to Chapter 4 of the OpenCL 2.2 Specification . . . 140
15.6. Additions to Chapter 5 of the OpenCL 2.2 Specification . . . 141
15.7. Sharing Media Surfaces with OpenCL . . . 142
16. Depth Images . . . 151
16.1. General Information . . . 151
16.2. Additions to Chapter 5 of the OpenCL 1.2 Specification . . . 151
16.3. Additions to Chapter 6 of the OpenCL 1.2 Specification . . . 152
17. Sharing OpenGL and OpenGL ES Depth and Depth-Stencil Images . . . 157
17.1. General Information . . . 157
17.2. Additions to Chapter 5 of the OpenCL 2.2 Specification . . . 157
17.3. Additions to the OpenCL Extension Specification . . . 158
18. Creating OpenCL Memory Objects from OpenGL MSAA Textures . . . 159
18.1. General Information . . . 159
18.2. Additions to the OpenCL Extension Specification . . . 159
18.3. Additions to Chapter 5 of the OpenCL 2.2 Specification . . . 160
18.4. Additions to Chapter 6 of the OpenCL 2.2 Specification . . . 160
19. Creating OpenCL Event Objects from EGL Sync Objects . . . 166
19.1. Overview . . . 166
19.2. General Information . . . 166
19.3. New Procedures and Functions . . . 166
19.4. New Tokens . . . 166
19.5. Additions to Chapter 5 of the OpenCL 2.2 Specification . . . 166
19.6. Additions to the OpenCL Extension Specification . . . 168
19.7. Issues . . . 168
20. Creating OpenCL Memory Objects from EGL Images . . . 170
20.1. Overview . . . 170
20.2. General Information . . . 170
20.3. New Procedures and Functions . . . 170
20.4. New Tokens . . . 170
20.5. Additions to Chapter 5 of the OpenCL 2.2 Specification . . . 171
20.6. Issues . . . 176
21. Creating a 2D Image From A Buffer . . . 177
21.1. General Information . . . 177
21.2. Additions to Chapter 4 of the OpenCL 1.2 Specification . . . 177
21.3. Additions to Chapter 5 of the OpenCL 1.2 Specification . . . 178
22. Local and Private Memory Initialization . . . 179

22.1. General Information . . . 179
22.2. Additions to Chapter 4 of the OpenCL 2.2 Specification . . . 179
22.3. Additions to Chapter 6 of the OpenCL 2.2 Specification . . . 179
23. Terminating OpenCL Contexts . . . 181
23.1. General Information . . . 181
23.2. Additions to Chapter 4 of the OpenCL 2.2 Specification . . . 181
24. Standard Portable Intermediate Representation Binaries . . . 184
24.1. General Information . . . 184
24.2. Additions to Chapter 4 of the OpenCL 2.2 Specification . . . 184
24.3. Additions to Chapter 5 of the OpenCL 2.2 Specification . . . 184
25. Intermediate Language Programs . . . 187
25.1. General Information . . . 187
25.2. New Procedures and Functions . . . 187
25.3. New Tokens . . . 187
25.4. Additions to Chapter 3 of the OpenCL 2.0 Specification . . . 187
25.5. Additions to Chapter 4 of the OpenCL 2.0 Specification . . . 188
25.6. Additions to Chapter 5 of the OpenCL 2.0 Specification . . . 188
26. Creating Command-Queues with Properties . . . 191
26.1. Overview . . . 191
26.2. General Information . . . 191
26.3. New API Functions . . . 191
26.4. New API Types . . . 191
26.5. Modifications to the OpenCL 1.2 Specification . . . 192
27. Device Enqueue Local Argument Types . . . 195
27.1. General Information . . . 195
27.2. Additions to Chapter 6 of the OpenCL 2.0 C Specification . . . 195
28. Sub-groups . . . 196
28.1. General Information . . . 196
28.2. Additions to Chapter 3 of the OpenCL 2.0 Specification . . . 196
28.3. Additions to section 3.2 — Execution Model . . . 196
28.4. Additions to Chapter 5 of the OpenCL 2.0 Specification . . . 197
28.5. Additions to Chapter 6 of the OpenCL 2.0 C Specification . . . 200
29. Mipmaps . . . 206
29.1. General Information . . . 206
29.2. Additions to Chapter 5 of the OpenCL 2.2 Specification . . . 206
29.3. Additions to Chapter 6 of the OpenCL 2.0 Specification . . . 207
29.4. Additions to Creating OpenCL Memory Objects from OpenGL Objects . . . 217
30. sRGB Image Writes . . . 218
30.1. General Information . . . 218
31. Priority Hints . . . 219
31.1. General Information . . . 219

31.2. Host-side API modifications . . . 219
32. Throttle Hints . . . 220
32.1. General Information . . . 220
32.2. Host-side API modifications . . . 220
33. Named Barriers for Sub-groups . . . 221
33.1. General Information . . . 221
33.2. Changes to OpenCL specification . . . 221
34. Extended Async Copies . . . 222
34.1. General Information . . . 222
34.2. Additions to Chapter 6 of the OpenCL C Specification . . . 222
35. Async Work-group Copy Fence . . . 225
35.1. General Information . . . 225
35.2. Additions to Chapter 6 of the OpenCL C Specification . . . 225
36. Unique Device Identifiers . . . 227
36.1. General Information . . . 227
36.2. Additions to Chapter 4 of the OpenCL 3.0 API Specification . . . 227
37. Extended versioning . . . 229
37.1. General Information . . . 229
37.2. New API Types . . . 229
37.3. New API Enums . . . 230
37.4. Modifications to the OpenCL API Specification . . . 231
37.5. Conformance tests . . . 232
37.6. Issues . . . 233
38. Extended Sub-group Functions . . . 234
38.1. Overview . . . 234
38.2. General Information . . . 234
38.3. Summary of New OpenCL C Functions . . . 235
38.4. Extended Types . . . 237
38.5. Votes and Elections . . . 237
38.6. Ballots . . . 238
38.7. Non-Uniform Arithmetic . . . 242
38.8. General Purpose Shuffles . . . 246
38.9. Relative Shuffles . . . 247
38.10. Clustered Reductions . . . 248
38.11. Function Mapping and Capabilities . . . 250
39. PCI Bus Information Query . . . 256
39.1. General Information . . . 256
39.2. New API Types . . . 256
39.3. New API Enums . . . 256
39.4. Modifications to the OpenCL API Specification . . . 257
40. Extended Bit Operations . . . 258

40.1. General Information 258
40.2. New OpenCL C Functions 258
40.3. Modifications to the OpenCL C Specification 258
41. Suggested Local Work Size Query 261
41.1. General Information 261
41.2. New API Functions 261
41.3. Modifications to the OpenCL API Specification 261
42. Integer dot product 264
42.1. General Information 264
42.2. New API Enums 264
42.3. New OpenCL C Functions 265
42.4. Modifications to the OpenCL API Specification 265
42.5. Modifications to the OpenCL C Specification 267
42.6. Modifications to the OpenCL SPIR-V Environment Specification 268
42.7. Interactions with Other Extensions 269
43. Semaphores (Provisional) 270
43.1. General Information 270
43.2. New Types 271
43.3. New API Functions 271
43.4. New API Enums 272
43.5. Modifications to existing APIs added by this spec 273
43.6. Description of new types added by this spec 273
43.7. Description of new APIs added by this spec 274
43.8. Sample Code 281
44. External Semaphores (Provisional) 285
44.1. General Information 285
44.2. New Types 287
44.3. New API Functions 287
44.4. New API Enums 287
44.5. Modifications to existing APIs added by this spec 288
44.6. Exporting semaphore external handles 290
44.7. Importing semaphore external handles 291
44.8. Descriptions of External Semaphore Handle Types 292
44.9. Sample Code 295
45. External Memory (Provisional) 300
45.1. General Information 300
45.2. New Types 301
45.3. New API Functions 301
45.4. New API Enums 302
45.5. Modifications to existing APIs added by this spec 303
45.6. Description of new types added by this spec 305

45.7. Descriptions of External Memory Handle Types . . . 308
45.8. Sample Code . . . 310
45.9. Issues . . . 313
46. Command Buffers (Provisional) . . . 314
46.1. General Information . . . 314
46.2. Overview . . . 315
46.3. Interactions with Other Extensions . . . 316
46.4. New Types . . . 317
46.5. New API Functions . . . 318
46.6. New API Enums . . . 322
46.7. Modifications to section 4.2 of the OpenCL API Specification . . . 323
46.8. Add new section "Section 5.X - Command Buffers" to OpenCL API Specification . . . 323
46.9. Modifications to section 5.11 of the OpenCL API Specification . . . 349
46.10. Sample Code . . . 350
46.11. Issues . . . 353
47. Kernel Optimization Hints . . . 354
47.1. General Information . . . 354
47.2. Sample Code . . . 354
48. Sub-group Rotation . . . 356
48.1. General Information . . . 356
48.2. New OpenCL C Functions . . . 356
48.3. Modifications to the OpenCL C Specification . . . 356
48.4. Modifications to the OpenCL SPIR-V Environment Specification . . . 358
48.5. Interactions with Other Extensions . . . 358
49. Work-group Uniform Arithmetic . . . 359
49.1. General Information . . . 359
49.2. New OpenCL C Functions . . . 359
49.3. Modifications to the OpenCL C Specification . . . 360
49.4. Issues . . . 363
50. Command Buffers - Mutable Dispatch (Provisional) . . . 364
50.1. General Information . . . 364
50.2. Overview . . . 365
50.3. Interactions with Other Extensions . . . 365
50.4. New Types . . . 365
50.5. New API Functions . . . 368
50.6. New API Enums . . . 368
50.7. Modifications to section 4.2 of the OpenCL API Specification . . . 369
50.8. Modifications to Section 5.X - Command Buffers of the OpenCL API Specification . . . 370
50.9. Sample Code . . . 379
50.10. Issues . . . 383
51. Command Buffers - Multiple Devices (Provisional) . . . 385

51.1. General Information . . . 385
51.2. Overview . . . 386
51.3. New Types . . . 386
51.4. New API Functions . . . 386
51.5. New API Enums . . . 386
51.6. Modifications to section 4.1 of the OpenCL API Specification . . . 387
51.7. Modifications to section 4.2 of the OpenCL API Specification . . . 388
51.8. Modifications to section 5.11 of the OpenCL API Specification . . . 389
51.9. Modifications to section 5.14 of the OpenCL API Specification . . . 389
51.10. Modifications to Section 5.X - Command Buffers of the OpenCL API Specification . . . 390
51.11. Sample Code . . . 395
51.12. Issues . . . 399
52. Extensions to the OpenCL SPIR-V Environment . . . 400
Index . . . 401
Appendix A: Extensions Promoted to Core Features . . . 402
A.1. For OpenCL 1.1: . . . 402
A.2. For OpenCL 1.2: . . . 402
A.3. For OpenCL 2.0: . . . 402
A.4. For OpenCL 2.1: . . . 402
A.5. For OpenCL 3.0: . . . 402
Appendix B: Deprecated Extensions . . . 403
B.1. For OpenCL 1.1: . . . 403
Appendix C: Quick Reference . . . 404

Chapter 1. Extensions Overview

This document describes the list of optional features supported by OpenCL. Optional extensions are not required to be supported by a conformant OpenCL implementation, but are expected to be widely available, and in some cases may define functionality that is likely to be required in a future revision of the OpenCL specification.

This document describes all extensions that have been approved by the OpenCL working group. It is a *unified* specification, meaning that the extensions described in this document are not specific to a specific core OpenCL specification version.

OpenCL extensions approved by the OpenCL working group may be *promoted* to core features in later revisions of OpenCL. When this occurs, the feature described by the extension specification is merged into the core OpenCL specification. The extension will continue to be documented in this specification, both for backwards compatibility and for devices that wish to support the feature but that are unable to support the newer core OpenCL version.

1.1. Naming Convention for Optional Extensions

OpenCL extensions approved by the OpenCL working group use the following naming convention:

- A unique *name string* of the form **"cl_khr_<name>"** is associated with each extension. If the extension is supported by an implementation, this string will be present in the implementation's `CL_PLATFORM_EXTENSIONS` string or `CL_DEVICE_EXTENSIONS` string.
- All API functions defined by the extension will have names of the form **cl<*function_name*>KHR**.
- All enumerants defined by the extension will have names of the form **CL_<*enum_name*>_KHR.**

Functions and enumerants defined by extensions that are promoted to core features will have their **KHR** affix removed. OpenCL implementations of such later revisions must also export the name strings of promoted extensions in the `CL_PLATFORM_EXTENSIONS` or `CL_DEVICE_EXTENSIONS` string, and support the **KHR**-affixed versions of functions and enumerants as a transition aid.

Vendor extensions are strongly encouraged to follow a similar naming convention:

- A unique *name string* of the form **"cl_<vendor_name>_<name>"** is associated with each extension. If the extension is supported by an implementation, this string will be present in the implementation's `CL_PLATFORM_EXTENSIONS` string or `CL_DEVICE_EXTENSIONS` string.
- All API functions defined by the vendor extension will have names of the form **cl<*function_name*><*vendor_name*>**.
- All enumerants defined by the vendor extension will have names of the form **CL_<*enum_name*>_<*vendor_name*>.**

1.2. Compiler Directives for Optional Extensions

The **#pragma OPENCL EXTENSION** directive controls the behavior of the OpenCL compiler with respect to extensions. The **#pragma OPENCL EXTENSION** directive is defined as:

```
#pragma OPENCL EXTENSION <extension_name> : <behavior>
#pragma OPENCL EXTENSION all : <behavior>
```

where *extension_name* is the name of the extension. The *extension_name* will have names of the form **cl_khr_<*name*>** for an extension approved by the OpenCL working group and will have names of the form **cl_<*vendor_name*>_<*name*>** for vendor extensions. The token **all** means that the behavior applies to all extensions supported by the compiler. The *behavior* can be set to one of the following values given by the table below.

behavior	**Description**
enable	Behave as specified by the extension *extension_name*. Report an error on the `#pragma OPENCL EXTENSION` if the *extension_name* is not supported, or if **all** is specified.
disable	Behave (including issuing errors and warnings) as if the extension *extension_name* is not part of the language definition. If **all** is specified, then behavior must revert back to that of the non-extended core version of the language being compiled to. Warn on the `#pragma OPENCL EXTENSION` if the extension *extension_name* is not supported.

The `#pragma OPENCL EXTENSION` directive is a simple, low-level mechanism to set the behavior for each extension. It does not define policies such as which combinations are appropriate; those must be defined elsewhere. The order of directives matter in setting the behavior for each extension. Directives that occur later override those seen earlier. The **all** variant sets the behavior for all extensions, overriding all previously issued extension directives, but only if the *behavior* is set to **disable**.

The initial state of the compiler is as if the directive

```
#pragma OPENCL EXTENSION all : disable
```

was issued, telling the compiler that all error and warning reporting must be done according to this specification, ignoring any extensions.

Every extension which affects the OpenCL language semantics, syntax or adds built-in functions to the language must create a preprocessor `#define` that matches the extension name string. This `#define` would be available in the language if and only if the extension is supported on a given implementation.

Example:

An extension which adds the extension string "cl_khr_3d_image_writes" should also add a preprocessor #define called **cl_khr_3d_image_writes**. A kernel can now use this preprocessor #define to do something like:

```
#ifdef cl_khr_3d_image_writes
    // do something using the extension
#else
    // do something else or #error!
#endif
```

1.3. Getting OpenCL API Extension Function Pointers

The function

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

returns the address of the extension function named by *funcname* for a given *platform* The pointer returned should be cast to a function pointer type matching the extension function's definition defined in the appropriate extension specification and header file. A return value of NULL indicates that the specified function does not exist for the implementation or *platform* is not a valid platform. A non-NULL return value for **clGetExtensionFunctionAddressForPlatform** does not guarantee that an extension function is actually supported by the platform. The application must also make a corresponding query using **clGetPlatformInfo**(platform, CL_PLATFORM_EXTENSIONS, …) or **clGetDeviceInfo**(device, CL_DEVICE_EXTENSIONS, …) to determine if an extension is supported by the OpenCL implementation.

Since there is no way to qualify the query with a device, the function pointer returned must work for all implementations of that extension on different devices for a platform. The behavior of calling a device extension function on a device not supporting that extension is undefined.

clGetExtensionFunctionAddressForPlatform may not be be used to query for core (non-extension) functions in OpenCL. For extension functions that may be queried using **clGetExtensionFunctionAddressForPlatform**, implementations may also choose to export those functions statically from the object libraries implementing those functions, however, portable applications cannot rely on this behavior.

Function pointer typedefs must be declared for all extensions that add API entrypoints. These typedefs are a required part of the extension interface, to be provided in an appropriate header (such as cl_ext.h if the extension is an OpenCL extension, or cl_gl_ext.h if the extension is an OpenCL / OpenGL sharing extension).

The following convention must be followed for all extensions affecting the host API:

```
#ifndef extension_name
#define extension_name 1
```

```
// all data typedefs, token #defines, prototypes, and
// function pointer typedefs for this extension

// function pointer typedefs must use the
// following naming convention

typedef return_type
            (CL_API_CALL *clExtensionFunctionNameTAG_fn)(...);

#endif // _extension_name_
```

where `TAG` can be `KHR`, `EXT` or `vendor-specific`.

Consider, for example, the **cl_khr_gl_sharing** extension. This extension would add the following to cl_gl_ext.h:

```
#ifndef cl_khr_gl_sharing
#define cl_khr_gl_sharing 1

// all data typedefs, token #defines, prototypes, and
// function pointer typedefs for this extension
#define CL_INVALID_GL_SHAREGROUP_REFERENCE_KHR  -1000
#define CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR    0x2006
#define CL_DEVICES_FOR_GL_CONTEXT_KHR           0x2007
#define CL_GL_CONTEXT_KHR                       0x2008
#define CL_EGL_DISPLAY_KHR                      0x2009
#define CL_GLX_DISPLAY_KHR                      0x200A
#define CL_WGL_HDC_KHR                          0x200B
#define CL_CGL_SHAREGROUP_KHR                   0x200C

// function pointer typedefs must use the
// following naming convention
typedef cl_int
        (CL_API_CALL *clGetGLContextInfoKHR_fn)(
            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*/);

#endif // cl_khr_gl_sharing
```

Chapter 2. Installable Client Drivers

2.1. Overview

This section describes a platform extension which defines a simple mechanism through which the Khronos OpenCL installable client driver loader (ICD Loader) may expose multiple separate vendor installable client drivers (Vendor ICDs) for OpenCL. An application written against the ICD Loader will be able to access all `cl_platform_id`s exposed by all vendor implementations with the ICD Loader acting as a demultiplexor.

This is a platform extension, so if this extension is supported by an implementation, the string **cl_khr_icd** will be present in the `CL_PLATFORM_EXTENSIONS` string.

2.2. General Information

2.2.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

2.3. Inferring Vendors from Function Call Arguments

At every OpenCL function call, the ICD Loader infers the vendor ICD function to call from the arguments to the function. An object is said to be ICD compatible if it is of the following structure:

```
struct _cl_<object>
{
    struct _cl_icd_dispatch *dispatch;
    // ... remainder of internal data
};
```

<object> is one of platform_id, device_id, context, command_queue, mem, program, kernel, event, or sampler.

The structure `_cl_icd_dispatch` is a function pointer dispatch table which is used to direct calls to a particular vendor implementation. All objects created from ICD compatible objects must be ICD compatible.

The definition for `_cl_icd_dispatch` is provided along with the OpenCL headers. Existing members can never be removed from that structure but new members can be appended.

Functions which do not have an argument from which the vendor implementation may be inferred have been deprecated and may be ignored.

2.4. ICD Data

A Vendor ICD is defined by two pieces of data:

- The Vendor ICD library specifies a library which contains the OpenCL entry points for the vendor's OpenCL implementation. The vendor ICD's library file name should include the vendor name, or a vendor-specific implementation identifier.
- The Vendor ICD extension suffix is a short string which specifies the default suffix for extensions implemented only by that vendor. The vendor suffix string is optional.

2.5. ICD Loader Vendor Enumeration on Windows

To enumerate Vendor ICDs on Windows, the ICD Loader will first scan for REG_SZ string values in the "Display Adapter" and "Software Components" HKR registry keys. The exact registry keys to scan should be obtained via PnP Configuration Manager APIs, but will look like:

For 64-bit ICDs:

```
HKLM\SYSTEM\CurrentControlSet\Control\Class\
{Display Adapter GUID}\{Instance ID}\OpenCLDriverName, or

HKLM\SYSTEM\CurrentControlSet\Control\Class\
{Software Component GUID}\{Instance ID}\OpenCLDriverName
```

For 32-bit ICDs:

```
HKLM\SYSTEM\CurrentControlSet\Control\Class\
{Display Adapter GUID}\{Instance ID}\OpenCLDriverNameWoW, or

HKLM\SYSTEM\CurrentControlSet\Control\Class\
{Software Component GUID}\{Instance ID}\OpenCLDriverNameWoW
```

These registry values contain the path to the Vendor ICD library. For example, if the registry contains the value:

```
[HKLM\SYSTEM\CurrentControlSet\Control\Class\{GUID}\{Instance}]
"OpenCLDriverName"="c:\\vendor a\\vndra_ocl.dll"
```

Then the ICD Loader will open the Vendor ICD library:

```
c:\vendor a\vndra_ocl.dll
```

The ICD Loader will also scan for REG_DWORD values in the registry key:

```
HKLM\SOFTWARE\Khronos\OpenCL\Vendors
```

For each registry value in this key which has data set to 0, the ICD Loader will open the Vendor ICD library specified by the name of the registry value.

For example, if the registry contains the value:

```
[HKLM\SOFTWARE\Khronos\OpenCL\Vendors]
"c:\\vendor a\\vndra_ocl.dll"=dword:00000000
```

Then the ICD Loader will open the Vendor ICD library:

```
c:\vendor a\vndra_ocl.dll
```

2.6. ICD Loader Vendor Enumeration on Linux

To enumerate vendor ICDs on Linux, the ICD Loader scans the files in the path `/etc/OpenCL/vendors`. For each file in this path, the ICD Loader opens the file as a text file. The expected format for the file is a single line of text which specifies the Vendor ICD's library. The ICD Loader will attempt to open that file as a shared object using dlopen(). Note that the library specified may be an absolute path or just a file name.

For example, if the following file exists

```
/etc/OpenCL/vendors/VendorA.icd
```

and contains the text

```
libVendorAOpenCL.so
```

then the ICD Loader will load the library `libVendorAOpenCL.so`.

2.7. ICD Loader Vendor Enumeration on Android

To enumerate vendor ICDs on Android, the ICD Loader scans the files in the path `/system/vendor/Khronos/OpenCL/vendors`. For each file in this path, the ICD Loader opens the file as a text file. The expected format for the file is a single line of text which specifies the Vendor ICD's library. The ICD Loader will attempt to open that file as a shared object using dlopen(). Note that the library specified may be an absolute path or just a file name.

For example, if the following file exists

```
/system/vendor/Khronos/OpenCL/vendors/VendorA.icd
```

and contains the text

```
libVendorAOpenCL.so
```

then the ICD Loader will load the library `libVendorAOpenCL.so`.

2.8. Adding a Vendor Library

Upon successfully loading a Vendor ICD's library, the ICD Loader queries the following functions from the library: **clIcdGetPlatformIDsKHR**, **clGetPlatformInfo**, and **clGetExtensionFunctionAddress** (note: **clGetExtensionFunctionAddress** has been deprecated, but is still required for the ICD Loader). If any of these functions are not present then the ICD Loader will close and ignore the library.

Next the ICD Loader queries available ICD-enabled platforms in the library using **clIcdGetPlatformIDsKHR**. For each of these platforms, the ICD Loader queries the platform's extension string to verify that **cl_khr_icd** is supported, then queries the platform's Vendor ICD extension suffix using **clGetPlatformInfo** with the value `CL_PLATFORM_ICD_SUFFIX_KHR`.

If any of these steps fail, the ICD Loader will ignore the Vendor ICD and continue on to the next.

2.9. New Procedures and Functions

```
cl_int clIcdGetPlatformIDsKHR(cl_uint num_entries,
                              cl_platform_id *platforms,
                              cl_uint *num_platforms);
```

2.10. New Tokens

Accepted as *param_name* to the function **clGetPlatformInfo**:

```
CL_PLATFORM_ICD_SUFFIX_KHR
```

Returned by **clGetPlatformIDs** when no platforms are found:

```
CL_PLATFORM_NOT_FOUND_KHR
```

2.11. Additions to Chapter 4 of the OpenCL 2.2 Specification

In *section 4.1*, replace the description of the return values of **clGetPlatformIDs** with:

"**clGetPlatformIDs** returns `CL_SUCCESS` if the function is executed successfully and there are a non zero number of platforms available. It returns `CL_PLATFORM_NOT_FOUND_KHR` if zero platforms are available. It returns `CL_INVALID_VALUE` if *num_entries* is equal to zero and *platforms* is not `NULL` or if both *num_platforms* and *platforms* are `NULL`."

In *section 4.1*, add the following after the description of **clGetPlatformIDs**:

"The list of platforms accessible through the Khronos ICD Loader can be obtained using the following function:

```
cl_int clIcdGetPlatformIDsKHR(
    cl_uint num_entries,
    cl_platform_id* platforms,
    cl_uint* num_platforms);
```

num_entries is the number of `cl_platform_id` entries that can be added to *platforms*. If *platforms* is not `NULL`, then *num_entries* must be greater than zero.

platforms returns a list of OpenCL platforms available for access through the Khronos ICD Loader. The `cl_platform_id` values returned in *platforms* are ICD compatible and can be used to identify a specific OpenCL platform. If the *platforms* argument is `NULL`, then this argument is ignored. The number of OpenCL platforms returned is the minimum of the value specified by *num_entries* or the number of OpenCL platforms available.

num_platforms returns the number of OpenCL platforms available. If *num_platforms* is `NULL`, then this argument is ignored.

clIcdGetPlatformIDsKHR returns `CL_SUCCESS` if the function is executed successfully and there are a non zero number of platforms available. It returns `CL_PLATFORM_NOT_FOUND_KHR` if zero platforms are available. It returns `CL_INVALID_VALUE` if *num_entries* is equal to zero and *platforms* is not `NULL` or if both *num_platforms* and *platforms* are `NULL`."

Add the following to *table 4.1*:

Platform Info	Return Type	Description
`CL_PLATFORM_ICD_SUFFIX_KHR`	`char`[]	The function name suffix used to identify extension functions to be directed to this platform by the ICD Loader.

2.12. Source Code

The official source for the ICD Loader is available on github, at:

https://github.com/KhronosGroup/OpenCL-ICD-Loader

The complete `_cl_icd_dispatch` structure is defined in the header **cl_icd.h**, which is available as a part of the OpenCL headers.

2.13. Issues

1. Some OpenCL functions do not take an object argument from which their vendor library may be identified (e.g, **clUnloadCompiler**), how will they be handled?

   RESOLVED: Such functions will be a noop for all calls through the ICD Loader.

2. How are OpenCL extension to be handled?

   RESOLVED: Extension APIs must be queried using **clGetExtensionFunctionAddressForPlatform**.

3. How will the ICD Loader handle a `NULL` `cl_platform_id`?

   RESOLVED: The ICD will by default choose the first enumerated platform as the `NULL` platform.

4. There exists no mechanism to unload the ICD Loader, should there be one?

   RESOLVED: As there is no standard mechanism for unloading a vendor implementation, do not add one for the ICD Loader.

5. How will the ICD Loader handle `NULL` objects passed to the OpenCL functions?

   RESOLVED: The ICD Loader will check for `NULL` objects passed to the OpenCL functions without trying to dereference the `NULL` objects for obtaining the ICD dispatch table. On detecting a `NULL` object it will return one of the an invalid object error values (e.g. `CL_INVALID_DEVICE` corresponding to the object in question.

Chapter 3. Byte Addressable Stores

This section describes the **cl_khr_byte_addressable_store** extension. This extension relaxes restrictions on pointers to `char`, `uchar`, `char2`, `uchar2`, `short`, `ushort` and `half` that were present in *Section 6.8m: Restrictions* of the OpenCL 1.0 specification. With this extension, applications are able to read from and write to pointers to these types.

This extension became a core feature in OpenCL 1.1.

3.1. General Information

3.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

Chapter 4. Writing to 3D Image Objects

This section describes the **cl_khr_3d_image_writes** extension.

This extension adds built-in functions that allow a kernel to write to 3D image objects in addition to 2D image objects.

This extension became a core feature in OpenCL 2.0.

4.1. General Information

4.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

The new built-in functions are described in the table below:

Table 1. 3D Image Built-in Image Write Functions

Function	Description
void **write_imagef** (image3d_t *image*, int4 *coord*, float4 *color*) void **write_imagei** (image3d_t *image*, int4 *coord*, int4 *color*) void **write_imageui** (image3d_t *image*, int4 *coord*, uint4 *color*)	Write *color* value to the location specified by coordinate (*x*, *y*, *z*) in the 3D image specified by *image*. The appropriate data format conversion to the specified image format is done before writing the color value. *coord.x*, *coord.y*, and *coord.z* are considered to be unnormalized coordinates and must be in the range 0 … image width - 1, 0 … image height - 1, and 0 … image depth - 1. **write_imagef** can only be used with image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or set to `CL_SNORM_INT8`, `CL_UNORM_INT8`, `CL_SNORM_INT16`, `CL_UNORM_INT16`, `CL_HALF_FLOAT`, or `CL_FLOAT`. Appropriate data format conversion will be done to convert the channel data from a floating-point value to the actual data format in which the channels are stored. **write_imagei** can only be used with image objects created with *image_channel_data_type* set to one of the following values: `CL_SIGNED_INT8`, `CL_SIGNED_INT16`, or `CL_SIGNED_INT32`. **write_imageui** can only be used with image objects created with *image_channel_data_type* set to one of the following values: `CL_UNSIGNED_INT8`, `CL_UNSIGNED_INT16`, or `CL_UNSIGNED_INT32`. The behavior of **write_imagef**, **write_imagei**, and **write_imageui** for image objects created with *image_channel_data_type* values not specified in the description above, or with (*x*, *y*, *z*) coordinate values that are not in the range (0 … image width - 1, 0 … image height - 1, 0 … image depth - 1) respectively, is undefined.

Chapter 5. Half Precision Floating-Point

This section describes the **cl_khr_fp16** extension. This extension adds support for half scalar and vector types as built-in types that can be used for arithmetic operations, conversions etc.

5.1. General Information

5.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

5.2. Additions to Chapter 6 of the OpenCL 2.0 C Specification

The list of built-in scalar, and vector data types defined in *tables 6.1*, and *6.2* are extended to include the following:

Type	Description
half2	A 2-component half-precision floating-point vector.
half3	A 3-component half-precision floating-point vector.
half4	A 4-component half-precision floating-point vector.
half8	A 8-component half-precision floating-point vector.
half16	A 16-component half-precision floating-point vector.

The built-in vector data types for `halfn` are also declared as appropriate types in the OpenCL API (and header files) that can be used by an application. The following table describes the built-in vector data types for `halfn` as defined in the OpenCL C programming language and the corresponding data type available to the application:

Type in OpenCL Language	API type for application
half2	**cl_half2**
half3	**cl_half3**
half4	**cl_half4**
half8	**cl_half8**
half16	**cl_half16**

The relational, equality, logical and logical unary operators described in *section 6.3* can be used with `half` scalar and `halfn` vector types and shall produce a scalar `int` and vector `shortn` result respectively.

The OpenCL compiler accepts an h and H suffix on floating-point literals, indicating the literal is typed as a half.

5.2.1. Conversions

The implicit conversion rules specified in *section 6.2.1* now include the `half` scalar and `halfn` vector data types.

The explicit casts described in *section 6.2.2* are extended to take a `half` scalar data type and a `halfn` vector data type.

The explicit conversion functions described in *section 6.2.3* are extended to take a `half` scalar data type and a `halfn` vector data type.

The `as_typen()` function for re-interpreting types as described in *section 6.2.4.2* is extended to allow conversion-free casts between `shortn`, `ushortn`, and `halfn` scalar and vector data types.

5.2.2. Math Functions

The built-in math functions defined in *table 6.8* (also listed below) are extended to include appropriate versions of functions that take `half` and `half{2|3|4|8|16}` as arguments and return values. `gentype` now also includes `half`, `half2`, `half3`, `half4`, `half8`, and `half16`.

For any specific use of a function, the actual type has to be the same for all arguments and the return type.

Table 2. Half Precision Built-in Math Functions

Function	Description
gentype **acos** (gentype *x*)	Arc cosine function.
gentype **acosh** (gentype *x*)	Inverse hyperbolic cosine.
gentype **acospi** (gentype *x*)	Compute **acos** (*x*) / π.
gentype **asin** (gentype *x*)	Arc sine function.
gentype **asinh** (gentype *x*)	Inverse hyperbolic sine.
gentype **asinpi** (gentype *x*)	Compute **asin** (*x*) / π.
gentype **atan** (gentype *y_over_x*)	Arc tangent function.
gentype **atan2** (gentype *y*, gentype *x*)	Arc tangent of *y* / *x*.
gentype **atanh** (gentype *x*)	Hyperbolic arc tangent.
gentype **atanpi** (gentype *x*)	Compute **atan** (*x*) / π.
gentype **atan2pi** (gentype *y*, gentype *x*)	Compute **atan2** (*y*, *x*) / π.
gentype **cbrt** (gentype *x*)	Compute cube-root.
gentype **ceil** (gentype *x*)	Round to integral value using the round to positive infinity rounding mode.
gentype **copysign** (gentype *x*, gentype *y*)	Returns *x* with its sign changed to match the sign of *y*.

Function	Description
gentype **cos** (gentype *x*)	Compute cosine.
gentype **cosh** (gentype *x*)	Compute hyperbolic cosine.
gentype **cospi** (gentype *x*)	Compute **cos** (π *x*).
gentype **erfc** (gentype *x*)	Complementary error function.
gentype **erf** (gentype *x*)	Error function encountered in integrating the normal distribution.
gentype **exp** (gentype *x*)	Compute the base- e exponential of *x*.
gentype **exp2** (gentype *x*)	Exponential base 2 function.
gentype **exp10** (gentype *x*)	Exponential base 10 function.
gentype **expm1** (gentype *x*)	Compute e^{x}- 1.0.
gentype **fabs** (gentype *x*)	Compute absolute value of a floating-point number.
gentype **fdim** (gentype *x*, gentype *y*)	*x* - *y* if *x* > *y*, +0 if x is less than or equal to y.
gentype **floor** (gentype *x*)	Round to integral value using the round to negative infinity rounding mode.
gentype **fma** (gentype *a*, gentype *b*, gentype *c*)	Returns the correctly rounded floating-point representation of the sum of *c* with the infinitely precise product of *a* and *b*. Rounding of intermediate products shall not occur. Edge case behavior is per the IEEE 754-2008 standard.
gentype **fmax** (gentype *x*, gentype *y*) gentype **fmax** (gentype *x*, half *y*)	Returns *y* if *x* < *y*, otherwise it returns *x*. If one argument is a NaN, **fmax()** returns the other argument. If both arguments are NaNs, **fmax()** returns a NaN.
gentype **fmin** (gentype *x*, gentype *y*) gentype **fmin** (gentype *x*, half *y*)	Returns *y* if *y* < *x*, otherwise it returns *x*. If one argument is a NaN, **fmin()** returns the other argument. If both arguments are NaNs, **fmin()** returns a NaN.
gentype **fmod** (gentype *x*, gentype *y*)	Modulus. Returns *x* - *y* * **trunc** (*x*/*y*) .

Function	Description
gentype **fract** (gentype *x*, __global gentype **iptr*) gentype **fract** (gentype *x*, __local gentype **iptr*) gentype **fract** (gentype *x*, __private gentype * *iptr*) For OpenCL C 2.0 or with the `__opencl_c_generic_address_space` feature macro: gentype **fract** (gentype *x*, gentype **iptr*)	Returns **fmin**(*x* - **floor** (*x*), 0x1.ffcp-1f). **floor**(x) is returned in *iptr*.
half*n* **frexp** (half*n* *x*, __global int*n* *exp) half **frexp** (half *x*, __global int *exp) half*n* **frexp** (half*n* *x*, __local int*n* *exp) half **frexp** (half *x*, __local int *exp) half*n* **frexp** (half*n* *x*, __private int*n* *exp) half **frexp** (half *x*, __private int *exp) For OpenCL C 2.0 or with the `__opencl_c_generic_address_space` feature macro: half*n* **frexp** (half*n* *x*, int*n* *exp) half **frexp** (half *x*, int *exp)	Extract mantissa and exponent from *x*. For each component the mantissa returned is a `half` with magnitude in the interval [1/2, 1) or 0. Each component of *x* equals mantissa returned * 2^{exp}.
gentype **hypot** (gentype *x*, gentype *y*)	Compute the value of the square root of $x^2 + y^2$ without undue overflow or underflow.
int*n* **ilogb** (half*n* *x*) int **ilogb** (half *x*)	Return the exponent as an integer value.
half*n* **ldexp** (half*n* *x*, int*n* *k*) half*n* **ldexp** (half*n* *x*, int *k*) half **ldexp** (half *x*, int *k*)	Multiply *x* by 2 to the power *k*.

Function	Description
gentype **lgamma** (gentype *x*) halfn **lgamma_r** (halfn *x*, __global intn *signp) half **lgamma_r** (half *x*, __global int *signp) halfn **lgamma_r** (halfn *x*, __local intn *signp) half **lgamma_r** (half *x*, __local int *signp) halfn **lgamma_r** (halfn *x*, __private intn *signp) half **lgamma_r** (half *x*, __private int *signp) For OpenCL C 2.0 or with the `__opencl_c_generic_address_space` feature macro: halfn **lgamma_r** (halfn *x*, intn *signp) half **lgamma_r** (half *x*, int *signp)	Log gamma function. Returns the natural logarithm of the absolute value of the gamma function. The sign of the gamma function is returned in the *signp* argument of **lgamma_r**.
gentype **log** (gentype *x*)	Compute natural logarithm.
gentype **log2** (gentype *x*)	Compute a base 2 logarithm.
gentype **log10** (gentype *x*)	Compute a base 10 logarithm.
gentype **log1p** (gentype *x*)	Compute $\log_e(1.0 + x)$.
gentype **logb** (gentype *x*)	Compute the exponent of *x*, which is the integral part of $\log_r\lvert x\rvert$.
gentype **mad** (gentype *a*, gentype *b*, gentype *c*)	**mad** computes $a * b + c$. The function may compute $a * b + c$ with reduced accuracy in the embedded profile. See the OpenCL SPIR-V Environment Specification for details. On some hardware the mad instruction may provide better performance than expanded computation of $a * b + c$. Note: For some usages, e.g. **mad**(a, b, -a*b), the half precision definition of **mad**() is loose enough that almost any result is allowed from **mad**() for some values of a and b.
gentype **maxmag** (gentype *x*, gentype *y*)	Returns *x* if $\lvert x\rvert > \lvert y\rvert$, *y* if $\lvert y\rvert > \lvert x\rvert$, otherwise **fmax**(*x*, *y*).

Function	Description
gentype **minmag** (gentype *x*, gentype *y*)	Returns *x* if |*x*| < |*y*|, *y* if |*y*| < |*x*|, otherwise **fmin**(*x*, *y*).
gentype **modf** (gentype *x*, __global gentype **iptr*) gentype **modf** (gentype *x*, __local gentype **iptr*) gentype **modf** (gentype *x*, __private gentype **iptr*) For OpenCL C 2.0 or with the `__opencl_c_generic_address_space` feature macro: gentype **modf** (gentype *x*, gentype **iptr*)	Decompose a floating-point number. The **modf** function breaks the argument *x* into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part in the object pointed to by *iptr*.
half*n* **nan** (ushort*n* *nancode*) half **nan** (ushort *nancode*)	Returns a quiet NaN. The *nancode* may be placed in the significand of the resulting NaN.
gentype **nextafter** (gentype *x*, gentype *y*)	Computes the next representable half-precision floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, **nextafter**() returns the largest representable floating-point number less than *x*.
gentype **pow** (gentype *x*, gentype *y*)	Compute *x* to the power *y*.
half*n* **pown** (half*n* *x*, int*n* *y*) half **pown** (half *x*, int *y*)	Compute *x* to the power *y*, where *y* is an integer.
gentype **powr** (gentype *x*, gentype *y*)	Compute *x* to the power *y*, where *x* is >= 0.
gentype **remainder** (gentype *x*, gentype *y*)	Compute the value *r* such that *r* = *x* - *n***y*, where *n* is the integer nearest the exact value of *x*/*y*. If there are two integers closest to *x*/*y*, *n* shall be the even one. If *r* is zero, it is given the same sign as *x*.

Function	Description
half*n* **remquo** (half*n* *x*, half*n* *y*, __global int*n* **quo*) half **remquo** (half *x*, half *y*, __global int **quo*) half*n* **remquo** (half*n* *x*, half*n* *y*, __local int*n* **quo*) half **remquo** (half *x*, half *y*, __local int **quo*) half*n* **remquo** (half*n* *x*, half*n* *y*, __private int*n* **quo*) half **remquo** (half *x*, half *y*, __private int **quo*) For OpenCL C 2.0 or with the `__opencl_c_generic_address_space` feature macro: half*n* **remquo** (half*n* *x*, half*n* *y*, int*n* **quo*) half **remquo** (half *x*, half *y*, int **quo*)	The **remquo** function computes the value r such that *r* = *x* - *k***y*, where *k* is the integer nearest the exact value of *x*/*y*. If there are two integers closest to *x*/*y*, *k* shall be the even one. If *r* is zero, it is given the same sign as *x*. This is the same value that is returned by the **remainder** function. **remquo** also calculates the lower seven bits of the integral quotient *x*/*y*, and gives that value the same sign as *x*/*y*. It stores this signed value in the object pointed to by *quo*.
gentype **rint** (gentype *x*)	Round to integral value (using round to nearest even rounding mode) in floating-point format. Refer to section 7.1 for description of rounding modes.
half*n* **rootn** (half*n* *x*, int*n* *y*) half **rootn** (half *x*, int *y*)	Compute *x* to the power 1/*y*.
gentype **round** (gentype *x*)	Return the integral value nearest to *x* rounding halfway cases away from zero, regardless of the current rounding direction.
gentype **rsqrt** (gentype *x*)	Compute inverse square root.
gentype **sin** (gentype *x*)	Compute sine.

Function	Description
gentype **sincos** (gentype *x*, __global gentype **cosval*) gentype **sincos** (gentype *x*, __local gentype **cosval*) gentype **sincos** (gentype *x*, __private gentype **cosval*) For OpenCL C 2.0 or with the `__opencl_c_generic_address_space` feature macro: gentype **sincos** (gentype *x*, gentype **cosval*)	Compute sine and cosine of x. The computed sine is the return value and computed cosine is returned in *cosval*.
gentype **sinh** (gentype *x*)	Compute hyperbolic sine.
gentype **sinpi** (gentype *x*)	Compute **sin** (π *x*).
gentype **sqrt** (gentype *x*)	Compute square root.
gentype **tan** (gentype *x*)	Compute tangent.
gentype **tanh** (gentype *x*)	Compute hyperbolic tangent.
gentype **tanpi** (gentype *x*)	Compute **tan** (π *x*).
gentype **tgamma** (gentype *x*)	Compute the gamma function.
gentype **trunc** (gentype *x*)	Round to integral value using the round to zero rounding mode.

The **FP_FAST_FMA_HALF** macro indicates whether the **fma()** family of functions are fast compared with direct code for half precision floating-point. If defined, the **FP_FAST_FMA_HALF** macro shall indicate that the **fma()** function generally executes about as fast as, or faster than, a multiply and an add of **half** operands.

The macro names given in the following list must use the values specified. These constant expressions are suitable for use in #if preprocessing directives.

```
#define HALF_DIG            3
#define HALF_MANT_DIG       11
#define HALF_MAX_10_EXP     +4
#define HALF_MAX_EXP        +16
#define HALF_MIN_10_EXP     -4
#define HALF_MIN_EXP        -13
#define HALF_RADIX          2
#define HALF_MAX            0x1.ffcp15h
#define HALF_MIN            0x1.0p-14h
#define HALF_EPSILON        0x1.0p-10h
```

The following table describes the built-in macro names given above in the OpenCL C programming language and the corresponding macro names available to the application.

Macro in OpenCL Language	Macro for application
HALF_DIG	CL_HALF_DIG
HALF_MANT_DIG	CL_HALF_MANT_DIG
HALF_MAX_10_EXP	CL_HALF_MAX_10_EXP
HALF_MAX_EXP	CL_HALF_MAX_EXP
HALF_MIN_10_EXP	CL_HALF_MIN_10_EXP
HALF_MIN_EXP	CL_HALF_MIN_EXP
HALF_RADIX	CL_HALF_RADIX
HALF_MAX	CL_HALF_MAX
HALF_MIN	CL_HALF_MIN
HALF_EPSILSON	CL_HALF_EPSILON

The following constants are also available. They are of type half and are accurate within the precision of the half type.

Constant	Description
M_E_H	Value of e
M_LOG2E_H	Value of $\log_2 e$
M_LOG10E_H	Value of $\log_{10} e$
M_LN2_H	Value of $\log_e 2$
M_LN10_H	Value of $\log_e 10$
M_PI_H	Value of π
M_PI_2_H	Value of π / 2
M_PI_4_H	Value of π / 4
M_1_PI_H	Value of 1 / π
M_2_PI_H	Value of 2 / π
M_2_SQRTPI_H	Value of 2 / √π
M_SQRT2_H	Value of √2
M_SQRT1_2_H	Value of 1 / √2

5.2.3. Common Functions

The built-in common functions defined in *table 6.12* (also listed below) are extended to include appropriate versions of functions that take half and half{2|3|4|8|16} as arguments and return values. gentype now also includes half, half2, half3, half4, half8 and half16. These are described below.

Table 3. Half Precision Built-in Common Functions

Function	Description
gentype **clamp** (gentype *x*, gentype *minval*, gentype *maxval*) gentype **clamp** (gentype *x*, half *minval*, half *maxval*)	Returns **fmin**(**fmax**(*x*, *minval*), *maxval*). Results are undefined if *minval* > *maxval*.
gentype **degrees** (gentype *radians*)	Converts *radians* to degrees, i.e. (180 / π) * *radians*.
gentype **max** (gentype *x*, gentype *y*) gentype **max** (gentype *x*, half *y*)	Returns *y* if *x* < *y*, otherwise it returns *x*. If *x* and *y* are infinite or NaN, the return values are undefined.
gentype **min** (gentype *x*, gentype *y*) gentype **min** (gentype *x*, half *y*)	Returns *y* if *y* < *x*, otherwise it returns *x*. If *x* and *y* are infinite or NaN, the return values are undefined.
gentype **mix** (gentype *x*, gentype *y*, gentype *a*) gentype **mix** (gentype *x*, gentype *y*, half *a*)	Returns the linear blend of *x* and *y* implemented as: *x* + (*y* - *x)* * *a* *a* must be a value in the range 0.0 … 1.0. If *a* is not in the range 0.0 … 1.0, the return values are undefined. Note: The half precision **mix** function can be implemented using contractions such as **mad** or **fma**.
gentype **radians** (gentype *degrees*)	Converts *degrees* to radians, i.e. (π / 180) * *degrees*.
gentype **step** (gentype *edge*, gentype *x*) gentype **step** (half *edge*, gentype *x*)	Returns 0.0 if *x* < *edge*, otherwise it returns 1.0.

Function	Description
gentype **smoothstep** (gentype *edge0*, gentype *edge1*, gentype *x*) gentype **smoothstep** (half *edge0*, half *edge1*, gentype *x*)	Returns 0.0 if *x* <= *edge0* and 1.0 if *x* >= *edge1* and performs smooth Hermite interpolation between 0 and 1 when *edge0* < *x* < *edge1*. This is useful in cases where you would want a threshold function with a smooth transition. This is equivalent to: gentype *t*; *t* = clamp ((*x* - *edge0*) / (*edge1* - *edge0*), 0, 1); return *t* * *t* * (3 - 2 * *t*); Results are undefined if *edge0* >= *edge1*. Note: The half precision **smoothstep** function can be implemented using contractions such as **mad** or **fma**.
gentype **sign** (gentype *x*)	Returns 1.0 if *x* > 0, -0.0 if *x* = -0.0, +0.0 if *x* = +0.0, or -1.0 if *x* < 0. Returns 0.0 if *x* is a NaN.

5.2.4. Geometric Functions

The built-in geometric functions defined in *table 6.13* (also listed below) are extended to include appropriate versions of functions that take `half` and `half{2|3|4}` as arguments and return values. gentype now also includes `half`, `half2`, `half3` and `half4`. These are described below.

Note: The half precision geometric functions can be implemented using contractions such as **mad** or **fma**.

Table 4. Half Precision Built-in Geometric Functions

Function	Description
half4 **cross** (half4 *p0*, half4 *p1*) half3 **cross** (half3 *p0*, half3 *p1*)	Returns the cross product of *p0.xyz* and *p1.xyz*. The *w* component of the result will be 0.0.
half **dot** (gentype *p0*, gentype *p1*)	Compute the dot product of *p0* and *p1*.
half **distance** (gentype *p0*, gentype *p1*)	Returns the distance between *p0* and *p1*. This is calculated as **length**(*p0* - *p1*).
half **length** (gentype *p*)	Return the length of vector x, i.e., sqrt($p.x^2 + p.y^2 + \ldots$)
gentype **normalize** (gentype *p*)	Returns a vector in the same direction as *p* but with a length of 1.

5.2.5. Relational Functions

The scalar and vector relational functions described in *table 6.14* are extended to include versions that take `half`, `half2`, `half3`, `half4`, `half8` and `half16` as arguments.

The relational and equality operators (<, <=, >, >=, !=, ==) can be used with `halfn` vector types and shall produce a vector `shortn` result as described in *section 6.3*.

The functions **isequal**, **isnotequal**, **isgreater**, **isgreaterequal**, **isless**, **islessequal**, **islessgreater**, **isfinite**, **isinf**, **isnan**, **isnormal**, **isordered**, **isunordered** and **signbit** shall return a 0 if the specified relation is *false* and a 1 if the specified relation is true for scalar argument types. These functions shall return a 0 if the specified relation is *false* and a -1 (i.e. all bits set) if the specified relation is *true* for vector argument types.

The relational functions **isequal**, **isgreater**, **isgreaterequal**, **isless**, **islessequal**, and **islessgreater** always return 0 if either argument is not a number (NaN). **isnotequal** returns 1 if one or both arguments are not a number (NaN) and the argument type is a scalar and returns -1 if one or both arguments are not a number (NaN) and the argument type is a vector.

The functions described in *table 6.14* are extended to include the `halfn` vector types.

Table 5. Half Precision Relational Functions

Function	**Description**
int **isequal** (half *x*, half *y*) short*n* **isequal** (half*n* *x*, half*n* *y*)	Returns the component-wise compare of *x* == *y*.
int **isnotequal** (half *x*, half *y*) short*n* **isnotequal** (half*n* *x*, half*n* *y*)	Returns the component-wise compare of *x* != *y*.
int **isgreater** (half *x*, half *y*) short*n* **isgreater** (half*n* *x*, half*n* *y*)	Returns the component-wise compare of *x* > *y*.
int **isgreaterequal** (half *x*, half *y*) short*n* **isgreaterequal** (half*n* *x*, half*n* *y*)	Returns the component-wise compare of *x* >= *y*.
int **isless** (half *x*, half *y*) short*n* **isless** (half*n* *x*, half*n* *y*)	Returns the component-wise compare of *x* < *y*.
int **islessequal** (half *x*, half *y*) short*n* **islessequal** (half*n* *x*, half*n* *y*)	Returns the component-wise compare of *x* <= *y*.
int **islessgreater** (half *x*, half *y*) short*n* **islessgreater** (half*n* *x*, half*n* *y*)	Returns the component-wise compare of (*x* < *y*) || (*x* > *y*) .
int **isfinite** (half) short*n* **isfinite** (half*n*)	Test for finite value.
int **isinf** (half) short*n* **isinf** (half*n*)	Test for infinity value (positive or negative) .
int **isnan** (half) short*n* **isnan** (half*n*)	Test for a NaN.

Function	Description
int **isnormal** (half) short*n* **isnormal** (half*n*)	Test for a normal value.
int **isordered** (half *x*, half *y*) short*n* **isordered** (half*n* *x*, half*n* *y*)	Test if arguments are ordered. **isordered**() takes arguments *x* and *y*, and returns the result **isequal**(*x*, *x*) && **isequal**(*y*, *y*).
int **isunordered** (half *x*, half *y*) short*n* **isunordered** (half*n* *x*, half*n* *y*)	Test if arguments are unordered. **isunordered**() takes arguments *x* and *y*, returning non-zero if *x* or *y* is a NaN, and zero otherwise.
int **signbit** (half) short*n* **signbit** (half*n*)	Test for sign bit. The scalar version of the function returns a 1 if the sign bit in the half is set else returns 0. The vector version of the function returns the following for each component in half*n*: -1 (i.e all bits set) if the sign bit in the half is set else returns 0.
half*n* **bitselect** (half*n* *a*, half*n* *b*, half*n* *c*)	Each bit of the result is the corresponding bit of *a* if the corresponding bit of *c* is 0. Otherwise it is the corresponding bit of *b*.
half*n* **select** (half*n* *a*, half*n* *b*, short*n* *c*) half*n* **select** (half*n* *a*, half*n* *b*, ushort*n* *c*)	For each component, *result[i]* = if MSB of *c[i]* is set ? *b[i]* : *a[i]*.

5.2.6. Vector Data Load and Store Functions

The vector data load (**vload*n***) and store (**vstore*n***) functions described in *table 6.13* (also listed below) are extended to include versions that read or write half vector values. The generic type `gentype` is extended to include `half`. The generic type `gentypen` is extended to include `half2`, `half3`, `half4`, `half8`, and `half16`.

Note: **vload3** reads *x*, *y*, *z* components from address (*p* + (*offset* * 3)) into a 3-component vector and **vstore3** writes *x*, *y*, *z* components from a 3-component vector to address (*p* + (*offset* * 3)).

Table 6. Half Precision Vector Data Load and Store Functions

Function	Description
gentype*n* **vload*n***(size_t *offset*, const __global gentype **p*) gentype*n* **vload*n***(size_t *offset*, const __local gentype **p*) gentype*n* **vload*n***(size_t *offset*, const __constant gentype **p*) gentype*n* **vload*n***(size_t *offset*, const __private gentype **p*) For OpenCL C 2.0 or with the `__opencl_c_generic_address_space` feature macro: gentype*n* **vload*n***(size_t *offset*, const gentype **p*)	Return sizeof (gentype*n*) bytes of data read from address (*p* + (*offset* * *n*)). If gentype is half, the read address computed as (*p* + (*offset* * *n*)) must be 16-bit aligned.
void **vstore*n***(gentype*n* *data*, size_t *offset*, __global gentype **p*) void **vstore*n***(gentype*n* *data*, size_t *offset*, __local gentype **p*) void **vstore*n***(gentype*n* *data*, size_t *offset*, __private gentype **p*) For OpenCL C 2.0 or with the `__opencl_c_generic_address_space` feature macro: void **vstore*n***(gentype*n* *data*, size_t *offset*, gentype **p*)	Write sizeof (gentype*n*) bytes given by *data* to address (*p* + (*offset* * *n*)). If gentype is half, the write address computed as (*p* + (*offset* * *n*)) must be 16-bit aligned.

5.2.7. Async Copies from Global to Local Memory, Local to Global Memory, and Prefetch

The OpenCL C programming language implements the following functions that provide asynchronous copies between global and local memory and a prefetch from global memory.

The generic type `gentype` is extended to include `half`, `half2`, `half3`, `half4`, `half8`, and `half16`.

Table 7. Half Precision Built-in Async Copy and Prefetch Functions

Function	Description
event_t **async_work_group_copy** (__local gentype **dst*, const __global gentype **src*, size_t *num_gentypes*, event_t *event*) event_t **async_work_group_copy** (__global gentype **dst*, const __local gentype **src*, size_t *num_gentypes*, event_t *event*)	Perform an async copy of *num_gentypes* gentype elements from *src* to *dst*. The async copy is performed by all work-items in a work-group and this built-in function must therefore be encountered by all work-items in a work-group executing the kernel with the same argument values; otherwise the results are undefined. Returns an event object that can be used by **wait_group_events** to wait for the async copy to finish. The *event* argument can also be used to associate the **async_work_group_copy** with a previous async copy allowing an event to be shared by multiple async copies; otherwise *event* should be zero. If *event* argument is not zero, the event object supplied in *event* argument will be returned. This function does not perform any implicit synchronization of source data such as using a **barrier** before performing the copy.

Function	Description
event_t **async_work_group_strided_copy** (__local gentype **dst*, const __global gentype **src*, size_t *num_gentypes*, size_t *src_stride*, event_t *event*) event_t **async_work_group_strided_copy** (__global gentype **dst*, const __local gentype **src*, size_t *num_gentypes*, size_t *dst_stride*, event_t *event*)	Perform an async gather of *num_gentypes* gentype elements from *src* to *dst*. The *src_stride* is the stride in elements for each gentype element read from *src*. The async gather is performed by all work-items in a work-group and this built-in function must therefore be encountered by all work-items in a work-group executing the kernel with the same argument values; otherwise the results are undefined. Returns an event object that can be used by **wait_group_events** to wait for the async copy to finish. The *event* argument can also be used to associate the **async_work_group_strided_copy** with a previous async copy allowing an event to be shared by multiple async copies; otherwise *event* should be zero. If *event* argument is not zero, the event object supplied in *event* argument will be returned. This function does not perform any implicit synchronization of source data such as using a **barrier** before performing the copy. The behavior of **async_work_group_strided_copy** is undefined if *src_stride* or *dst_stride* is 0, or if the *src_stride* or *dst_stride* values cause the *src* or *dst* pointers to exceed the upper bounds of the address space during the copy.
void **wait_group_events** (int *num_events*, event_t **event_list*)	Wait for events that identify the **async_work_group_copy** operations to complete. The event objects specified in *event_list* will be released after the wait is performed. This function must be encountered by all work-items in a work-group executing the kernel with the same *num_events* and event objects specified in *event_list*; otherwise the results are undefined.

Function	Description
void **prefetch** (const __global gentype **p*, size_t *num_gentypes*)	Prefetch *num_gentypes* * sizeof(gentype) bytes into the global cache. The prefetch instruction is applied to a work-item in a work-group and does not affect the functional behavior of the kernel.

5.2.8. Image Read and Write Functions

The image read and write functions defined in *tables 6.23*, *6.24* and *6.25* are extended to support image color values that are a `half` type.

5.2.9. Built-in Image Read Functions

Table 8. Half Precision Built-in Image Read Functions

Function	Description
half4 **read_imageh** (read_only image2d_t *image*, sampler_t *sampler*, int2 *coord*) half4 **read_imageh** (read_only image2d_t *image*, sampler_t *sampler*, float2 *coord*)	Use the coordinate *(coord.x, coord.y)* to do an element lookup in the 2D image object specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats, `CL_UNORM_INT8`, or `CL_UNORM_INT16`. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to `CL_SNORM_INT8`, or `CL_SNORM_INT16`. **read_imageh** returns half precision floating-point values for image objects created with *image_channel_data_type* set to `CL_HALF_FLOAT`. The **read_imageh** calls that take integer coordinates must use a sampler with filter mode set to `CLK_FILTER_NEAREST`, normalized coordinates set to `CLK_NORMALIZED_COORDS_FALSE` and addressing mode set to `CLK_ADDRESS_CLAMP_TO_EDGE`, `CLK_ADDRESS_CLAMP` or `CLK_ADDRESS_NONE`; otherwise the values returned are undefined. Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description above are undefined.

Function	Description
half4 **read_imageh** (read_only image3d_t *image*, sampler_t *sampler*, int4 *coord*) half4 **read_imageh** (read_only image3d_t *image*, sampler_t *sampler*, float4 *coord*)	Use the coordinate *(coord.x, coord.y, coord.z)* to do an elementlookup in the 3D image object specified by *image*. *coord.w* is ignored. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or `CL_UNORM_INT8`, or `CL_UNORM_INT16`. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to `CL_SNORM_INT8`, or `CL_SNORM_INT16`. **read_imageh**returns half precision floating-point values for image objects created with *image_channel_data_type* set to `CL_HALF_FLOAT`. The **read_imageh** calls that take integer coordinates must use a sampler with filter mode set to `CLK_FILTER_NEAREST`, normalized coordinates set to `CLK_NORMALIZED_COORDS_FALSE` and addressing mode set to `CLK_ADDRESS_CLAMP_TO_EDGE`, `CLK_ADDRESS_CLAMP` or `CLK_ADDRESS_NONE`; otherwise the values returned are undefined. Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description are undefined.

Function	Description
half4 **read_imageh** (read_only image2d_array_t *image*, sampler_t *sampler*, int4 *coord*) half4 **read_imageh** (read_only image2d_array_t *image*, sampler_t *sampler*, float4 *coord*)	Use *coord.xy* to do an element lookup in the 2D image identified by *coord.z* in the 2D image array specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with image_channel_data_type set to one of the pre-defined packed formats or `CL_UNORM_INT8`, or `CL_UNORM_INT16`. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with image_channel_data_type set to `CL_SNORM_INT8`, or `CL_SNORM_INT16`. **read_imageh** returns half precision floating-point values for image objects created with image_channel_data_type set to `CL_HALF_FLOAT`. The **read_imageh** calls that take integer coordinates must use a sampler with filter mode set to `CLK_FILTER_NEAREST`, normalized coordinates set to `CLK_NORMALIZED_COORDS_FALSE` and addressing mode set to `CLK_ADDRESS_CLAMP_TO_EDGE`, `CLK_ADDRESS_CLAMP` or `CLK_ADDRESS_NONE`; otherwise the values returned are undefined. Values returned by **read_imageh** for image objects with image_channel_data_type values not specified in the description above are undefined.

Function	Description
half4 **read_imageh** (read_only image1d_t *image*, sampler_t *sampler*, int *coord*) half4 **read_imageh** (read_only image1d_t *image*, sampler_t *sampler*, float *coord*)	Use *coord* to do an element lookup in the 1D image object specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or `CL_UNORM_INT8`, or `CL_UNORM_INT16`. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to `CL_SNORM_INT8`, or `CL_SNORM_INT16`. **read_imageh** returns half precision floating-point values for image objects created with *image_channel_data_type* set to `CL_HALF_FLOAT`. The **read_imageh** calls that take integer coordinates must use a sampler with filter mode set to `CLK_FILTER_NEAREST`, normalized coordinates set to `CLK_NORMALIZED_COORDS_FALSE` and addressing mode set to `CLK_ADDRESS_CLAMP_TO_EDGE`, `CLK_ADDRESS_CLAMP` or `CLK_ADDRESS_NONE`; otherwise the values returned are undefined. Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description above are undefined.

Function	Description
half4 **read_imageh** (read_only image1d_array_t *image*, sampler_t *sampler*, int2 *coord*) half4 **read_imageh** (read_only image1d_array_t *image*, sampler_t *sampler*, float2 *coord*)	Use *coord.x* to do an element lookup in the 1D image identified by *coord.y* in the 1D image array specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with image_channel_data_type set to one of the pre-defined packed formats or `CL_UNORM_INT8`, or `CL_UNORM_INT16`. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with image_channel_data_type set to `CL_SNORM_INT8`, or `CL_SNORM_INT16`. **read_imageh** returns half precision floating-point values for image objects created with image_channel_data_type set to `CL_HALF_FLOAT`. The **read_imageh** calls that take integer coordinates must use a sampler with filter mode set to `CLK_FILTER_NEAREST`, normalized coordinates set to `CLK_NORMALIZED_COORDS_FALSE` and addressing mode set to `CLK_ADDRESS_CLAMP_TO_EDGE`, `CLK_ADDRESS_CLAMP` or `CLK_ADDRESS_NONE`; otherwise the values returned are undefined. Values returned by **read_imageh** for image objects with image_channel_data_type values not specified in the description above are undefined.

5.2.10. Built-in Image Sampler-less Read Functions

aQual in Table 6.24 refers to one of the access qualifiers. For sampler-less read functions this may be *read_only* or *read_write*.

Table 9. Half Precision Built-in Image Sampler-less Read Functions

Function	Description
half4 **read_imageh** (*aQual* image2d_t *image*, int2 *coord*)	Use the coordinate *(coord.x, coord.y)* to do an element lookup in the 2D image object specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or `CL_UNORM_INT8`, or `CL_UNORM_INT16`. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to `CL_SNORM_INT8`, or `CL_SNORM_INT16`. **read_imageh** returns half precision floating-point values for image objects created with *image_channel_data_type* set to `CL_HALF_FLOAT`. Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description above are undefined.
half4 **read_imageh** (*aQual* image3d_t *image*, int4 *coord*)	Use the coordinate *(coord.x*, *coord.y*, *coord.z)* to do an element lookup in the 3D image object specified by *image*. *coord.w* is ignored. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or `CL_UNORM_INT8`, or `CL_UNORM_INT16`. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to `CL_SNORM_INT8`, or `CL_SNORM_INT16`. **read_imageh** returns half precision floating-point values for image objects created with *image_channel_data_type* set to `CL_HALF_FLOAT`. Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description are undefined.

Function	Description
half4 **read_imageh** (*aQual* image2d_array_t *image*, int4 *coord*)	Use *coord.xy* to do an element lookup in the 2D image identified by *coord.z* in the 2D image array specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or `CL_UNORM_INT8`, or `CL_UNORM_INT16`. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to `CL_SNORM_INT8`, or `CL_SNORM_INT16`. **read_imageh** returns half precision floating-point values for image objects created with *image_channel_data_type* set to `CL_HALF_FLOAT`. Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description above are undefined.
half4 **read_imageh** (*aQual* image1d_t *image*, int *coord*) half4 **read_imageh** (*aQual* image1d_buffer_t *image*, int *coord*)	Use *coord* to do an element lookup in the 1D image or 1D image buffer object specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or `CL_UNORM_INT8`, or `CL_UNORM_INT16`. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to `CL_SNORM_INT8`, or `CL_SNORM_INT16`. **read_imageh** returns half precision floating-point values for image objects created with *image_channel_data_type* set to `CL_HALF_FLOAT`. Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description above are undefined.

Function	Description
half4 **read_imageh** (*aQual* image1d_array_t *image*, int2 *coord*)	Use *coord.x* to do an element lookup in the 2D image identified by *coord.y* in the 2D image array specified by *image*. **read_imageh** returns half precision floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or `CL_UNORM_INT8`, or `CL_UNORM_INT16`. **read_imageh** returns half precision floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to `CL_SNORM_INT8`, or `CL_SNORM_INT16`. **read_imageh** returns half precision floating-point values for image objects created with *image_channel_data_type* set to `CL_HALF_FLOAT`. Values returned by **read_imageh** for image objects with *image_channel_data_type* values not specified in the description above are undefined.

5.2.11. Built-in Image Write Functions

aQual in Table 6.25 refers to one of the access qualifiers. For write functions this may be *write_only* or *read_write*.

Table 10. Half Precision Built-in Image Write Functions

Function	Description
void **write_imageh** (*aQual* image2d_t *image*, int2 *coord*, half4 *color*)	Write *color* value to location specified by *coord.xy* in the 2D image specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. *x* & *y* are considered to be unnormalized coordinates and must be in the range 0 … width - 1, and 0 … height - 1. **write_imageh** can only be used with image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or set to `CL_SNORM_INT8`, `CL_UNORM_INT8`, `CL_SNORM_INT16`, `CL_UNORM_INT16` or `CL_HALF_FLOAT`. The behavior of **write_imageh** for image objects created with *image_channel_data_type* values not specified in the description above or with (*x*, *y*) coordinate values that are not in the range (0 … width - 1, 0 … height - 1) respectively, is undefined.

Function	Description
void **write_imageh** (*aQual* image2d_array_t *image*, int4 *coord*, half4 *color*)	Write *color* value to location specified by *coord.xy* in the 2D image identified by *coord.z* in the 2D image array specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. *coord.x*, *coord.y* and *coord.z* are considered to be unnormalized coordinates and must be in the range 0 … image width - 1, 0 … image height - 1 and 0 … image number of layers - 1. **write_imageh** can only be used with image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or set to `CL_SNORM_INT8`, `CL_UNORM_INT8`, `CL_SNORM_INT16`, `CL_UNORM_INT16` or `CL_HALF_FLOAT`. The behavior of **write_imageh** for image objects created with *image_channel_data_type* values not specified in the description above or with (*x*, *y*, *z*) coordinate values that are not in the range (0 … image width - 1, 0 … image height - 1, 0 … image number of layers - 1), respectively, is undefined.

Function	Description
void **write_imageh** (*aQual* image1d_t *image*, int *coord*, half4 *color*) void **write_imageh** (*aQual* image1d_buffer_t *image*, int *coord*, half4 *color*)	Write *color* value to location specified by *coord* in the 1D image or 1D image buffer object specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. *coord* is considered to be unnormalized coordinates and must be in the range 0 … image width - 1. **write_imageh** can only be used with image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or set to `CL_SNORM_INT8`, `CL_UNORM_INT8`, `CL_SNORM_INT16`, `CL_UNORM_INT16` or `CL_HALF_FLOAT`. Appropriate data format conversion will be done to convert channel data from a floating-point value to actual data format in which the channels are stored. The behavior of **write_imageh** for image objects created with *image_channel_data_type* values not specified in the description above or with coordinate values that is not in the range (0 … image width - 1), is undefined.

Function	Description
void **write_imageh** (*aQual* image1d_array_t *image*, int2 *coord*, half4 *color*)	Write *color* value to location specified by *coord.x* in the 1D image identified by *coord.y* in the 1D image array specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. *coord.x* and *coord.y* are considered to be unnormalized coordinates and must be in the range 0 … image width - 1 and 0 … image number of layers - 1. **write_imageh** can only be used with image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or set to `CL_SNORM_INT8`, `CL_UNORM_INT8`, `CL_SNORM_INT16`, `CL_UNORM_INT16` or `CL_HALF_FLOAT`. Appropriate data format conversion will be done to convert channel data from a floating-point value to actual data format in which the channels are stored. The behavior of **write_imageh** for image objects created with *image_channel_data_type* values not specified in the description above or with (*x*, *y*) coordinate values that are not in the range (0 … image width - 1, 0 … image number of layers - 1), respectively, is undefined.

Function	Description
void **write_imageh** (*aQual* image3d_t *image*, int4 *coord*, half4 *color*)	Write color value to location specified by coord.xyz in the 3D image object specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. coord.x, coord.y and coord.z are considered to be unnormalized coordinates and must be in the range 0 … image width - 1, 0 … image height - 1 and 0 … image depth - 1. **write_imageh** can only be used with image objects created with image_channel_data_type set to one of the pre-defined packed formats or set to `CL_SNORM_INT8`, `CL_UNORM_INT8`, `CL_SNORM_INT16`, `CL_UNORM_INT16` or `CL_HALF_FLOAT`. The behavior of **write_imageh** for image objects created with image_channel_data_type values not specified in the description above or with (x, y, z) coordinate values that are not in the range (0 … image width - 1, 0 … image height - 1, 0 … image depth - 1), respectively, is undefined. Note: This built-in function is only available if the cl_khr_3d_image_writes extension is also supported by the device.

5.2.12. IEEE754 Compliance

The following table entry describes the additions to *table 4.3,* which allows applications to query the configuration information using **clGetDeviceInfo** for an OpenCL device that supports half precision floating-point.

Op-code	Return Type	Description
CL_DEVICE_HALF_FP_ CONFIG	cl_device_fp_config	Describes half precision floating-point capability of the OpenCL device. This is a bit-field that describes one or more of the following values: CL_FP_DENORM — denorms are supported CL_FP_INF_NAN — INF and NaNs are supported CL_FP_ROUND_TO_NEAREST — round to nearest even rounding mode supported CL_FP_ROUND_TO_ZERO — round to zero rounding mode supported CL_FP_ROUND_TO_INF — round to positive and negative infinity rounding modes supported CL_FP_FMA — IEEE754-2008 fused multiply-add is supported CL_FP_SOFT_FLOAT — Basic floating-point operations (such as addition, subtraction, multiplication) are implemented in software. The required minimum half precision floating-point capability as implemented by this extension is: CL_FP_ROUND_TO_ZERO, or CL_FP_ROUND_TO_NEAREST | CL_FP_INF_NAN.

5.2.13. Rounding Modes

If CL_FP_ROUND_TO_NEAREST is supported, the default rounding mode for half-precision floating-point operations will be round to nearest even; otherwise the default rounding mode will be round to zero.

Conversions to half floating-point format must be correctly rounded using the indicated convert operator rounding mode or the default rounding mode for half-precision floating-point operations if no rounding mode is specified by the operator, or a C-style cast is used.

Conversions from half to integer format shall correctly round using the indicated convert operator rounding mode, or towards zero if no rounding mode is specified by the operator or a C-style cast is used. All conversions from half to floating-point formats are exact.

5.2.14. Relative Error as ULPs

In this section we discuss the maximum relative error defined as *ulp* (units in the last place).

Addition, subtraction, multiplication, fused multiply-add operations on half types are required to be correctly rounded using the default rounding mode for half-precision floating-point operations.

The following table describes the minimum accuracy of half precision floating-point arithmetic operations given as ULP values. 0 ULP is used for math functions that do not require rounding. The reference value used to compute the ULP value of an arithmetic operation is the infinitely precise result.

Table 11. ULP Values for Half Precision Floating-Point Arithmetic Operations

Function	Min Accuracy - Full Profile	Min Accuracy - Embedded Profile
x* + *y	Correctly rounded	Correctly rounded
x* - *y	Correctly rounded	Correctly rounded
x* * *y	Correctly rounded	Correctly rounded
1.0 / *x*	Correctly rounded	<= 1 ulp
x* / *y	Correctly rounded	<= 1 ulp
acos	<= 2 ulp	<= 3 ulp
acosh	<= 2 ulp	<= 3 ulp
acospi	<= 2 ulp	<= 3 ulp
asin	<= 2 ulp	<= 3 ulp
asinh	<= 2 ulp	<= 3 ulp
asinpi	<= 2 ulp	<= 3 ulp
atan	<= 2 ulp	<= 3 ulp
atanh	<= 2 ulp	<= 3 ulp
atanpi	<= 2 ulp	<= 3 ulp
atan2	<= 2 ulp	<= 3 ulp
atan2pi	<= 2 ulp	<= 3 ulp
cbrt	<= 2 ulp	<= 2 ulp
ceil	Correctly rounded	Correctly rounded
clamp	0 ulp	0 ulp
copysign	0 ulp	0 ulp
cos	<= 2 ulp	<= 2 ulp
cosh	<= 2 ulp	<= 3 ulp
cospi	<= 2 ulp	<= 2 ulp

Function	Min Accuracy - Full Profile	Min Accuracy - Embedded Profile
cross	absolute error tolerance of 'max * max * (3 * HALF_EPSILON)' per vector component, where *max* is the maximum input operand magnitude	Implementation-defined
degrees	<= 2 ulp	<= 2 ulp
distance	<= 2n ulp, for gentype with vector width *n*	Implementation-defined
dot	absolute error tolerance of 'max * max * (2n - 1) * HALF_EPSILON', for vector width *n* and maximum input operand magnitude *max* across all vector components	Implementation-defined
erfc	<= 4 ulp	<= 4 ulp
erf	<= 4 ulp	<= 4 ulp
exp	<= 2 ulp	<= 3 ulp
exp2	<= 2 ulp	<= 3 ulp
exp10	<= 2 ulp	<= 3 ulp
expm1	<= 2 ulp	<= 3 ulp
fabs	0 ulp	0 ulp
fdim	Correctly rounded	Correctly rounded
floor	Correctly rounded	Correctly rounded
fma	Correctly rounded	Correctly rounded
fmax	0 ulp	0 ulp
fmin	0 ulp	0 ulp
fmod	0 ulp	0 ulp
fract	Correctly rounded	Correctly rounded
frexp	0 ulp	0 ulp
hypot	<= 2 ulp	<= 3 ulp
ilogb	0 ulp	0 ulp
ldexp	Correctly rounded	Correctly rounded
length	<= 0.25 + 0.5n ulp, for gentype with vector width *n*	Implementation-defined
log	<= 2 ulp	<= 3 ulp
log2	<= 2 ulp	<= 3 ulp

Function	Min Accuracy - Full Profile	Min Accuracy - Embedded Profile
log10	<= 2 ulp	<= 3 ulp
log1p	<= 2 ulp	<= 3 ulp
logb	0 ulp	0 ulp
mad	Implementation-defined	Implementation-defined
max	0 ulp	0 ulp
maxmag	0 ulp	0 ulp
min	0 ulp	0 ulp
minmag	0 ulp	0 ulp
mix	Implementation-defined	Implementation-defined
modf	0 ulp	0 ulp
nan	0 ulp	0 ulp
nextafter	0 ulp	0 ulp
normalize	<= 1 + n ulp, for gentype with vector width *n*	Implementation-defined
pow(x, y)	<= 4 ulp	<= 5 ulp
pown(x, y)	<= 4 ulp	<= 5 ulp
powr(x, y)	<= 4 ulp	<= 5 ulp
radians	<= 2 ulp	<= 2 ulp
remainder	0 ulp	0 ulp
remquo	0 ulp for the remainder, at least the lower 7 bits of the integral quotient	0 ulp for the remainder, at least the lower 7 bits of the integral quotient
rint	Correctly rounded	Correctly rounded
rootn	<= 4 ulp	<= 5 ulp
round	Correctly rounded	Correctly rounded
rsqrt	<=1 ulp	<=1 ulp
sign	0 ulp	0 ulp
sin	<= 2 ulp	<= 2 ulp
sincos	<= 2 ulp for sine and cosine values	<= 2 ulp for sine and cosine values
sinh	<= 2 ulp	<= 3 ulp
sinpi	<= 2 ulp	<= 2 ulp
smoothstep	Implementation-defined	Implementation-defined
sqrt	Correctly rounded	<= 1 ulp

Function	Min Accuracy - Full Profile	Min Accuracy - Embedded Profile
step	0 ulp	0 ulp
tan	<= 2 ulp	<= 3 ulp
tanh	<= 2 ulp	<= 3 ulp
tanpi	<= 2 ulp	<= 3 ulp
tgamma	<= 4 ulp	<= 4 ulp
trunc	Correctly rounded	Correctly rounded

Note: *Implementations may perform floating-point operations on* `half` *scalar or vector data types by converting the* `half` *values to single precision floating-point values and performing the operation in single precision floating-point. In this case, the implementation will use the* `half` *scalar or vector data type as a storage only format.*

5.3. Additions to Chapter 8 of the OpenCL 2.0 C Specification

Add new sub-sections to *section 8.3.1. Conversion rules for normalized integer channel data types*:

5.3.1. Converting normalized integer channel data types to half precision floating-point values

For images created with image channel data type of `CL_UNORM_INT8` and `CL_UNORM_INT16`, **read_imagef** will convert the channel values from an 8-bit or 16-bit unsigned integer to normalized half precision floating-point values in the range [`0.0h`, `1.0h`].

For images created with image channel data type of `CL_SNORM_INT8` and `CL_SNORM_INT16`, **read_imagef** will convert the channel values from an 8-bit or 16-bit signed integer to normalized half precision floating-point values in the range [`-1.0h`, `1.0h`].

These conversions are performed as follows:

`CL_UNORM_INT8` (8-bit unsigned integer) → `half`

normalized `half` value = `round_to_half(c / 255)`

`CL_UNORM_INT_101010` (10-bit unsigned integer) → `half`

normalized `half` value = `round_to_half(c / 1023)`

`CL_UNORM_INT16` (16-bit unsigned integer) → `half`

normalized `half` value = `round_to_half(c / 65535)`

`CL_SNORM_INT8` (8-bit signed integer) → `half`

normalized `half` value = `max(-1.0h, round_to_half(c / 127))`

`CL_SNORM_INT16` (16-bit signed integer) → `half`

normalized `half` value = `max(-1.0h, round_to_half(c / 32767))`

The accuracy of the above conversions must be <= 1.5 ulp except for the following cases.

For `CL_UNORM_INT8`

0 must convert to `0.0h` and

255 must convert to `1.0h`

For `CL_UNORM_INT_101010`

0 must convert to `0.0h` and

1023 must convert to `1.0h`

For `CL_UNORM_INT16`

0 must convert to `0.0h` and

65535 must convert to `1.0h`

For `CL_SNORM_INT8`

-128 and -127 must convert to `-1.0h`,

0 must convert to `0.0h` and

127 must convert to `1.0h`

For `CL_SNORM_INT16`

-32768 and -32767 must convert to `-1.0h`,

0 must convert to `0.0h` and

32767 must convert to `1.0h`

5.3.2. Converting half precision floating-point values to normalized integer channel data types

For images created with image channel data type of `CL_UNORM_INT8` and `CL_UNORM_INT16`, **write_imagef** will convert the floating-point color value to an 8-bit or 16-bit unsigned integer.

For images created with image channel data type of `CL_SNORM_INT8` and `CL_SNORM_INT16`, **write_imagef** will convert the floating-point color value to an 8-bit or 16-bit signed integer.

The preferred conversion uses the round to nearest even (`_rte`) rounding mode, but OpenCL implementations may choose to approximate the rounding mode used in the conversions described below. When approximate rounding is used instead of the preferred rounding, the result of the conversion must satisfy the bound given below.

`half` → `CL_UNORM_INT8` (8-bit unsigned integer)

Let f_{exact} = **max**(`0`, **min**(`f * 255`, `255`))

Let $f_{preferred}$ = **convert_uchar_sat_rte**(`f * 255.0f`)

Let f_{approx} = **convert_uchar_sat_<impl-rounding-mode>**(`f * 255.0f`)

fabs(f_{exact} - f_{approx}) must be <= 0.6

`half` → `CL_UNORM_INT_101010` (10-bit unsigned integer)

Let f_{exact} = **max**(`0`, **min**(`f * 1023`, `1023`))

Let $f_{preferred}$ = **min**(**convert_ushort_sat_rte**(`f * 1023.0f`), `1023`)

Let f_{approx} = **convert_ushort_sat_<impl-rounding-mode>**(`f * 1023.0f`)

fabs(f_{exact} - f_{approx}) must be <= 0.6

`half` → `CL_UNORM_INT16` (16-bit unsigned integer)

Let f_{exact} = **max**(`0`, **min**(`f * 65535`, `65535`))

Let $f_{preferred}$ = **convert_ushort_sat_rte**(`f * 65535.0f`)

Let f_{approx} = **convert_ushort_sat_<impl-rounding-mode>**(`f * 65535.0f`)

fabs(f_{exact} - f_{approx}) must be <= 0.6

`half` → `CL_SNORM_INT8` (8-bit signed integer)

Let f_{exact} = **max**(`-128`, **min**(`f * 127`, `127`))

Let $f_{preferred}$ = **convert_char_sat_rte**(`f * 127.0f`)

Let f_{approx} = **convert_char_sat_<impl_rounding_mode>**(`f * 127.0f`)

fabs(f_{exact} - f_{approx}) must be <= 0.6

`half` → `CL_SNORM_INT16` (16-bit signed integer)

Let f_{exact} = **max**(`-32768`, **min**(`f * 32767`, `32767`))

Let $f_{preferred}$ = **convert_short_sat_rte**(`f * 32767.0f`)

Let f_{approx} = **convert_short_sat_<impl-rounding-mode>**(`f * 32767.0f`)

fabs(f_{exact} - f_{approx}) must be <= 0.6

Chapter 6. Double Precision Floating-Point

This section describes the **cl_khr_fp64** extension. This extension became an optional core feature in OpenCL 1.2.

6.1. General Information

6.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

6.2. Additions to Chapter 6

The list of built-in scalar, and vector data types defined in *tables 6.1* and *6.2* are extended to include the following:

Type	Description
double	A double precision float.
double2	A 2-component double-precision floating-point vector.
double3	A 3-component double-precision floating-point vector.
double4	A 4-component double-precision floating-point vector.
double8	A 8-component double-precision floating-point vector.
double16	A 16-component double-precision floating-point vector.

The built-in scalar and vector data types for `doublen` are also declared as appropriate types in the OpenCL API (and header files) that can be used by an application. The following table describes the built-in scalar and vector data types for `doublen` as defined in the OpenCL C programming language and the corresponding data type available to the application:

Type in OpenCL Language	API type for application
double	**cl_double**
double2	**cl_double2**
double3	**cl_double3**
double4	**cl_double4**
double8	**cl_double8**
double16	**cl_double16**

The double data type must conform to the IEEE-754 double precision storage format.

The following text is added to *Section 6.1.1.1 The half data type*:

Conversions from double to half are correctly rounded. Conversions from half to double are lossless.

6.2.1. Conversions

The implicit conversion rules specified in *section 6.2.1* now include the `double` scalar and `doublen` vector data types.

The explicit casts described in *section 6.2.2* are extended to take a `double` scalar data type and a `doublen` vector data type.

The explicit conversion functions described in *section 6.2.3* are extended to take a `double` scalar data type and a `doublen` vector data type.

The `as_typen()` function for re-interpreting types as described in *section 6.2.4.2* is extended to allow conversion-free casts between `longn`, `ulongn` and `doublen` scalar and vector data types.

6.2.2. Math Functions

The built-in math functions defined in *table 6.8* (also listed below) are extended to include appropriate versions of functions that take `double` and `double{2|3|4|8|16}` as arguments and return values. `gentype` now also includes `double`, `double2`, `double3`, `double4`, `double8` and `double16`.

For any specific use of a function, the actual type has to be the same for all arguments and the return type.

Table 12. Double Precision Built-in Math Functions

Function	Description
gentype **acos** (gentype *x*)	Arc cosine function.
gentype **acosh** (gentype *x*)	Inverse hyperbolic cosine.
gentype **acospi** (gentype *x*)	Compute **acos** (*x*) / π.
gentype **asin** (gentype *x*)	Arc sine function.
gentype **asinh** (gentype *x*)	Inverse hyperbolic sine.
gentype **asinpi** (gentype *x*)	Compute **asin** (*x*) / π.
gentype **atan** (gentype *y_over_x*)	Arc tangent function.
gentype **atan2** (gentype *y*, gentype *x*)	Arc tangent of *y* / *x*.
gentype **atanh** (gentype *x*)	Hyperbolic arc tangent.
gentype **atanpi** (gentype *x*)	Compute **atan** (*x*) / π.
gentype **atan2pi** (gentype *y*, gentype *x*)	Compute **atan2** (*y*, *x*) / π.
gentype **cbrt** (gentype *x*)	Compute cube-root.
gentype **ceil** (gentype *x*)	Round to integral value using the round to positive infinity rounding mode.

Function	Description
gentype **copysign** (gentype *x*, gentype *y*)	Returns *x* with its sign changed to match the sign of *y*.
gentype **cos** (gentype *x*)	Compute cosine.
gentype **cosh** (gentype *x*)	Compute hyperbolic cosine.
gentype **cospi** (gentype *x*)	Compute **cos** (π *x*).
gentype **erfc** (gentype *x*)	Complementary error function.
gentype **erf** (gentype *x*)	Error function encountered in integrating the normal distribution.
gentype **exp** (gentype *x*)	Compute the base- e exponential of *x*.
gentype **exp2** (gentype *x*)	Exponential base 2 function.
gentype **exp10** (gentype *x*)	Exponential base 10 function.
gentype **expm1** (gentype *x*)	Compute e^x- 1.0.
gentype **fabs** (gentype *x*)	Compute absolute value of a floating-point number.
gentype **fdim** (gentype *x*, gentype *y*)	*x* - *y* if *x* > *y*, +0 if x is less than or equal to y.
gentype **floor** (gentype *x*)	Round to integral value using the round to negative infinity rounding mode.
gentype **fma** (gentype *a*, gentype *b*, gentype *c*)	Returns the correctly rounded floating-point representation of the sum of *c* with the infinitely precise product of *a* and *b*. Rounding of intermediate products shall not occur. Edge case behavior is per the IEEE 754-2008 standard.
gentype **fmax** (gentype *x*, gentype *y*) gentype **fmax** (gentype *x*, double *y*)	Returns *y* if *x* < *y*, otherwise it returns *x*. If one argument is a NaN, **fmax()** returns the other argument. If both arguments are NaNs, **fmax()** returns a NaN.
gentype **fmin** (gentype *x*, gentype *y*) gentype **fmin** (gentype *x*, double *y*)	Returns *y* if *y* < *x*, otherwise it returns *x*. If one argument is a NaN, **fmin()** returns the other argument. If both arguments are NaNs, **fmin()** returns a NaN.
gentype **fmod** (gentype *x*, gentype *y*)	Modulus. Returns *x* - *y* * **trunc** (*x*/*y*) .
gentype **fract** (gentype *x*, __global gentype **iptr*) gentype **fract** (gentype *x*, __local gentype **iptr*) gentype **fract** (gentype *x*, __private gentype * *iptr*)	Returns **fmin**(*x* - **floor** (*x*), 0x1. fffffffffffffp-1). **floor**(x) is returned in *iptr*.

Function	Description
double*n* **frexp** (double*n* *x*, __global int*n* *exp) double*n* **frexp** (double*n* *x*, __local int*n* *exp) double*n* **frexp** (double*n* *x*, __private int*n* *exp) double **frexp** (double *x*, __global int *exp) double **frexp** (double *x*, __local int *exp) double **frexp** (double *x*, __private int *exp)	Extract mantissa and exponent from *x*. For each component the mantissa returned is a `double` with magnitude in the interval [1/2, 1) or 0. Each component of *x* equals mantissa returned * 2^{exp}.
gentype **hypot** (gentype *x*, gentype *y*)	Compute the value of the square root of $x^2 + y^2$ without undue overflow or underflow.
int*n* **ilogb** (double*n* *x*) int **ilogb** (double *x*)	Return the exponent as an integer value.
double*n* **ldexp** (double*n* *x*, int*n* *k*) double*n* **ldexp** (double*n* *x*, int *k*) double **ldexp** (double *x*, int *k*)	Multiply *x* by 2 to the power *k*.
gentype **lgamma** (gentype *x*) double*n* **lgamma_r** (double*n* *x*, __global int*n* **signp*) double*n* **lgamma_r** (double*n* *x*, __local int*n* **signp*) double*n* **lgamma_r** (double*n* *x*, __private int*n* **signp*) double **lgamma_r** (double *x*, __global int **signp*) double **lgamma_r** (double *x*, __local int **signp*) double **lgamma_r** (double *x*, __private int * *signp*)	Log gamma function. Returns the natural logarithm of the absolute value of the gamma function. The sign of the gamma function is returned in the *signp* argument of **lgamma_r**.
gentype **log** (gentype *x*)	Compute natural logarithm.
gentype **log2** (gentype *x*)	Compute a base 2 logarithm.
gentype **log10** (gentype *x*)	Compute a base 10 logarithm.
gentype **log1p** (gentype *x*)	Compute $\log_e(1.0 + x)$.
gentype **logb** (gentype *x*)	Compute the exponent of *x*, which is the integral part of $\log_r\|x\|$.
gentype **mad** (gentype *a*, gentype *b*, gentype *c*)	**mad** computes *a* * *b* + *c*. The function may compute *a* * *b* + *c* with reduced accuracy in the embedded profile. See the OpenCL SPIR-V Environment Specification for details. On some hardware the mad instruction may provide better performance than expanded computation of *a* * *b* + *c*.
gentype **maxmag** (gentype *x*, gentype *y*)	Returns *x* if $\|x\| > \|y\|$, *y* if $\|y\| > \|x\|$, otherwise **fmax**(*x*, *y*).
gentype **minmag** (gentype *x*, gentype *y*)	Returns *x* if $\|x\| < \|y\|$, *y* if $\|y\| < \|x\|$, otherwise **fmin**(*x*, *y*).

Function	Description
gentype **modf** (gentype *x*, __global gentype **iptr*) gentype **modf** (gentype *x*, __local gentype **iptr*) gentype **modf** (gentype *x*, __private gentype * *iptr*)	Decompose a floating-point number. The **modf** function breaks the argument *x* into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part in the object pointed to by *iptr*.
double*n* **nan** (ulong*n* *nancode*) double **nan** (ulong *nancode*)	Returns a quiet NaN. The *nancode* may be placed in the significand of the resulting NaN.
gentype **nextafter** (gentype *x*, gentype *y*)	Computes the next representable double-precision floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, **nextafter**() returns the largest representable floating-point number less than *x*.
gentype **pow** (gentype *x*, gentype *y*)	Compute *x* to the power *y*.
double*n* **pown** (double*n* *x*, int*n* *y*) double **pown** (double *x*, int *y*)	Compute *x* to the power *y*, where *y* is an integer.
gentype **powr** (gentype *x*, gentype *y*)	Compute *x* to the power *y*, where *x* is >= 0.
gentype **remainder** (gentype *x*, gentype *y*)	Compute the value *r* such that *r* = *x* - *n***y*, where *n* is the integer nearest the exact value of *x*/*y*. If there are two integers closest to *x*/*y*, *n* shall be the even one. If *r* is zero, it is given the same sign as *x*.
double*n* **remquo** (double*n* *x*, double*n* *y*, __global int*n* **quo*) double*n* **remquo** (double*n* *x*, double*n* *y*, __local int*n* **quo*) double*n* **remquo** (double*n* *x*, double*n* *y*, __private int*n* **quo*) double **remquo** (double *x*, double *y*, __global int **quo*) double **remquo** (double *x*, double *y*, __local int **quo*) double **remquo** (double *x*, double *y*, __private int **quo*)	The **remquo** function computes the value r such that *r* = *x* - *k***y*, where *k* is the integer nearest the exact value of *x*/*y*. If there are two integers closest to *x*/*y*, *k* shall be the even one. If *r* is zero, it is given the same sign as *x*. This is the same value that is returned by the **remainder** function. **remquo** also calculates the lower seven bits of the integral quotient *x*/*y*, and gives that value the same sign as *x*/*y*. It stores this signed value in the object pointed to by *quo*.
gentype **rint** (gentype *x*)	Round to integral value (using round to nearest even rounding mode) in floating-point format. Refer to section 7.1 for description of rounding modes.
double*n* **rootn** (double*n* *x*, int*n* *y*) double **rootn** (double *x*, int *y*)	Compute *x* to the power 1/*y*.
gentype **round** (gentype *x*)	Return the integral value nearest to *x* rounding halfway cases away from zero, regardless of the current rounding direction.

Function	Description
gentype **rsqrt** (gentype *x*)	Compute inverse square root.
gentype **sin** (gentype *x*)	Compute sine.
gentype **sincos** (gentype *x*, __global gentype **cosval*) gentype **sincos** (gentype *x*, __local gentype **cosval*) gentype **sincos** (gentype *x*, __private gentype **cosval*)	Compute sine and cosine of x. The computed sine is the return value and computed cosine is returned in *cosval*.
gentype **sinh** (gentype *x*)	Compute hyperbolic sine.
gentype **sinpi** (gentype *x*)	Compute **sin** (π *x*).
gentype **sqrt** (gentype *x*)	Compute square root.
gentype **tan** (gentype *x*)	Compute tangent.
gentype **tanh** (gentype *x*)	Compute hyperbolic tangent.
gentype **tanpi** (gentype *x*)	Compute **tan** (π *x*).
gentype **tgamma** (gentype *x*)	Compute the gamma function.
gentype **trunc** (gentype *x*)	Round to integral value using the round to zero rounding mode.

In addition, the following symbolic constant will also be available:

HUGE_VAL - A positive double expression that evaluates to infinity. Used as an error value returned by the built-in math functions.

The **FP_FAST_FMA** macro indicates whether the **fma()** family of functions are fast compared with direct code for double precision floating-point. If defined, the **FP_FAST_FMA** macro shall indicate that the **fma()** function generally executes about as fast as, or faster than, a multiply and an add of **double** operands.

The macro names given in the following list must use the values specified. These constant expressions are suitable for use in #if preprocessing directives.

```
#define DBL_DIG             15
#define DBL_MANT_DIG        53
#define DBL_MAX_10_EXP      +308
#define DBL_MAX_EXP         +1024
#define DBL_MIN_10_EXP      -307
#define DBL_MIN_EXP         -1021
#define DBL_RADIX           2
#define DBL_MAX             0x1.fffffffffffffp1023
#define DBL_MIN             0x1.0p-1022
#define DBL_EPSILON         0x1.0p-52
```

The following table describes the built-in macro names given above in the OpenCL C programming language and the corresponding macro names available to the application.

Macro in OpenCL Language	Macro for application
DBL_DIG	CL_DBL_DIG
DBL_MANT_DIG	CL_DBL_MANT_DIG
DBL_MAX_10_EXP	CL_DBL_MAX_10_EXP
DBL_MAX_EXP	CL_DBL_MAX_EXP
DBL_MIN_10_EXP	CL_DBL_MIN_10_EXP
DBL_MIN_EXP	CL_DBL_MIN_EXP
DBL_RADIX	CL_DBL_RADIX
DBL_MAX	CL_DBL_MAX
DBL_MIN	CL_DBL_MIN
DBL_EPSILSON	CL_DBL_EPSILON

The following constants are also available. They are of type `double` and are accurate within the precision of the `double` type.

Constant	Description
M_E	Value of e
M_LOG2E	Value of $\log_2 e$
M_LOG10E	Value of $\log_{10} e$
M_LN2	Value of $\log_e 2$
M_LN10	Value of $\log_e 10$
M_PI	Value of π
M_PI_2	Value of $\pi / 2$
M_PI_4	Value of $\pi / 4$
M_1_PI	Value of $1 / \pi$
M_2_PI	Value of $2 / \pi$
M_2_SQRTPI	Value of $2 / \sqrt{\pi}$
M_SQRT2	Value of $\sqrt{2}$
M_SQRT1_2	Value of $1 / \sqrt{2}$

6.2.3. Common Functions

The built-in common functions defined in *table 6.12* (also listed below) are extended to include appropriate versions of functions that take `double` and `double{2|3|4|8|16}` as arguments and return values. gentype now also includes `double`, `double2`, `double3`, `double4`, `double8` and `double16`. These are described below.

Table 13. Double Precision Built-in Common Functions

Function	Description
gentype **clamp** (gentype *x*, gentype *minval*, gentype *maxval*) gentype **clamp** (gentype *x*, double *minval*, double *maxval*)	Returns **fmin**(**fmax**(*x*, *minval*), *maxval*). Results are undefined if *minval* > *maxval*.
gentype **degrees** (gentype *radians*)	Converts *radians* to degrees, i.e. (180 / π) * *radians*.
gentype **max** (gentype *x*, gentype *y*) gentype **max** (gentype *x*, double *y*)	Returns *y* if *x* < *y*, otherwise it returns *x*. If *x* and *y* are infinite or NaN, the return values are undefined.
gentype **min** (gentype *x*, gentype *y*) gentype **min** (gentype *x*, double *y*)	Returns *y* if *y* < *x*, otherwise it returns *x*. If *x* and *y* are infinite or NaN, the return values are undefined.
gentype **mix** (gentype *x*, gentype *y*, gentype *a*) gentype **mix** (gentype *x*, gentype *y*, double *a*)	Returns the linear blend of *x* and *y* implemented as: *x* + (*y* - *x*) * *a* *a* must be a value in the range 0.0 … 1.0. If *a* is not in the range 0.0 … 1.0, the return values are undefined. Note: The double precision **mix** function can be implemented using contractions such as **mad** or **fma**.
gentype **radians** (gentype *degrees*)	Converts *degrees* to radians, i.e. (π / 180) * *degrees*.
gentype **step** (gentype *edge*, gentype *x*) gentype **step** (double *edge*, gentype *x*)	Returns 0.0 if *x* < *edge*, otherwise it returns 1.0.

Function	Description
gentype **smoothstep** (gentype *edge0*, gentype *edge1*, gentype *x*) gentype **smoothstep** (double *edge0*, double *edge1*, gentype *x*)	Returns 0.0 if *x* <= *edge0* and 1.0 if *x* >= *edge1* and performs smooth Hermite interpolation between 0 and 1 when *edge0* < *x* < *edge1*. This is useful in cases where you would want a threshold function with a smooth transition. This is equivalent to: gentype *t*; *t* = clamp ((*x* - *edge0*) / (*edge1* - *edge0*), 0, 1); return *t* * *t* * (3 - 2 * *t*); Results are undefined if *edge0* >= *edge1*. Note: The double precision **smoothstep** function can be implemented using contractions such as **mad** or **fma**.
gentype **sign** (gentype *x*)	Returns 1.0 if *x* > 0, -0.0 if *x* = -0.0, +0.0 if *x* = +0.0, or -1.0 if *x* < 0. Returns 0.0 if *x* is a NaN.

6.2.4. Geometric Functions

The built-in geometric functions defined in *table 6.13* (also listed below) are extended to include appropriate versions of functions that take `double` and `double{2|3|4}` as arguments and return values. gentype now also includes `double`, `double2`, `double3` and `double4`. These are described below.

Note: The double precision geometric functions can be implemented using contractions such as **mad** or **fma**.

Table 14. Double Precision Built-in Geometric Functions

Function	Description
double4 **cross** (double4 *p0*, double4 *p1*) double3 **cross** (double3 *p0*, double3 *p1*)	Returns the cross product of *p0.xyz* and *p1.xyz*. The *w* component of the result will be 0.0.
double **dot** (gentype *p0*, gentype *p1*)	Compute the dot product of *p0* and *p1*.
double **distance** (gentype *p0*, gentype *p1*)	Returns the distance between *p0* and *p1*. This is calculated as **length**(*p0* - *p1*).
double **length** (gentype *p*)	Return the length of vector x, i.e., sqrt($p.x^2 + p.y^2 + \ldots$)
gentype **normalize** (gentype *p*)	Returns a vector in the same direction as *p* but with a length of 1.

6.2.5. Relational Functions

The scalar and vector relational functions described in *table 6.14* are extended to include versions that take `double`, `double2`, `double3`, `double4`, `double8` and `double16` as arguments.

The relational and equality operators (<, <=, >, >=, !=, ==) can be used with `doublen` vector types and shall produce a vector `longn` result as described in *section 6.3*.

The functions **isequal**, **isnotequal**, **isgreater**, **isgreaterequal**, **isless**, **islessequal**, **islessgreater**, **isfinite**, **isinf**, **isnan**, **isnormal**, **isordered**, **isunordered** and **signbit** shall return a 0 if the specified relation is *false* and a 1 if the specified relation is true for scalar argument types. These functions shall return a 0 if the specified relation is *false* and a -1 (i.e. all bits set) if the specified relation is *true* for vector argument types.

The relational functions **isequal**, **isgreater**, **isgreaterequal**, **isless**, **islessequal**, and **islessgreater** always return 0 if either argument is not a number (NaN). **isnotequal** returns 1 if one or both arguments are not a number (NaN) and the argument type is a scalar and returns -1 if one or both arguments are not a number (NaN) and the argument type is a vector.

The functions described in *table 6.14* are extended to include the `doublen` vector types.

Table 15. Double Precision Relational Functions

Function	Description
int **isequal** (double *x*, double *y*) long*n* **isequal** (double*n* *x*, double*n* *y*)	Returns the component-wise compare of *x* == *y*.
int **isnotequal** (double *x*, double *y*) long*n* **isnotequal** (double*n* *x*, double*n* *y*)	Returns the component-wise compare of *x* != *y*.
int **isgreater** (double *x*, double *y*) long*n* **isgreater** (double*n* *x*, double*n* *y*)	Returns the component-wise compare of *x* > *y*.
int **isgreaterequal** (double *x*, double *y*) long*n* **isgreaterequal** (double*n* *x*, double*n* *y*)	Returns the component-wise compare of *x* >= *y*.
int **isless** (double *x*, double *y*) long*n* **isless** (double*n* *x*, double*n* *y*)	Returns the component-wise compare of *x* < *y*.
int **islessequal** (double *x*, double *y*) long*n* **islessequal** (double*n* *x*, double*n* *y*)	Returns the component-wise compare of *x* <= *y*.
int **islessgreater** (double *x*, double *y*) long*n* **islessgreater** (double*n* *x*, double*n* *y*)	Returns the component-wise compare of (*x* < *y*) || (*x* > *y*) .
int **isfinite** (double) long*n* **isfinite** (double*n*)	Test for finite value.
int **isinf** (double) long*n* **isinf** (double*n*)	Test for infinity value (positive or negative) .
int **isnan** (double) long*n* **isnan** (double*n*)	Test for a NaN.

Function	Description
int **isnormal** (double) long*n* **isnormal** (double*n*)	Test for a normal value.
int **isordered** (double *x*, double *y*) long*n* **isordered** (double*n* *x*, double*n* *y*)	Test if arguments are ordered. **isordered**() takes arguments *x* and *y*, and returns the result **isequal**(*x*, *x*) && **isequal**(*y*, *y*).
int **isunordered** (double *x*, double *y*) long*n* **isunordered** (double*n* *x*, double*n* *y*)	Test if arguments are unordered. **isunordered**() takes arguments *x* and *y*, returning non-zero if *x* or *y* is a NaN, and zero otherwise.
int **signbit** (double) long*n* **signbit** (double*n*)	Test for sign bit. The scalar version of the function returns a 1 if the sign bit in the double is set else returns 0. The vector version of the function returns the following for each component in double*n*: -1 (i.e all bits set) if the sign bit in the double is set else returns 0.
double*n* **bitselect** (double*n* *a*, double*n* *b*, double*n* *c*)	Each bit of the result is the corresponding bit of *a* if the corresponding bit of *c* is 0. Otherwise it is the corresponding bit of *b*.
double*n* **select** (double*n* *a*, double*n* *b*, long*n* *c*) double*n* **select** (double*n* *a*, double*n* *b*, ulong*n* *c*)	For each component, *result[i]* = if MSB of *c[i]* is set ? *b[i]* : *a[i]*.

6.2.6. Vector Data Load and Store Functions

The vector data load (**vload*n***) and store (**vstore*n***) functions described in *table 6.13* (also listed below) are extended to include versions that read from or write to double scalar or vector values. The generic type `gentype` is extended to include `double`. The generic type `gentypen` is extended to include `double2`, `double3`, `double4`, `double8` and `double16`. The **vstore_half**, **vstore_half*n*** and **vstorea_half*n*** functions are extended to allow a double precision scalar or vector value to be written to memory as half values.

Note: **vload3** reads (*x*,*y*,*z*) components from address `(p + (offset * 3))` into a 3-component vector. **vstore3**, and **vstore_half3** write (*x*,*y*,*z*) components from a 3-component vector to address `(p + (offset * 3))`. In addition, **vloada_half3** reads (*x*,*y*,*z*) components from address `(p + (offset * 4))` into a 3-component vector and **vstorea_half3** writes (*x*,*y*,*z*) components from a 3-component vector to address `(p + (offset * 4))`. Whether **vloada_half3** and **vstorea_half3** read/write padding data between the third vector element and the next alignment boundary is implementation-defined. **vloada_** and **vstoreaa_** variants are provided to access data that is aligned to the size of the vector, and are intended to enable performance on hardware that can take advantage of the increased alignment.

Table 16. Double Precision Vector Data Load and Store Functions

Function	Description
gentype*n* **vload*n***(size_t *offset*, const __global gentype **p*) gentype*n* **vload*n***(size_t *offset*, const __local gentype **p*) gentype*n* **vload*n***(size_t *offset*, const __constant gentype **p*) gentype*n* **vload*n***(size_t *offset*, const __private gentype **p*)	Return sizeof (gentype*n*) bytes of data read from address (*p* + (*offset* * *n*)). If gentype is double, the read address computed as (*p* + (*offset* * *n*)) must be 64-bit aligned.
void **vstore*n***(gentype*n* *data*, size_t *offset*, __global gentype **p*) void **vstore*n***(gentype*n* *data*, size_t *offset*, __local gentype **p*) void **vstore*n***(gentype*n* *data*, size_t *offset*, __private gentype **p*)	Write sizeof (gentype*n*) bytes given by *data* to address (*p* + (*offset* * *n*)). If gentype is double, the write address computed as (*p* + (*offset* * *n*)) must be 64-bit aligned.

Function	Description
void **vstore_half**(double *data*, size_t *offset*, __global half **p*) void **vstore_half_rte**(double *data*, size_t *offset*, __global half **p*) void **vstore_half_rtz**(double *data*, size_t *offset*, __global half **p*) void **vstore_half_rtp**(double *data*, size_t *offset*, __global half **p*) void **vstore_half_rtn**(double *data*, size_t *offset*, __global half **p*) void **vstore_half**(double *data*, size_t *offset*, __local half **p*) void **vstore_half_rte**(double *data*, size_t *offset*, __local half **p*) void **vstore_half_rtz**(double *data*, size_t *offset*, __local half **p*) void **vstore_half_rtp**(double *data*, size_t *offset*, __local half **p*) void **vstore_half_rtn**(double *data*, size_t *offset*, __local half **p*) void **vstore_half**(double *data*, size_t *offset*, __private half **p*) void **vstore_half_rte**(double *data*, size_t *offset*, __private half **p*) void **vstore_half_rtz**(double *data*, size_t *offset*, __private half **p*) void **vstore_half_rtp**(double *data*, size_t *offset*, __private half **p*) void **vstore_half_rtn**(double *data*, size_t *offset*, __private half **p*)	The double value given by *data* is first converted to a half value using the appropriate rounding mode. The half value is then written to the address computed as (*p* + *offset*). The address computed as (*p* + *offset*) must be 16-bit aligned. **vstore_half** uses the current rounding mode. The default current rounding mode is round to nearest even.

Function	Description
void **vstore_half*n***(double*n* *data*, size_t *offset*, __global half *p) void **vstore_half*n*_rte**(double*n* *data*, size_t *offset*, __global half *p) void **vstore_half*n*_rtz**(double*n* *data*, size_t *offset*, __global half *p) void **vstore_half*n*_rtp**(double*n* *data*, size_t *offset*, __global half *p) void **vstore_half*n*_rtn**(double*n* *data*, size_t *offset*, __global half *p) void **vstore_half*n***(double*n* *data*, size_t *offset*, __local half *p) void **vstore_half*n*_rte**(double*n* *data*, size_t *offset*, __local half *p) void **vstore_half*n*_rtz**(double*n* *data*, size_t *offset*, __local half *p) void **vstore_half*n*_rtp**(double*n* *data*, size_t *offset*, __local half *p) void **vstore_half*n*_rtn**(double*n* *data*, size_t *offset*, __local half *p) void **vstore_half*n***(double*n* *data*, size_t *offset*, __private half *p) void **vstore_half*n*_rte**(double*n* *data*, size_t *offset*, __private half *p) void **vstore_half*n*_rtz**(double*n* *data*, size_t *offset*, __private half *p) void **vstore_half*n*_rtp**(double*n* *data*, size_t *offset*, __private half *p) void **vstore_half*n*_rtn**(double*n* *data*, size_t *offset*, __private half *p)	The double*n* value given by *data* is converted to a half*n* value using the appropriate rounding mode. The half*n* value is then written to the address computed as (*p* + (*offset* * *n*)). The address computed as (*p* + (*offset* * *n*)) must be 16-bit aligned. **vstore_half*n*** uses the current rounding mode. The default current rounding mode is round to nearest even.

Function	Description
void **vstorea_half*n***(double*n* *data*, size_t *offset*, __global half **p*) void **vstorea_half*n*_rte**(double*n* *data*, size_t *offset*, __global half **p*) void **vstorea_half*n*_rtz**(double*n* *data*, size_t *offset*, __global half **p*) void **vstorea_half*n*_rtp**(double*n* *data*, size_t *offset*, __global half **p*) void **vstorea_half*n*_rtn**(double*n* *data*, size_t *offset*, __global half **p*) void **vstorea_half*n***(double*n* *data*, size_t *offset*, __local half **p*) void **vstorea_half*n*_rte**(double*n* *data*, size_t *offset*, __local half **p*) void **vstorea_half*n*_rtz**(double*n* *data*, size_t *offset*, __local half **p*) void **vstorea_half*n*_rtp**(double*n* *data*, size_t *offset*, __local half **p*) void **vstorea_half*n*_rtn**(double*n* *data*, size_t *offset*, __local half **p*) void **vstorea_half*n***(double*n* *data*, size_t *offset*, __private half **p*) void **vstorea_half*n*_rte**(double*n* *data*, size_t *offset*, __private half **p*) void **vstorea_half*n*_rtz**(double*n* *data*, size_t *offset*, __private half **p*) void **vstorea_half*n*_rtp**(double*n* *data*, size_t *offset*, __private half **p*) void **vstorea_half*n*_rtn**(double*n* *data*, size_t *offset*, __private half **p*)	The double*n* value is converted to a half*n* value using the appropriate rounding mode. For n = 1, 2, 4, 8 or 16, the half*n* value is written to the address computed as (*p* + (*offset* * *n*)). The address computed as (*p* + (*offset* * *n*)) must be aligned to sizeof (half*n*) bytes. For n = 3, the half*3* value is written to the address computed as (*p* + (*offset* * *4*)). The address computed as (*p* + (*offset* * *4*)) must be aligned to sizeof (half) * 4 bytes. **vstorea_half*n*** uses the current rounding mode. The default current rounding mode is round to nearest even.

6.2.7. Async Copies from Global to Local Memory, Local to Global Memory, and Prefetch

The OpenCL C programming language implements the following functions that provide asynchronous copies between global and local memory and a prefetch from global memory.

The generic type gentype is extended to include `double`, `double2`, `double3`, `double4`, `double8` and `double16`.

Table 17. Double Precision Built-in Async Copy and Prefetch Functions

Function	Description
event_t **async_work_group_copy** (__local gentype **dst*, const __global gentype **src*, size_t *num_gentypes*, event_t *event*) event_t **async_work_group_copy** (__global gentype **dst*, const __local gentype **src*, size_t *num_gentypes*, event_t *event*)	Perform an async copy of *num_gentypes* gentype elements from *src* to *dst*. The async copy is performed by all work-items in a work-group and this built-in function must therefore be encountered by all work-items in a work-group executing the kernel with the same argument values; otherwise the results are undefined. Returns an event object that can be used by **wait_group_events** to wait for the async copy to finish. The *event* argument can also be used to associate the **async_work_group_copy** with a previous async copy allowing an event to be shared by multiple async copies; otherwise *event* should be zero. If *event* argument is not zero, the event object supplied in *event* argument will be returned. This function does not perform any implicit synchronization of source data such as using a **barrier** before performing the copy.

Function	Description
event_t **async_work_group_strided_copy** (__local gentype **dst*, const __global gentype **src*, size_t *num_gentypes*, size_t *src_stride*, event_t *event*) event_t **async_work_group_strided_copy** (__global gentype **dst*, const __local gentype **src*, size_t *num_gentypes*, size_t *dst_stride*, event_t *event*)	Perform an async gather of *num_gentypes* gentype elements from *src* to *dst*. The *src_stride* is the stride in elements for each gentype element read from *src*. The async gather is performed by all work-items in a work-group and this built-in function must therefore be encountered by all work-items in a work-group executing the kernel with the same argument values; otherwise the results are undefined. Returns an event object that can be used by **wait_group_events** to wait for the async copy to finish. The *event* argument can also be used to associate the **async_work_group_strided_copy** with a previous async copy allowing an event to be shared by multiple async copies; otherwise *event* should be zero. If *event* argument is not zero, the event object supplied in *event* argument will be returned. This function does not perform any implicit synchronization of source data such as using a **barrier** before performing the copy. The behavior of **async_work_group_strided_copy** is undefined if *src_stride* or *dst_stride* is 0, or if the *src_stride* or *dst_stride* values cause the *src* or *dst* pointers to exceed the upper bounds of the address space during the copy.
void **wait_group_events** (int *num_events*, event_t **event_list*)	Wait for events that identify the **async_work_group_copy** operations to complete. The event objects specified in *event_list* will be released after the wait is performed. This function must be encountered by all work-items in a work-group executing the kernel with the same *num_events* and event objects specified in *event_list*; otherwise the results are undefined.

Function	Description
void **prefetch** (const __global gentype **p*, size_t *num_gentypes*)	Prefetch *num_gentypes* * sizeof(gentype) bytes into the global cache. The prefetch instruction is applied to a work-item in a work-group and does not affect the functional behavior of the kernel.

6.2.8. IEEE754 Compliance

The following table entry describes the additions to *table 4.3*, which allows applications to query the configuration information using **clGetDeviceInfo** for an OpenCL device that supports double precision floating-point.

Op-code	Return Type	Description
CL_DEVICE_DOUBLE_FP_CONFIG	cl_device_fp_config	Describes double precision floating-point capability of the OpenCL device. This is a bit-field that describes one or more of the following values: CL_FP_DENORM — denorms are supported CL_FP_INF_NAN — INF and NaNs are supported CL_FP_ROUND_TO_NEAREST — round to nearest even rounding mode supported CL_FP_ROUND_TO_ZERO — round to zero rounding mode supported CL_FP_ROUND_TO_INF — round to positive and negative infinity rounding modes supported CL_FP_FMA — IEEE754-2008 fused multiply-add is supported CL_FP_SOFT_FLOAT — Basic floating-point operations (such as addition, subtraction, multiplication) are implemented in software. The required minimum double precision floating-point capability as implemented by this extension is: CL_FP_FMA | CL_FP_ROUND_TO_NEAREST | CL_FP_ROUND_TO_ZERO | CL_FP_ROUND_TO_INF | CL_FP_INF_NAN | CL_FP_DENORM.

IEEE754 fused multiply-add, denorms, INF and NaNs are required to be supported for double precision floating-point numbers and operations on double precision floating-point numbers.

6.2.9. Relative Error as ULPs

In this section we discuss the maximum relative error defined as *ulp* (units in the last place).

Addition, subtraction, multiplication, fused multiply-add and conversion between integer and a floating-point format are IEEE 754 compliant and are therefore correctly rounded using round-to-nearest even rounding mode.

The following table describes the minimum accuracy of double precision floating-point arithmetic operations given as ULP values. 0 ULP is used for math functions that do not require rounding. The reference value used to compute the ULP value of an arithmetic operation is the infinitely precise result.

Table 18. ULP Values for Double Precision Floating-Point Arithmetic Operations

Function	Min Accuracy
x* + *y	Correctly rounded
x* - *y	Correctly rounded
x* * *y	Correctly rounded
1.0 / *x*	Correctly rounded
x* / *y	Correctly rounded
acos	<= 4 ulp
acosh	<= 4 ulp
acospi	<= 5 ulp
asin	<= 4 ulp
asinh	<= 4 ulp
asinpi	<= 5 ulp
atan	<= 5 ulp
atanh	<= 5 ulp
atanpi	<= 5 ulp
atan2	<= 6 ulp
atan2pi	<= 6 ulp
cbrt	<= 2 ulp
ceil	Correctly rounded
clamp	0 ulp
copysign	0 ulp
cos	<= 4 ulp
cosh	<= 4 ulp
cospi	<= 4 ulp
cross	absolute error tolerance of 'max * max * (3 * FLT_EPSILON)' per vector component, where *max* is the maximum input operand magnitude
degrees	<= 2 ulp
distance	<= 5.5 + 2n ulp, for gentype with vector width *n*

Function	Min Accuracy
dot	absolute error tolerance of 'max * max * (2n - 1) * FLT_EPSILON', for vector width *n* and maximum input operand magnitude *max* across all vector components
erfc	<= 16 ulp
erf	<= 16 ulp
exp	<= 3 ulp
exp2	<= 3 ulp
exp10	<= 3 ulp
expm1	<= 3 ulp
fabs	0 ulp
fdim	Correctly rounded
floor	Correctly rounded
fma	Correctly rounded
fmax	0 ulp
fmin	0 ulp
fmod	0 ulp
fract	Correctly rounded
frexp	0 ulp
hypot	<= 4 ulp
ilogb	0 ulp
ldexp	Correctly rounded
length	<= 5.5 + n ulp, for gentype with vector width *n*
log	<= 3 ulp
log2	<= 3 ulp
log10	<= 3 ulp
log1p	<= 2 ulp
logb	0 ulp
mad	Implementation-defined
max	0 ulp
maxmag	0 ulp
min	0 ulp
minmag	0 ulp
mix	Implementation-defined

Function	Min Accuracy
modf	0 ulp
nan	0 ulp
nextafter	0 ulp
normalize	<= 4.5 + n ulp, for gentype with vector width *n*
pow(x, y)	<= 16 ulp
pown(x, y)	<= 16 ulp
powr(x, y)	<= 16 ulp
radians	<= 2 ulp
remainder	0 ulp
remquo	0 ulp for the remainder, at least the lower 7 bits of the integral quotient
rint	Correctly rounded
rootn	<= 16 ulp
round	Correctly rounded
rsqrt	<= 2 ulp
sign	0 ulp
sin	<= 4 ulp
sincos	<= 4 ulp for sine and cosine values
sinh	<= 4 ulp
sinpi	<= 4 ulp
smoothstep	Implementation-defined
sqrt	Correctly rounded
step	0 ulp
tan	<= 5 ulp
tanh	<= 5 ulp
tanpi	<= 6 ulp
tgamma	<= 16 ulp
trunc	Correctly rounded

Chapter 7. 32-bit Atomics

This section describes the extensions **cl_khr_global_int32_base_atomics**, **cl_khr_global_int32_extended_atomics**, **cl_khr_local_int32_base_atomics**, and **cl_khr_local_int32_extended_atomics**. These extensions allow atomic operations to be performed on 32-bit signed and unsigned integers in global and local memory.

These extensions became core features in OpenCL 1.1, except the built-in atomic function names are changed to use the **atomic_** prefix instead of **atom_** and the volatile qualifier was added to the pointer parameter *p*.

7.1. General Information

7.1.1. Version History

Date	**Version**	**Description**
2020-04-21	1.0.0	First assigned version.

7.2. Global Atomics for 32-bit Integers

7.2.1. Base Atomics

Table 19. Built-in Atomic Functions for **cl_khr_global_int32_base_atomics**

Function	**Description**
int **atom_add** (volatile __global int **p*, int *val*) uint **atom_add** (volatile __global uint **p*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* + *val*) and store result at location pointed by *p*. The function returns *old*.
int **atom_sub** (volatile __global int **p*, int *val*) uint **atom_sub** (volatile __global uint **p*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* - *val*) and store result at location pointed by *p*. The function returns *old*.
int **atom_xchg** (volatile __global int **p*, int *val*) uint **atom_xchg** (volatile __global uint **p*, uint *val*)	Swaps the *old* value stored at location *p* with new value given by *val*. Returns *old* value.
int **atom_inc** (volatile __global int **p*) uint **atom_inc** (volatile __global uint **p*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* + *1*) and store result at location pointed by *p*. The function returns *old*.

Function	Description
int **atom_dec** (volatile __global int **p*) uint **atom_dec** (volatile __global uint **p*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* - *1*) and store result at location pointed by *p*. The function returns *old*.
int **atom_cmpxchg** (volatile __global int **p*, int *cmp*, int *val*) uint **atom_cmpxchg** (volatile __global uint **p*, uint *cmp*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* == *cmp*) ? *val* : *old* and store result at location pointed by *p*. The function returns *old*.

7.2.2. Extended Atomics

Table 20. Built-in Atomic Functions for **cl_khr_global_int32_extended_atomics**

Function	Description
int **atom_min** (volatile __global int **p*, int *val*) uint **atom_min** (volatile __global uint **p*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute **min**(*old*, *val*) and store minimum value at location pointed by *p*. The function returns *old*.
int **atom_max** (volatile __global int **p*, int *val*) uint **atom_max** (volatile __global uint **p*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute **max**(*old*, *val*) and store maximum value at location pointed by *p*. The function returns *old*.
int **atom_and** (volatile __global int **p*, int *val*) uint **atom_and** (volatile __global uint **p*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* & val) and store result at location pointed by *p*. The function returns *old*.
int **atom_or** (volatile __global int **p*, int *val*) uint **atom_or** (volatile __global uint **p*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* | val) and store result at location pointed by *p*. The function returns *old*.
int **atom_xor** (volatile __global int **p*, int *val*) uint **atom_xor** (volatile __global uint **p*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* ^ val) and store result at location pointed by *p*. The function returns *old*.

7.3. Local Atomics for 32-bit Integers

7.3.1. Base Atomics

Table 21. Built-in Atomic Functions for **cl_khr_local_int32_base_atomics**

Function	Description
int **atom_add** (volatile __local int **p*, int *val*) uint **atom_add** (volatile __local uint **p*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* + *val*) and store result at location pointed by *p*. The function returns *old*.
int **atom_sub** (volatile __local int **p*, int *val*) uint **atom_sub** (volatile __local uint **p*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* - *val*) and store result at location pointed by *p*. The function returns *old*.
int **atom_xchg** (volatile __local int **p*, int *val*) uint **atom_xchg** (volatile __local uint **p*, uint *val*)	Swaps the *old* value stored at location *p* with new value given by *val*. Returns *old* value.
int **atom_inc** (volatile __local int **p*) uint **atom_inc** (volatile __local uint **p*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* + *1*) and store result at location pointed by *p*. The function returns *old*.
int **atom_dec** (volatile __local int **p*) uint **atom_dec** (volatile __local uint **p*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* - *1*) and store result at location pointed by *p*. The function returns *old*.
int **atom_cmpxchg** (volatile __local int **p*, int *cmp*, int *val*) uint **atom_cmpxchg** (volatile __local uint **p*, uint *cmp*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* == *cmp*) ? *val* : *old* and store result at location pointed by *p*. The function returns *old*.

7.3.2. Extended Atomics

Table 22. Built-in Atomic Functions for **cl_khr_local_int32_extended_atomics**

Function	Description
int **atom_min** (volatile __local int **p*, int *val*) uint **atom_min** (volatile __local uint **p*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute **min**(*old*, *val*) and store minimum value at location pointed by *p*. The function returns *old*.
int **atom_max** (volatile __local int **p*, int *val*) uint **atom_max** (volatile __local uint **p*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute **max**(*old*, *val*) and store maximum value at location pointed by *p*. The function returns *old*.
int **atom_and** (volatile __local int **p*, int *val*) uint **atom_and** (volatile __local uint **p*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* & val) and store result at location pointed by *p*. The function returns *old*.
int **atom_or** (volatile __local int **p*, int *val*) uint **atom_or** (volatile __local uint **p*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* | val) and store result at location pointed by *p*. The function returns *old*.
int **atom_xor** (volatile __local int **p*, int *val*) uint **atom_xor** (volatile __local uint **p*, uint *val*)	Read the 32-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* ^ val) and store result at location pointed by *p*. The function returns *old*.

Chapter 8. 64-bit Atomics

This section describes the **cl_khr_int64_base_atomics** and **cl_khr_int64_extended_atomics** extensions. These extensions allow atomic operations to be performed on 64-bit signed and unsigned integers in global and local memory.

8.1. General Information

8.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

Table 23. Built-in Atomic Functions for **cl_khr_int64_base_atomics**

Function	Description
long **atom_add** (volatile __global long **p*, long *val*) long **atom_add** (volatile __local long **p*, long *val*) ulong **atom_add** (volatile __global ulong **p*, ulong *val*) ulong **atom_add** (volatile __local ulong **p*, ulong *val*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* + *val*) and store result at location pointed by *p*. The function returns *old*.
long **atom_sub** (volatile __global long **p*, long *val*) long **atom_sub** (volatile __local long **p*, long *val*) ulong **atom_sub** (volatile __global ulong **p*, ulong *val*) ulong **atom_sub** (volatile __local ulong **p*, ulong *val*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* - *val*) and store result at location pointed by *p*. The function returns *old*.
long **atom_xchg** (volatile __global long **p*, long *val*) long **atom_xchg** (volatile __local long **p*, long *val*) ulong **atom_xchg** (volatile __global ulong **p*, ulong *val*) ulong **atom_xchg** (volatile __local ulong **p*, ulong *val*)	Swaps the *old* value stored at location *p* with new value given by *val*. Returns *old* value.
long **atom_inc** (volatile __global long **p*) long **atom_inc** (volatile __local long **p*) ulong **atom_inc** (volatile __global ulong **p*) ulong **atom_inc** (volatile __local ulong **p*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* + *1*) and store result at location pointed by *p*. The function returns *old*.
long **atom_dec** (volatile __global long **p*) long **atom_dec** (volatile __local long **p*) ulong **atom_dec** (volatile __global ulong **p*) ulong **atom_dec** (volatile __local ulong **p*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* - *1*) and store result at location pointed by *p*. The function returns *old*.

Function	Description
long **atom_cmpxchg** (volatile __global long **p*, long *cmp*, long *val*) long **atom_cmpxchg** (volatile __local long **p*, long *cmp*, long *val*) ulong **atom_cmpxchg** (volatile __global ulong **p*, ulong *cmp*, ulong *val*) ulong **atom_cmpxchg** (volatile __local ulong **p*, ulong *cmp*, ulong *val*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* == *cmp*) ? *val* : *old* and store result at location pointed by *p*. The function returns *old*.

Table 24. Built-in Atomic Functions for **cl_khr_int64_extended_atomics**

Function	Description
long **atom_min** (volatile __global long **p*, long *val*) long **atom_min** (volatile __local long **p*, long *val*) ulong **atom_min** (volatile __global ulong **p*, ulong *val*) ulong **atom_min** (volatile __local ulong **p*, ulong *val*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute **min**(*old*, *val*) and store minimum value at location pointed by *p*. The function returns *old*.
long **atom_max** (volatile __global long **p*, long *val*) long **atom_max** (volatile __local long **p*, long *val*) ulong **atom_max** (volatile __global ulong **p*, ulong *val*) ulong **atom_max** (volatile __local ulong **p*, ulong *val*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute **max**(*old*, *val*) and store maximum value at location pointed by *p*. The function returns *old*.
long **atom_and** (volatile __global long **p*, long *val*) long **atom_and** (volatile __local long **p*, long *val*) ulong **atom_and** (volatile __global ulong **p*, ulong *val*) ulong **atom_and** (volatile __local ulong **p*, ulong *val*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* & val) and store result at location pointed by *p*. The function returns *old*.
long **atom_or** (volatile __global long **p*, long *val*) long **atom_or** (volatile __local long **p*, long *val*) ulong **atom_or** (volatile __global ulong **p*, ulong *val*) ulong **atom_or** (volatile __local ulong **p*, ulong *val*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* | val) and store result at location pointed by *p*. The function returns *old*.

Function	Description
long **atom_xor** (volatile __global long *p, long *val*) long **atom_xor** (volatile __local long *p, long *val*) ulong **atom_xor** (volatile __global ulong *p, ulong *val*) ulong **atom_xor** (volatile __local ulong *p, ulong *val*)	Read the 64-bit value (referred to as *old*) stored at location pointed by *p*. Compute (*old* ^ val) and store result at location pointed by *p*. The function returns *old*.

Note: Atomic operations on 64-bit integers and 32-bit integers (and float) are also atomic w.r.t. each other.

Chapter 9. Selecting the Rounding Mode (DEPRECATED)

This section describes the **cl_khr_select_fprounding_mode** extension. It allows an application to specify the rounding mode for an instruction or group of instructions in the program source.

This extension was deprecated in OpenCL 1.1 and its use is not recommended.

9.1. General Information

9.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

9.2. Changes to OpenCL C specification

With this extension, the rounding mode may be specified using the following **#pragma** in the OpenCL program source:

```
#pragma OPENCL SELECT_ROUNDING_MODE <rounding-mode>
```

The <*rounding-mode*> may be one of the following values:

- **rte** - round to nearest even
- **rtz** - round to zero
- **rtp** - round to positive infinity
- **rtn** - round to negative infinity

If this extensions is supported then the OpenCL implementation must support all four rounding modes for single precision floating-point.

The **#pragma** sets the rounding mode for all instructions that operate on floating-point types (scalar or vector types) or produce floating-point values that follow this pragma in the program source until the next **#pragma**. Note that the rounding mode specified for a block of code is known at compile time. When inside a compound statement, the pragma takes effect from its occurrence until another **#pragma** is encountered (including within a nested compound statement), or until the end of the compound statement; at the end of a compound statement the state for the pragma is restored to its condition just before the compound statement. Except where otherwise documented, the callee functions do not inherit the rounding mode of the caller function.

If this extension is enabled, the `__ROUNDING_MODE__` preprocessor symbol shall be defined to be one of the following according to the current rounding mode:

```
#define __ROUNDING_MODE__ rte
#define __ROUNDING_MODE__ rtz
#define __ROUNDING_MODE__ rtp
#define __ROUNDING_MODE__ rtz
```

This is intended to enable remapping `foo()` to `foo_rte()` by the preprocessor by using:

```
#define foo foo ## __ROUNDING_MODE__
```

The default rounding mode is round to nearest even. The built-in math functions described in *section 6.11.2*, the common functions described in *section 6.11.4* and the geometric functions described in *section 6.11.5* are implemented with the round to nearest even rounding mode. Various built-in conversions and the **vstore_half** and **vstorea_half** built-in functions that do not specify a rounding mode inherit the current rounding mode. Conversions from floating-point to integer type always use `rtz` mode, except where the user specifically asks for another rounding mode.

Chapter 10. Creating an OpenCL Context from an OpenGL Context or Share Group

10.1. Overview

This section describes functionality in the **cl_khr_gl_sharing** extension to associate an OpenCL context with an OpenGL context or share group object. Once an OpenCL context is associated with an OpenGL context or share group object, the functionality described in the section Creating OpenCL Memory Objects from OpenGL Objects may be used to share OpenGL buffer, texture, and renderbuffer objects with the OpenCL context.

An OpenGL implementation supporting buffer objects and sharing of texture and buffer object images with OpenCL is required by this extension.

10.2. General Information

10.2.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

10.3. New Procedures and Functions

```
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);
```

10.4. New Tokens

Returned by **clCreateContext**, **clCreateContextFromType**, and **clGetGLContextInfoKHR** when an invalid OpenGL context or share group object handle is specified in *properties*:

```
CL_INVALID_GL_SHAREGROUP_REFERENCE_KHR
```

Accepted as the *param_name* argument of **clGetGLContextInfoKHR**:

```
CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR
CL_DEVICES_FOR_GL_CONTEXT_KHR
```

Accepted as an attribute name in the *properties* argument of **clCreateContext** and **clCreateContextFromType**:

```
CL_GL_CONTEXT_KHR
CL_EGL_DISPLAY_KHR
CL_GLX_DISPLAY_KHR
CL_WGL_HDC_KHR
CL_CGL_SHAREGROUP_KHR
```

10.5. Additions to Chapter 4 of the OpenCL 2.2 Specification

In *section 4.4*, replace the description of *properties* under **clCreateContext** with:

"`*properties* points to an attribute list, which is a array of ordered <attribute name, value> pairs terminated with zero. If an attribute is not specified in *properties*, then its default value (listed in *table 4.5*) is used (it is said to be specified implicitly). If *properties* is `NULL` or empty (points to a list whose first value is zero), all attributes take on their default values.

Attributes control sharing of OpenCL memory objects with OpenGL buffer, texture, and renderbuffer objects. Depending on the platform-specific API used to bind OpenGL contexts to the window system, the following attributes may be set to identify an OpenGL context:

- When the CGL binding API is supported, the attribute CL_CGL_SHAREGROUP_KHR should be set to a CGLShareGroup handle to a CGL share group object.
- When the EGL binding API is supported, the attribute CL_GL_CONTEXT_KHR should be set to an EGLContext handle to an OpenGL ES or OpenGL context, and the attribute CL_EGL_DISPLAY_KHR should be set to the EGLDisplay handle of the display used to create the OpenGL ES or OpenGL context.
- When the GLX binding API is supported, the attribute CL_GL_CONTEXT_KHR should be set to a GLXContext handle to an OpenGL context, and the attribute CL_GLX_DISPLAY_KHR should be set to the Display handle of the X Window System display used to create the OpenGL context.
- When the WGL binding API is supported, the attribute CL_GL_CONTEXT_KHR should be set to an HGLRC handle to an OpenGL context, and the attribute CL_WGL_HDC_KHR should be set to the HDC handle of the display used to create the OpenGL context.

Memory objects created in the context so specified may be shared with the specified OpenGL or OpenGL ES context (as well as with any other OpenGL contexts on the share list of that context, according to the description of sharing in the GLX 1.4 and EGL 1.4 specifications, and the WGL documentation for OpenGL implementations on Microsoft Windows), or with the explicitly identified OpenGL share group for CGL. If no OpenGL or OpenGL ES context or share group is specified in the attribute list, then memory objects may not be shared, and calling any of the commands described in Creating OpenCL Memory Objects from OpenGL Objects will result in a CL_INVALID_GL_SHAREGROUP_REFERENCE_KHR error.`"

OpenCL / OpenGL sharing does not support the CL_CONTEXT_INTEROP_USER_SYNC property defined in *table 4.5*. Specifying this property when creating a context with OpenCL / OpenGL sharing will return an appropriate error.

Add to *table 4.5*:

Table 25. OpenGL Sharing Context Creation Attributes

Attribute Name	**Allowed Values (Default value is in bold)**	**Description**
CL_GL_CONTEXT_KHR	**0**, OpenGL context handle	OpenGL context to associated the OpenCL context with
CL_CGL_SHAREGROUP_KHR	**0**, CGL share group handle	CGL share group to associate the OpenCL context with
CL_EGL_DISPLAY_KHR	**EGL_NO_DISPLAY**, EGLDisplay handle	EGLDisplay an OpenGL context was created with respect to
CL_GLX_DISPLAY_KHR	**None**, X handle	X Display an OpenGL context was created with respect to
CL_WGL_HDC_KHR	**0**, HDC handle	HDC an OpenGL context was created with respect to

Replace the first error in the list for **clCreateContext** with:

"`*errcode_ret* returns CL_INVALID_GL_SHAREGROUP_REFERENCE_KHR if a context was specified by any of the following means:

- A context was specified for an EGL-based OpenGL ES or OpenGL implementation by setting the attributes CL_GL_CONTEXT_KHR and CL_EGL_DISPLAY_KHR.
- A context was specified for a GLX-based OpenGL implementation by setting the attributes CL_GL_CONTEXT_KHR and CL_GLX_DISPLAY_KHR.
- A context was specified for a WGL-based OpenGL implementation by setting the attributes CL_GL_CONTEXT_KHR and CL_WGL_HDC_KHR

and any of the following conditions hold:

- The specified display and context attributes do not identify a valid OpenGL or OpenGL ES context.
- The specified context does not support buffer and renderbuffer objects.
- The specified context is not compatible with the OpenCL context being created (for example, it exists in a physically distinct address space, such as another hardware device; or it does not support sharing data with OpenCL due to implementation restrictions).

errcode_ret returns CL_INVALID_GL_SHAREGROUP_REFERENCE_KHR if a share group was specified for a CGL-based OpenGL implementation by setting the attribute CL_CGL_SHAREGROUP_KHR, and the specified share group does not identify a valid CGL share group object.

errcode_ret returns CL_INVALID_OPERATION if a context was specified as described above and any of the following conditions hold:

- A context or share group object was specified for one of CGL, EGL, GLX, or WGL and the OpenGL implementation does not support that window-system binding API.
- More than one of the attributes CL_CGL_SHAREGROUP_KHR, CL_EGL_DISPLAY_KHR, CL_GLX_DISPLAY_KHR, and CL_WGL_HDC_KHR is set to a non-default value.
- Both of the attributes CL_CGL_SHAREGROUP_KHR and CL_GL_CONTEXT_KHR are set to non-default values.
- Any of the devices specified in the *devices* argument cannot support OpenCL objects which share the data store of an OpenGL object.

errcode_ret returns CL_INVALID_PROPERTY if an attribute name other than those specified in *table 4.5* or if CL_CONTEXT_INTEROP_USER_SYNC is specified in *properties*.`"

Replace the description of *properties* under **clCreateContextFromType** with:

"_properties_ points to an attribute list whose format and valid contents are identical to the **properties** argument of **clCreateContext**."

Replace the first error in the list for **clCreateContextFromType** with the same two new errors described above for **clCreateContext**.

10.6. Additions to Chapter 5 of the OpenCL 2.2 Specification

Add a new section to describe the new API for querying OpenCL devices that support sharing with OpenGL:

"`OpenCL device(s) corresponding to an OpenGL context may be queried. Such a device may not always exist (for example, if an OpenGL context is specified on a GPU not supporting OpenCL command-queues, but which does support shared CL/GL objects), and if it does exist, may change over time. When such a device does exist, acquiring and releasing shared CL/GL objects may be faster on a command-queue corresponding to this device than on command-queues corresponding to other devices available to an OpenCL context.

To query the currently corresponding device, use the function

```
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)
```

properties points to an attribute list whose format and valid contents are identical to the *properties* argument of **clCreateContext**. *properties* must identify a single valid GL context or GL share group object.

param_name is a constant that specifies the device types to query, and must be one of the values shown in the table below.

param_value is a pointer to memory where the result of the query is returned as described in the table below. If *param_value* is NULL, it is ignored.

param_value_size specifies the size in bytes of memory pointed to by *param_value*. This size must be greater than or equal to the size of the return type described in the table below.

param_value_size_ret returns the actual size in bytes of data being queried by *param_value*. If *param_value_size_ret* is NULL, it is ignored.

Table 26. Supported Device Types for **clGetGLContextInfoKHR**

param_name	Return Type	Information returned in param_value
CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR	**cl_device_id**	Return the OpenCL device currently associated with the specified OpenGL context.
CL_DEVICES_FOR_GL_CONTEXT_KHR	**cl_device_id[]**	Return all OpenCL devices which may be associated with the specified OpenGL context.

clGetGLContextInfoKHR returns CL_SUCCESS if the function is executed successfully. If no device(s) exist corresponding to *param_name*, the call will not fail, but the value of *param_value_size_ret* will be zero.

clGetGLContextInfoKHR returns CL_INVALID_GL_SHAREGROUP_REFERENCE_KHR if a context was specified by any of the following means:

- A context was specified for an EGL-based OpenGL ES or OpenGL implementation by setting the attributes CL_GL_CONTEXT_KHR and CL_EGL_DISPLAY_KHR.
- A context was specified for a GLX-based OpenGL implementation by setting the attributes CL_GL_CONTEXT_KHR and CL_GLX_DISPLAY_KHR.
- A context was specified for a WGL-based OpenGL implementation by setting the attributes CL_GL_CONTEXT_KHR and CL_WGL_HDC_KHR.

and any of the following conditions hold:

- The specified display and context attributes do not identify a valid OpenGL or OpenGL ES context.
- The specified context does not support buffer and renderbuffer objects.
- The specified context is not compatible with the OpenCL context being created (for example, it exists in a physically distinct address space, such as another hardware device; or it does not support sharing data with OpenCL due to implementation restrictions).

clGetGLContextInfoKHR returns CL_INVALID_GL_SHAREGROUP_REFERENCE_KHR if a share group was specified for a CGL-based OpenGL implementation by setting the attribute CL_CGL_SHAREGROUP_KHR, and the specified share group does not identify a valid CGL share group object.

clGetGLContextInfoKHR returns CL_INVALID_OPERATION if a context was specified as described above and any of the following conditions hold:

- A context or share group object was specified for one of CGL, EGL, GLX, or WGL and the OpenGL implementation does not support that window-system binding API.
- More than one of the attributes CL_CGL_SHAREGROUP_KHR, CL_EGL_DISPLAY_KHR, CL_GLX_DISPLAY_KHR, and CL_WGL_HDC_KHR is set to a non-default value.
- Both of the attributes CL_CGL_SHAREGROUP_KHR and CL_GL_CONTEXT_KHR are set to non-default values.
- Any of the devices specified in the <devices> argument cannot support OpenCL objects which share the data store of an OpenGL object.

clGetGLContextInfoKHR returns CL_INVALID_VALUE if an attribute name other than those specified in *table 4.5* is specified in *properties*.

Additionally, **clGetGLContextInfoKHR** returns CL_INVALID_VALUE if *param_name* is not one of the values listed in the table *GL context information that can be queried with* **clGetGLContextInfoKHR**, or if the size in bytes specified by *param_value_size* is less than the size of the return type shown in the table and *param_value* is not a `NULL` value; CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device; or CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.`"

10.7. Issues

1. How should the OpenGL context be identified when creating an associated OpenCL context?

   RESOLVED: by using a (display,context handle) attribute pair to identify an arbitrary OpenGL or OpenGL ES context with respect to one of the window-system binding layers EGL, GLX, or WGL, or a share group handle to identify a CGL share group. If a context is specified, it need not be current to the thread calling clCreateContext*.

   A previously suggested approach would use a single boolean attribute CL_USE_GL_CONTEXT_KHR to allow creating a context associated with the currently bound OpenGL context. This may still be implemented as a separate extension, and might allow more efficient acquire/release behavior in the special case where they are being executed in the same thread as the bound GL context used to create the CL context.

2. What should the format of an attribute list be?

   After considerable discussion, we think we can live with a list of <attribute name,value> pairs terminated by zero. The list is passed as 'cl_context_properties **properties*', where cl_context_properties is typedefed to be 'intptr_t' in cl.h.

   This effectively allows encoding all scalar integer, pointer, and handle values in the host API into the argument list and is analogous to the structure and type of EGL attribute lists. `NULL` attribute lists are also allowed. Again as for EGL, any attributes not explicitly passed in the list will take on a defined default value that does something reasonable.

Experience with EGL, GLX, and WGL has shown attribute lists to be a sufficiently flexible and general mechanism to serve the needs of management calls such as context creation. It is not completely general (encoding floating-point and non-scalar attribute values is not straightforward), and other approaches were suggested such as opaque attribute lists with getter/setter methods, or arrays of varadic structures.

3. What's the behavior of an associated OpenGL or OpenCL context when using resources defined by the other associated context, and that context is destroyed?

   RESOLVED: OpenCL objects place a reference on the data store underlying the corresponding GL object when they're created. The GL name corresponding to that data store may be deleted, but the data store itself remains so long as any CL object has a reference to it. However, destroying all GL contexts in the share group corresponding to a CL context results in implementation-dependent behavior when using a corresponding CL object, up to and including program termination.

4. How about sharing with D3D?

   Sharing between D3D and OpenCL should use the same attribute list mechanism, though obviously with different parameters, and be exposed as a similar parallel OpenCL extension. There may be an interaction between that extension and this one since it's not yet clear if it will be possible to create a CL context simultaneously sharing GL and D3D objects.

5. Under what conditions will context creation fail due to sharing?

   RESOLVED: Several cross-platform failure conditions are described (GL context or CGL share group doesn't exist, GL context doesn't support types of GL objects, GL context implementation doesn't allow sharing), but additional failures may result due to implementation-dependent reasons and should be added to this extension as such failures are discovered. Sharing between OpenCL and OpenGL requires integration at the driver internals level.

6. What command-queues can **clEnqueueAcquire/ReleaseGLObjects** be placed on?

   RESOLVED: All command-queues. This restriction is enforced at context creation time. If any device passed to context creation cannot support shared CL/GL objects, context creation will fail with a CL_INVALID_OPERATION error.

7. How can applications determine which command-queue to place an Acquire/Release on?

   RESOLVED: The **clGetGLContextInfoKHR** returns either the CL device currently corresponding to a specified GL context (typically the display it's running on), or a list of all the CL devices the specified context might run on (potentially useful in multiheaded / "virtual screen" environments). This command is not simply placed in Creating OpenCL Memory Objects from OpenGL Objects because it relies on the same property-list method of specifying a GL context introduced by this extension.

   If no devices are returned, it means that the GL context exists on an older GPU not capable of running OpenCL, but still capable of sharing objects between GL running on that GPU and CL running elsewhere.

8. What is the meaning of the CL_DEVICES_FOR_GL_CONTEXT_KHR query?

   RESOLVED: The list of all CL devices that may ever be associated with a specific GL context. On platforms such as MacOS X, the “virtual screen” concept allows multiple GPUs to back a single virtual display. Similar functionality might be implemented on other windowing systems, such as a transparent heterogenous multiheaded X server. Therefore the exact meaning of this query is interpreted relative to the binding layer API in use.

Chapter 11. Creating OpenCL Memory Objects from OpenGL Objects

This section describes functionality in the **cl_khr_gl_sharing** extension to use OpenGL buffer, texture, and renderbuffer objects as OpenCL memory objects. OpenCL memory objects may be created from OpenGL objects if and only if the OpenCL context is associated with an OpenGL context or share group object. The section Creating an OpenCL Context from an OpenGL Context or Share Group describes how to create an OpenCL context associated with an OpenGL context or share group object.

An OpenCL image object may be created from an OpenGL texture or renderbuffer object. An OpenCL buffer object may be created from an OpenGL buffer object.

Any supported OpenGL object defined within the associated OpenGL context or share group object may be shared, with the exception of the default OpenGL objects (i.e. objects named zero), which may not be shared.

11.1. General Information

11.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

11.2. Lifetime of Shared Objects

An OpenCL memory object created from an OpenGL object (hereinafter referred to as a "shared CL/GL object") remains valid as long as the corresponding GL object has not been deleted. If the GL object is deleted through the GL API (e.g. **glDeleteBuffers**, **glDeleteTextures**, or **glDeleteRenderbuffers**), subsequent use of the CL buffer or image object will result in undefined behavior, including but not limited to possible CL errors and data corruption, but may not result in program termination.

The CL context and corresponding command-queues are dependent on the existence of the GL share group object, or the share group associated with the GL context from which the CL context is created. If the GL share group object or all GL contexts in the share group are destroyed, any use of the CL context or command-queue(s) will result in undefined behavior, which may include program termination. Applications should destroy the CL command-queue(s) and CL context before destroying the corresponding GL share group or contexts

11.3. OpenCL Buffer Objects from OpenGL Buffer Objects

The function

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

creates an OpenCL buffer object from an OpenGL buffer object.

context is a valid OpenCL context created from an OpenGL context.

flags is a bit-field that is used to specify usage information. Refer to *table 5.3* for a description of *flags*. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* can be used.

bufobj is the name of a GL buffer object. The data store of the GL buffer object must have have been previously created by calling **glBufferData**, although its contents need not be initialized. The size of the data store will be used to determine the size of the CL buffer object.

errcode_ret will return an appropriate error code as described below. If *errcode_ret* is `NULL`, no error code is returned.

clCreateFromGLBuffer returns a valid non-zero OpenCL buffer object and *errcode_ret* is set to CL_SUCCESS if the buffer object is created successfully. Otherwise, it returns a `NULL` value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context or was not created from a GL context.
- CL_INVALID_VALUE if values specified in *flags* are not valid.
- CL_INVALID_GL_OBJECT if *bufobj* is not a GL buffer object or is a GL buffer object but does not have an existing data store or the size of the buffer is 0.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The size of the GL buffer object data store at the time **clCreateFromGLBuffer** is called will be used as the size of buffer object returned by **clCreateFromGLBuffer**. If the state of a GL buffer object is modified through the GL API (e.g. **glBufferData**) while there exists a corresponding CL buffer object, subsequent use of the CL buffer object will result in undefined behavior.

The **clRetainMemObject** and **clReleaseMemObject** functions can be used to retain and release the buffer object.

The CL buffer object created using clCreateFromGLBuffer can also be used to create a CL 1D image buffer object.

11.4. OpenCL Image Objects from OpenGL Textures

The function

```
cl_mem clCreateFromGLTexture(cl_context context,
                             cl_mem_flags flags,
                             GLenum texture_target,
                             GLint miplevel,
                             GLuint texture,
                             cl_int *errcode_ret)
```

creates the following:

- an OpenCL 2D image object from an OpenGL 2D texture object or a single face of an OpenGL cubemap texture object,
- an OpenCL 2D image array object from an OpenGL 2D texture array object,
- an OpenCL 1D image object from an OpenGL 1D texture object,
- an OpenCL 1D image buffer object from an OpenGL texture buffer object,
- an OpenCL 1D image array object from an OpenGL 1D texture array object,
- an OpenCL 3D image object from an OpenGL 3D texture object.

context is a valid OpenCL context created from an OpenGL context.

flags is a bit-field that is used to specify usage information. Refer to *table 5.3* for a description of *flags*. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* may be used.

texture_target must be one of GL_TEXTURE_1D, GL_TEXTURE_1D_ARRAY, GL_TEXTURE_BUFFER, GL_TEXTURE_2D, GL_TEXTURE_2D_ARRAY, GL_TEXTURE_3D, GL_TEXTURE_CUBE_MAP_POSITIVE_X, GL_TEXTURE_CUBE_MAP_POSITIVE_Y, GL_TEXTURE_CUBE_MAP_POSITIVE_Z, GL_TEXTURE_CUBE_MAP_NEGATIVE_X, GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, or GL_TEXTURE_RECTANGLE (Note: GL_TEXTURE_RECTANGLE requires OpenGL 3.1. Alternatively, GL_TEXTURE_RECTANGLE_ARB may be specified if the OpenGL extension **GL_ARB_texture_rectangle** is supported.). *texture_target* is used only to define the image type of *texture*. No reference to a bound GL texture object is made or implied by this parameter.

miplevel is the mipmap level to be used. If *texture_target* is GL_TEXTURE_BUFFER, *miplevel* must be 0. Note: Implementations may return CL_INVALID_OPERATION for miplevel values > 0.

texture is the name of a GL 1D, 2D, 3D, 1D array, 2D array, cubemap, rectangle or buffer texture object. The texture object must be a complete texture as per OpenGL rules on texture completeness. The *texture* format and dimensions defined by OpenGL for the specified *miplevel* of the texture will be used to create the OpenCL image memory object. Only GL texture objects with an internal format that maps to appropriate image channel order and data type specified in *tables 5.5* and *5.6* may be used to create the OpenCL image memory object.

errcode_ret will return an appropriate error code as described below. If *errcode_ret* is `NULL`, no error code is returned.

clCreateFromGLTexture returns a valid non-zero OpenCL image object and *errcode_ret* is set to

CL_SUCCESS if the image object is created successfully. Otherwise, it returns a `NULL` value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context or was not created from a GL context.
- CL_INVALID_VALUE if values specified in *flags* are not valid or if value specified in *texture_target* is not one of the values specified in the description of *texture_target*.
- CL_INVALID_MIP_LEVEL if *miplevel* is less than the value of $level_{base}$ (for OpenGL implementations) or zero (for OpenGL ES implementations); or greater than the value of q (for both OpenGL and OpenGL ES). $level_{base}$ and q are defined for the texture in *section 3.8.10* (Texture Completeness) of the OpenGL 2.1 specification and *section 3.7.10* of the OpenGL ES 2.0.
- CL_INVALID_MIP_LEVEL if *miplevel* is greather than zero and the OpenGL implementation does not support creating from non-zero mipmap levels.
- CL_INVALID_GL_OBJECT if *texture* is not a GL texture object whose type matches *texture_target*, if the specified *miplevel* of *texture* is not defined, or if the width or height of the specified *miplevel* is zero or if the GL texture object is incomplete.
- CL_INVALID_IMAGE_FORMAT_DESCRIPTOR if the OpenGL texture internal format does not map to a supported OpenCL image format.
- CL_INVALID_OPERATION if *texture* is a GL texture object created with a border width value greater than zero.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

If the state of a GL texture object is modified through the GL API (e.g. **glTexImage2D**, **glTexImage3D** or the values of the texture parameters GL_TEXTURE_BASE_LEVEL or GL_TEXTURE_MAX_LEVEL are modified) while there exists a corresponding CL image object, subsequent use of the CL image object will result in undefined behavior.

The **clRetainMemObject** and **clReleaseMemObject** functions can be used to retain and release the image objects.

11.4.1. List of OpenGL and corresponding OpenCL Image Formats

The table below describes the list of OpenGL texture internal formats and the corresponding OpenCL image formats. If a OpenGL texture object with an internal format from the table below is successfully created by OpenGL, then there is guaranteed to be a mapping to one of the corresponding OpenCL image format(s) in that table. Texture objects created with other OpenGL internal formats may (but are not guaranteed to) have a mapping to an OpenCL image format; if such mappings exist, they are guaranteed to preserve all color components, data types, and at least the number of bits/component actually allocated by OpenGL for that format.

Table 27. OpenGL internal formats and corresponding OpenCL internal formats

GL internal format	CL image format (channel order, channel data type)
GL_RGBA8	CL_RGBA, CL_UNORM_INT8 or CL_BGRA, CL_UNORM_INT8
GL_SRGB8_ALPHA8	CL_sRGBA, CL_UNORM_INT8
GL_RGBA, GL_UNSIGNED_INT_8_8_8_8_REV	CL_RGBA, CL_UNORM_INT8
GL_BGRA, GL_UNSIGNED_INT_8_8_8_8_REV	CL_BGRA, CL_UNORM_INT8
GL_RGBA8I, GL_RGBA8I_EXT	CL_RGBA, CL_SIGNED_INT8
GL_RGBA16I, GL_RGBA16I_EXT	CL_RGBA, CL_SIGNED_INT16
GL_RGBA32I, GL_RGBA32I_EXT	CL_RGBA, CL_SIGNED_INT32
GL_RGBA8UI, GL_RGBA8UI_EXT	CL_RGBA, CL_UNSIGNED_INT8
GL_RGBA16UI, GL_RGBA16UI_EXT	CL_RGBA, CL_UNSIGNED_INT16
GL_RGBA32UI, GL_RGBA32UI_EXT	CL_RGBA, CL_UNSIGNED_INT32
GL_RGBA8_SNORM	CL_RGBA, CL_SNORM_INT8
GL_RGBA16	CL_RGBA, CL_UNORM_INT16
GL_RGBA16_SNORM	CL_RGBA, CL_SNORM_INT16
GL_RGBA16F, GL_RGBA16F_ARB	CL_RGBA, CL_HALF_FLOAT
GL_RGBA32F, GL_RGBA32F_ARB	CL_RGBA, CL_FLOAT
GL_R8	CL_R, CL_UNORM_INT8
GL_R8_SNORM	CL_R, CL_SNORM_INT8
GL_R16	CL_R, CL_UNORM_INT16
GL_R16_SNORM	CL_R, CL_SNORM_INT16
GL_R16F	CL_R, CL_HALF_FLOAT
GL_R32F	CL_R, CL_FLOAT
GL_R8I	CL_R, CL_SIGNED_INT8
GL_R16I	CL_R, CL_SIGNED_INT16
GL_R32I	CL_R, CL_SIGNED_INT32
GL_R8UI	CL_R, CL_UNSIGNED_INT8
GL_R16UI	CL_R, CL_UNSIGNED_INT16
GL_R32UI	CL_R, CL_UNSIGNED_INT32
GL_RG8	CL_RG, CL_UNORM_INT8
GL_RG8_SNORM	CL_RG, CL_SNORM_INT8

GL internal format	CL image format (channel order, channel data type)
GL_RG16	CL_RG, CL_UNORM_INT16
GL_RG16_SNORM	CL_RG, CL_SNORM_INT16
GL_RG16F	CL_RG, CL_HALF_FLOAT
GL_RG32F	CL_RG, CL_FLOAT
GL_RG8I	CL_RG, CL_SIGNED_INT8
GL_RG16I	CL_RG, CL_SIGNED_INT16
GL_RG32I	CL_RG, CL_SIGNED_INT32
GL_RG8UI	CL_RG, CL_UNSIGNED_INT8
GL_RG16UI	CL_RG, CL_UNSIGNED_INT16
GL_RG32UI	CL_RG, CL_UNSIGNED_INT32

11.5. OpenCL Image Objects from OpenGL Renderbuffers

The function

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

creates an OpenCL 2D image object from an OpenGL renderbuffer object.

context is a valid OpenCL context created from an OpenGL context.

flags is a bit-field that is used to specify usage information. Refer to *table 5.3* for a description of *flags*. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* can be used.

renderbuffer is the name of a GL renderbuffer object. The renderbuffer storage must be specified before the image object can be created. The *renderbuffer* format and dimensions defined by OpenGL will be used to create the 2D image object. Only GL renderbuffers with internal formats that maps to appropriate image channel order and data type specified in *tables 5.5* and *5.6* can be used to create the 2D image object.

errcode_ret will return an appropriate error code as described below. If *errcode_ret* is `NULL`, no error code is returned.

clCreateFromGLRenderbuffer returns a valid non-zero OpenCL image object and *errcode_ret* is set to CL_SUCCESS if the image object is created successfully. Otherwise, it returns a `NULL` value with one of the following error values returned in *errcode_ret*:

* CL_INVALID_CONTEXT if *context* is not a valid context or was not created from a GL context.
* CL_INVALID_VALUE if values specified in *flags* are not valid.
* CL_INVALID_GL_OBJECT if *renderbuffer* is not a GL renderbuffer object or if the width or height of *renderbuffer* is zero.
* CL_INVALID_IMAGE_FORMAT_DESCRIPTOR if the OpenGL renderbuffer internal format does not map to a supported OpenCL image format.
* CL_INVALID_OPERATION if *renderbuffer* is a multi-sample GL renderbuffer object.
* CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
* CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

If the state of a GL renderbuffer object is modified through the GL API (i.e. changes to the dimensions or format used to represent pixels of the GL renderbuffer using appropriate GL API calls such as **glRenderbufferStorage**) while there exists a corresponding CL image object, subsequent use of the CL image object will result in undefined behavior.

The **clRetainMemObject** and **clReleaseMemObject** functions can be used to retain and release the image objects.

The table *OpenGL internal formats and corresponding OpenCL internal formats* describes the list of OpenGL renderbuffer internal formats and the corresponding OpenCL image formats. If an OpenGL renderbuffer object with an internal format from the table is successfully created by OpenGL, then there is guaranteed to be a mapping to one of the corresponding OpenCL image format(s) in that table. Renderbuffer objects created with other OpenGL internal formats may (but are not guaranteed to) have a mapping to an OpenCL image format; if such mappings exist, they are guaranteed to preserve all color components, data types, and at least the number of bits/component actually allocated by OpenGL for that format.

11.6. Querying OpenGL object information from an OpenCL memory object

The OpenGL object used to create the OpenCL memory object and information about the object type i.e. whether it is a texture, renderbuffer or buffer object can be queried using the following function.

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

gl_object_type returns the type of GL object attached to *memobj* and can be CL_GL_OBJECT_BUFFER, CL_GL_OBJECT_TEXTURE2D, CL_GL_OBJECT_TEXTURE3D, CL_GL_OBJECT_TEXTURE2D_ARRAY, CL_GL_OBJECT_TEXTURE1D, CL_GL_OBJECT_TEXTURE1D_ARRAY, CL_GL_OBJECT_TEXTURE_BUFFER, or CL_GL_OBJECT_RENDERBUFFER. If *gl_object_type* is `NULL`, it is ignored

gl_object_name returns the GL object name used to create *memobj*. If *gl_object_name* is `NULL`, it is ignored.

clGetGLObjectInfo returns CL_SUCCESS if the call was executed successfully. Otherwise, it returns one of the following errors:

- CL_INVALID_MEM_OBJECT if *memobj* is not a valid OpenCL memory object.
- CL_INVALID_GL_OBJECT if there is no GL object associated with *memobj*.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

```
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)
```

returns additional information about the GL texture object associated with *memobj*.

param_name specifies what additional information about the GL texture object associated with *memobj* to query. The list of supported *param_name* types and the information returned in *param_value* by **clGetGLTextureInfo** is described in the table below.

param_value is a pointer to memory where the result being queried is returned. If *param_value* is `NULL`, it is ignored.

param_value_size is used to specify the size in bytes of memory pointed to by *param_value*. This size must be >= size of return type as described in the table below.

param_value_size_ret returns the actual size in bytes of data copied to *param_value*. If *param_value_size_ret* is `NULL`, it is ignored.

Table 28. OpenGL texture info that may be queried with **clGetGLTextureInfo**

cl_gl_texture_info	**Return Type**	**Info. returned in *param_value***
CL_GL_TEXTURE_TARGET	GLenum	The *texture_target* argument specified in **clCreateFromGLTexture**.
CL_GL_MIPMAP_LEVEL	GLint	The *miplevel* argument specified in **clCreateFromGLTexture**.

clGetGLTextureInfo returns CL_SUCCESS if the function is executed successfully. Otherwise, it returns one of the following errors:

- CL_INVALID_MEM_OBJECT if *memobj* is not a valid OpenCL memory object.
- CL_INVALID_GL_OBJECT if there is no GL texture object associated with *memobj*.
- CL_INVALID_VALUE if *param_name* is not valid, or if size in bytes specified by *param_value_size* is less than the size of the return type as described in the table above and *param_value* is not `NULL`, or if *param_value* and *param_value_size_ret* are `NULL`.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

11.7. Sharing memory objects that map to GL objects between GL and CL contexts

The function

```
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)
```

is used to acquire OpenCL memory objects that have been created from OpenGL objects. These objects need to be acquired before they can be used by any OpenCL commands queued to a command-queue or the behaviour is undefined. The OpenGL objects are acquired by the OpenCL context associated with *command_queue* and can therefore be used by all command-queues associated with the OpenCL context.

command_queue is a valid command-queue. All devices used to create the OpenCL context associated with *command_queue* must support acquiring shared CL/GL objects. This constraint is enforced at context creation time.

num_objects is the number of memory objects to be acquired in *mem_objects*.

mem_objects is a pointer to a list of CL memory objects that correspond to GL objects.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is `NULL`, then this particular command does not wait on any event to complete. If *event_wait_list* is `NULL`, *num_events_in_wait_list* must be 0. If *event_wait_list* is not `NULL`, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in

event_wait_list act as synchronization points.

event returns an event object that identifies this command and can be used to query wait for this command to complete. If *event* is `NULL` or the enqueue is unsuccessful, no event will be created and therefore it will not be possible to query the status of this command or to wait for this command to complete. If *event_wait_list* and *event* are not `NULL`, *event* must not refer to an element of the *event_wait_list* array.

clEnqueueAcquireGLObjects returns CL_SUCCESS if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is `NULL` the function does nothing and returns CL_SUCCESS. Otherwise, it returns one of the following errors:

- CL_INVALID_VALUE if *num_objects* is zero and *mem_objects* is not a `NULL` value or if *num_objects* > 0 and *mem_objects* is `NULL`.
- CL_INVALID_MEM_OBJECT if memory objects in *mem_objects* are not valid OpenCL memory objects.
- CL_INVALID_COMMAND_QUEUE if *command_queue* is not a valid command-queue.
- CL_INVALID_CONTEXT if context associated with *command_queue* was not created from an OpenGL context
- CL_INVALID_GL_OBJECT if memory objects in *mem_objects* have not been created from a GL object(s).
- CL_INVALID_EVENT_WAIT_LIST if *event_wait_list* is `NULL` and *num_events_in_wait_list* > 0, or *event_wait_list* is not `NULL` and *num_events_in_wait_list* is 0, or if event objects in *event_wait_list* are not valid events.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

```
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)
```

is used to release OpenCL memory objects that have been created from OpenGL objects. These objects need to be released before they can be used by OpenGL. The OpenGL objects are released by the OpenCL context associated with *command_queue*.

num_objects is the number of memory objects to be released in *mem_objects*.

mem_objects is a pointer to a list of CL memory objects that correspond to GL objects.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this command can be executed. If *event_wait_list* is `NULL`, then this particular command does not wait on any event to complete. If *event_wait_list* is `NULL`, *num_events_in_wait_list* must be 0. If *event_wait_list* is not `NULL`, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points.

event returns an event object that identifies this command and can be used to query or wait for this command to complete. If *event* is `NULL` or the enqueue is unsuccessful, no event will be created and therefore it will not be possible to query the status of this command or to wait for this command to complete. If *event_wait_list* and *event* are not `NULL`, *event* must not refer to an element of the *event_wait_list* array.

clEnqueueReleaseGLObjects returns CL_SUCCESS if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is `NULL` the function does nothing and returns CL_SUCCESS. Otherwise, it returns one of the following errors:

- CL_INVALID_VALUE if *num_objects* is zero and *mem_objects* is not a `NULL` value or if *num_objects* > 0 and *mem_objects* is `NULL`.
- CL_INVALID_MEM_OBJECT if memory objects in *mem_objects* are not valid OpenCL memory objects.
- CL_INVALID_COMMAND_QUEUE if *command_queue* is not a valid command-queue.
- CL_INVALID_CONTEXT if context associated with *command_queue* was not created from an OpenGL context
- CL_INVALID_GL_OBJECT if memory objects in *mem_objects* have not been created from a GL object(s).
- CL_INVALID_EVENT_WAIT_LIST if *event_wait_list* is `NULL` and *num_events_in_wait_list* > 0, or *event_wait_list* is not `NULL` and *num_events_in_wait_list* is 0, or if event objects in *event_wait_list* are not valid events.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

11.7.1. Synchronizing OpenCL and OpenGL Access to Shared Objects

In order to ensure data integrity, the application is responsible for synchronizing access to shared CL/GL objects by their respective APIs. Failure to provide such synchronization may result in race conditions and other undefined behavior including non-portability between implementations.

Prior to calling **clEnqueueAcquireGLObjects**, the application must ensure that any pending GL operations which access the objects specified in *mem_objects* have completed. This may be accomplished portably by issuing and waiting for completion of a **glFinish** command on all GL contexts with pending references to these objects. Implementations may offer more efficient synchronization methods; for example on some platforms calling **glFlush** may be sufficient, or synchronization may be implicit within a thread, or there may be vendor-specific extensions that enable placing a fence in the GL command stream and waiting for completion of that fence in the CL command-queue. Note that no synchronization methods other than **glFinish** are portable between OpenGL implementations at this time.

Similarly, after calling **clEnqueueReleaseGLObjects**, the application is responsible for ensuring that any pending OpenCL operations which access the objects specified in *mem_objects* have completed prior to executing subsequent GL commands which reference these objects. This may be accomplished portably by calling **clWaitForEvents** with the event object returned by **clEnqueueReleaseGLObjects**, or by calling **clFinish**. As above, some implementations may offer more efficient methods.

The application is responsible for maintaining the proper order of operations if the CL and GL contexts are in separate threads.

If a GL context is bound to a thread other than the one in which **clEnqueueReleaseGLObjects** is called, changes to any of the objects in *mem_objects* may not be visible to that context without additional steps being taken by the application. For an OpenGL 3.1 (or later) context, the requirements are described in Appendix D ("Shared Objects and Multiple Contexts") of the OpenGL 3.1 Specification. For prior versions of OpenGL, the requirements are implementation-dependent.

Attempting to access the data store of an OpenGL object after it has been acquired by OpenCL and before it has been released will result in undefined behavior. Similarly, attempting to access a shared CL/GL object from OpenCL before it has been acquired by the OpenCL command-queue, or after it has been released, will result in undefined behavior.

11.7.2. Event Command Types for Sharing memory objects that map to GL objects

The following table describes the event command types for the OpenCL commands to acquire and release OpenCL memory objects that have been created from OpenGL objects:

Table 29. List of supported event command types

Events Created By	Event Command Type
clEnqueueAcquireGLObjects	`CL_COMMAND_ACQUIRE_GL_OBJECTS`
clEnqueueReleaseGLObjects	`CL_COMMAND_RELEASE_GL_OBJECTS`

Chapter 12. Creating OpenCL Event Objects from OpenGL Sync Objects

12.1. Overview

This section describes the **cl_khr_gl_event** extension. This extension allows creating OpenCL event objects linked to OpenGL fence sync objects, potentially improving efficiency of sharing images and buffers between the two APIs. The companion **GL_ARB_cl_event** extension provides the complementary functionality of creating an OpenGL sync object from an OpenCL event object.

In addition, this extension modifies the behavior of **clEnqueueAcquireGLObjects** and **clEnqueueReleaseGLObjects** to implicitly guarantee synchronization with an OpenGL context bound in the same thread as the OpenCL context.

12.2. General Information

12.2.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

12.3. New Procedures and Functions

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

12.4. New Tokens

Returned by **clGetEventInfo** when *param_name* is CL_EVENT_COMMAND_TYPE:

```
CL_COMMAND_GL_FENCE_SYNC_OBJECT_KHR
```

12.5. Additions to Chapter 5 of the OpenCL 2.2 Specification

Add following to the fourth paragraph of *section 5.11* (prior to the description of **clWaitForEvents**):

“Event objects can also be used to reflect the status of an OpenGL sync object. The sync object in turn refers to a fence command executing in an OpenGL command stream. This provides another method of coordinating sharing of buffers and images between OpenGL and OpenCL.”

Add CL_COMMAND_GL_FENCE_SYNC_OBJECT_KHR to the valid *param_value* values returned by **clGetEventInfo** for *param_name* CL_EVENT_COMMAND_TYPE (in the third row and third column of *table 5.22*).

Add new *subsection 5.11.1*:

"`**5.11.1 Linking Event Objects to OpenGL Synchronization Objects**

An event object may be created by linking to an OpenGL **sync object**. Completion of such an event object is equivalent to waiting for completion of the fence command associated with the linked GL sync object.

The function

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

creates a linked event object.

context is a valid OpenCL context created from an OpenGL context or share group, using the **cl_khr_gl_sharing** extension.

sync is the name of a sync object in the GL share group associated with *context*.

clCreateEventFromGLsyncKHR returns a valid OpenCL event object and *errcode_ret* is set to CL_SUCCESS if the event object is created successfully. Otherwise, it returns a `NULL` value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context, or was not created from a GL context.
- CL_INVALID_GL_OBJECT if *sync* is not the name of a sync object in the GL share group associated with *context*.

The parameters of an event object linked to a GL sync object will return the following values when queried with **clGetEventInfo**:

- The CL_EVENT_COMMAND_QUEUE of a linked event is `NULL`, because the event is not associated with any OpenCL command-queue.
- The CL_EVENT_COMMAND_TYPE of a linked event is CL_COMMAND_GL_FENCE_SYNC_OBJECT_KHR, indicating that the event is associated with a GL sync object, rather than an OpenCL command.
- The CL_EVENT_COMMAND_EXECUTION_STATUS of a linked event is either CL_SUBMITTED, indicating that the fence command associated with the sync object has not yet completed, or CL_COMPLETE, indicating that the fence command has completed.

clCreateEventFromGLsyncKHR performs an implicit **clRetainEvent** on the returned event object. Creating a linked event object also places a reference on the linked GL sync object. When the event object is deleted, the reference will be removed from the GL sync object.

Events returned from **clCreateEventFromGLsyncKHR** can be used in the *event_wait_list* argument to **clEnqueueAcquireGLObjects** and CL APIs that take a cl_event as an argument but do not enqueue commands. Passing such events to any other CL API that enqueues commands will generate a CL_INVALID_EVENT error.`"

12.6. Additions to the OpenCL Extension Specification

Add following the paragraph describing parameter *event* to **clEnqueueAcquireGLObjects**:

"`If an OpenGL context is bound to the current thread, then any OpenGL commands which

1. affect or access the contents of a memory object listed in the *mem_objects* list, and
2. were issued on that OpenGL context prior to the call to **clEnqueueAcquireGLObjects**

will complete before execution of any OpenCL commands following the **clEnqueueAcquireGLObjects** which affect or access any of those memory objects. If a non-`NULL` *event* object is returned, it will report completion only after completion of such OpenGL commands.`"

Add following the paragraph describing parameter *event* to **clEnqueueReleaseGLObjects**:

"`If an OpenGL context is bound to the current thread, then then any OpenGL commands which

1. affect or access the contents of the memory objects listed in the *mem_objects* list, and
2. are issued on that context after the call to **clEnqueueReleaseGLObjects**

will not execute until after execution of any OpenCL commands preceding the

clEnqueueReleaseGLObjects which affect or access any of those memory objects. If a non-`NULL` *event* object is returned, it will report completion before execution of such OpenGL commands.`"

Replace the second paragraph of Synchronizing OpenCL and OpenGL Access to Shared Objects with:

"`Prior to calling **clEnqueueAcquireGLObjects**, the application must ensure that any pending OpenGL operations which access the objects specified in *mem_objects* have completed.

If the **cl_khr_gl_event** extension is supported, then the OpenCL implementation will ensure that any such pending OpenGL operations are complete for an OpenGL context bound to the same thread as the OpenCL context. This is referred to as *implicit synchronization*.

If the **cl_khr_gl_event** extension is supported and the OpenGL context in question supports fence sync objects, completion of OpenGL commands may also be determined by placing a GL fence command after those commands using **glFenceSync**, creating an event from the resulting GL sync object using **clCreateEventFromGLsyncKHR**, and determining completion of that event object via **clEnqueueAcquireGLObjects**. This method may be considerably more efficient than calling **glFinish**, and is referred to as *explicit synchronization*. Explicit synchronization is most useful when an OpenGL context bound to another thread is accessing the memory objects.

If the **cl_khr_gl_event** extension is not supported, completion of OpenGL commands may be determined by issuing and waiting for completion of a **glFinish** command on all OpenGL contexts with pending references to these objects. Some implementations may offer other efficient synchronization methods. If such methods exist they will be described in platform-specific documentation.

Note that no synchronization method other than **glFinish** is portable between all OpenGL implementations and all OpenCL implementations. While this is the only way to ensure completion that is portable to all platforms, **glFinish** is an expensive operation and its use should be avoided if the **cl_khr_gl_event** extension is supported on a platform.`"

12.7. Issues

1. How are references between CL events and GL syncs handled?

   PROPOSED: The linked CL event places a single reference on the GL sync object. That reference is removed when the CL event is deleted. A more expensive alternative would be to reflect changes in the CL event reference count through to the GL sync.

2. How are linkages to synchronization primitives in other APIs handled?

   UNRESOLVED. We will at least want to have a way to link events to EGL sync objects. There is probably no analogous DX concept. There would be an entry point for each type of synchronization primitive to be linked to, such as clCreateEventFromEGLSyncKHR.

   An alternative is a generic clCreateEventFromExternalEvent taking an attribute list. The attribute list would include information defining the type of the external primitive and additional information (GL sync object handle, EGL display and sync object handle, etc.) specific to that type. This allows a single entry point to be reused.

   These will probably be separate extensions following the API proposed here.

3. Should the CL_EVENT_COMMAND_TYPE correspond to the type of command (fence) or the type of the linked sync object?

   PROPOSED: To the type of the linked sync object.

4. Should we support both explicit and implicit synchronization?

   PROPOSED: Yes. Implicit synchronization is suitable when GL and CL are executing in the same application thread. Explicit synchronization is suitable when they are executing in different threads but the expense of glFinish is too high.

5. Should this be a platform or device extension?

   PROPOSED: Platform extension. This may result in considerable under-the-hood work to implement the sync→event semantics using only the public GL API, however, when multiple drivers and devices with different GL support levels coexist in the same runtime.

6. Where can events generated from GL syncs be usable?

   PROPOSED: Only with clEnqueueAcquireGLObjects, and attempting to use such an event elsewhere will generate an error. There is no apparent use case for using such events elsewhere, and possibly some cost to supporting it, balanced by the cost of checking the source of events in all other commands accepting them as parameters.

Chapter 13. Creating OpenCL Memory Objects from Direct3D 10 Buffers and Textures

13.1. Overview

This section describes the **cl_khr_d3d10_sharing** extension. The goal of this extension is to provide interoperability between OpenCL and Direct3D 10.

13.2. General Information

13.2.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

13.3. New Procedures and Functions

```
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);

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

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

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

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);
```

13.4. New Tokens

Accepted as a Direct3D 10 device source in the *d3d_device_source* parameter of **clGetDeviceIDsFromD3D10KHR**:

```
CL_D3D10_DEVICE_KHR
CL_D3D10_DXGI_ADAPTER_KHR
```

Accepted as a set of Direct3D 10 devices in the *d3d_device_set* parameter of **clGetDeviceIDsFromD3D10KHR**:

```
CL_PREFERRED_DEVICES_FOR_D3D10_KHR
CL_ALL_DEVICES_FOR_D3D10_KHR
```

Accepted as a property name in the *properties* parameter of **clCreateContext** and **clCreateContextFromType**:

```
CL_CONTEXT_D3D10_DEVICE_KHR
```

Accepted as a property name in the *param_name* parameter of **clGetContextInfo**:

```
CL_CONTEXT_D3D10_PREFER_SHARED_RESOURCES_KHR
```

Accepted as the property being queried in the *param_name* parameter of **clGetMemObjectInfo**:

```
CL_MEM_D3D10_RESOURCE_KHR
```

Accepted as the property being queried in the *param_name* parameter of **clGetImageInfo**:

```
CL_IMAGE_D3D10_SUBRESOURCE_KHR
```

Returned in the *param_value* parameter of **clGetEventInfo** when *param_name* is `CL_EVENT_COMMAND_TYPE`:

```
CL_COMMAND_ACQUIRE_D3D10_OBJECTS_KHR
CL_COMMAND_RELEASE_D3D10_OBJECTS_KHR
```

Returned by **clCreateContext** and **clCreateContextFromType** if the Direct3D 10 device specified for interoperability is not compatible with the devices against which the context is to be created:

```
CL_INVALID_D3D10_DEVICE_KHR
```

Returned by **clCreateFromD3D10BufferKHR** when *resource* is not a Direct3D 10 buffer object, and by **clCreateFromD3D10Texture2DKHR** and **clCreateFromD3D10Texture3DKHR** when *resource* is not a Direct3D 10 texture object:

```
CL_INVALID_D3D10_RESOURCE_KHR
```

Returned by **clEnqueueAcquireD3D10ObjectsKHR** when any of *mem_objects* are currently acquired by OpenCL:

```
CL_D3D10_RESOURCE_ALREADY_ACQUIRED_KHR
```

Returned by **clEnqueueReleaseD3D10ObjectsKHR** when any of *mem_objects* are not currently acquired by OpenCL:

```
CL_D3D10_RESOURCE_NOT_ACQUIRED_KHR
```

13.5. Additions to Chapter 4 of the OpenCL 2.2 Specification

In *section 4.4*, replace the description of *properties* under **clCreateContext** with:

"*properties* specifies a list of context property names and their corresponding values. Each property is followed immediately by the corresponding desired value. The list is terminated with zero. If a property is not specified in *properties*, then its default value (listed in *table 4.5*) is used (it is said to be specified implicitly). If *properties* is `NULL` or empty (points to a list whose first value is zero), all attributes take on their default values."

Add the following to *table 4.5*:

cl_context_properties enum	Property value	Description
CL_CONTEXT_D3D10_DEVICE_KHR	ID3D10Device *	Specifies the ID3D10Device * to use for Direct3D 10 interoperability. The default value is NULL.

Add to the list of errors for **clCreateContext**:

- CL_INVALID_D3D10_DEVICE_KHR if the value of the property CL_CONTEXT_D3D10_DEVICE_KHR is non-NULL and does not specify a valid Direct3D 10 device with which the *cl_device_ids* against which this context is to be created may interoperate.
- CL_INVALID_OPERATION if Direct3D 10 interoperability is specified by setting CL_INVALID_D3D10_DEVICE_KHR to a non-NULL value, and interoperability with another graphics API is also specified.

Add to the list of errors for **clCreateContextFromType** the same new errors described above for **clCreateContext**.

Add the following row to *table 4.6*:

cl_context_info	Return Type	Information returned in param_value
CL_CONTEXT_D3D10_PREFER_SHARED_RESOURCES_KHR	cl_bool	Returns CL_TRUE if Direct3D 10 resources created as shared by setting *MiscFlags* to include D3D10_RESOURCE_MISC_SHARED will perform faster when shared with OpenCL, compared with resources which have not set this flag. Otherwise returns CL_FALSE.

13.6. Additions to Chapter 5 of the OpenCL 2.2 Specification

Add to the list of errors for **clGetMemObjectInfo**:

- CL_INVALID_D3D10_RESOURCE_KHR if *param_name* is CL_MEM_D3D10_RESOURCE_KHR and *memobj* was not created by the function **clCreateFromD3D10BufferKHR**, **clCreateFromD3D10Texture2DKHR**, or **clCreateFromD3D10Texture3DKHR**.

Extend *table 5.12* to include the following entry.

cl_mem_info	Return type	Info. returned in *param_value*
CL_MEM_D3D10_RESOURCE_KHR	ID3D10Resource *	If *memobj* was created using **clCreateFromD3D10BufferKHR**, **clCreateFromD3D10Texture2DKHR**, or **clCreateFromD3D10Texture3DKHR**, returns the *resource* argument specified when *memobj* was created.

Add to the list of errors for **clGetImageInfo**:

- CL_INVALID_D3D10_RESOURCE_KHR if *param_name* is CL_IMAGE_D3D10_SUBRESOURCE_KHR and *image* was not created by the function **clCreateFromD3D10Texture2DKHR**, or **clCreateFromD3D10Texture3DKHR**.

Extend *table 5.9* to include the following entry.

cl_image_info	Return type	Info. returned in *param_value*
CL_IMAGE_D3D10_SUBRESOURCE_KHR	UINT	If *image* was created using **clCreateFromD3D10Texture2DKHR**, or **clCreateFromD3D10Texture3DKHR**, returns the *subresource* argument specified when *image* was created.

Add to *table 5.22* in the **Info returned in <param_value>** column for *cl_event_info* = CL_EVENT_COMMAND_TYPE:

```
CL_COMMAND_ACQUIRE_D3D10_OBJECTS_KHR
CL_COMMAND_RELEASE_D3D10_OBJECTS_KHR
```

13.7. Sharing Memory Objects with Direct3D 10 Resources

This section discusses OpenCL functions that allow applications to use Direct3D 10 resources as OpenCL memory objects. This allows efficient sharing of data between OpenCL and Direct3D 10. The OpenCL API may be used to execute kernels that read and/or write memory objects that are also Direct3D 10 resources. An OpenCL image object may be created from a Direct3D 10 texture resource. An OpenCL buffer object may be created from a Direct3D 10 buffer resource. OpenCL memory objects may be created from Direct3D 10 objects if and only if the OpenCL context has been created from a Direct3D 10 device.

13.7.1. Querying OpenCL Devices Corresponding to Direct3D 10 Devices

The OpenCL devices corresponding to a Direct3D 10 device may be queried. The OpenCL devices corresponding to a DXGI adapter may also be queried. The OpenCL devices corresponding to a Direct3D 10 device will be a subset of the OpenCL devices corresponding to the DXGI adapter against which the Direct3D 10 device was created.

The OpenCL devices corresponding to a Direct3D 10 device or a DXGI device may be queried using the function

```
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);
```

platform refers to the platform ID returned by **clGetPlatformIDs**.

d3d_device_source specifies the type of *d3d_object*, and must be one of the values shown in the table below.

d3d_object specifies the object whose corresponding OpenCL devices are being queried. The type of *d3d_object* must be as specified in the table below.

d3d_device_set specifies the set of devices to return, and must be one of the values shown in the table below.

num_entries is the number of `cl_device_id` entries that can be added to *devices*. If *devices* is not `NULL` then *num_entries* must be greater than zero.

devices returns a list of OpenCL devices found. The `cl_device_id` values returned in *devices* can be used to identify a specific OpenCL device. If *devices* is `NULL`, this argument is ignored. The number of OpenCL devices returned is the minimum of the value specified by *num_entries* and the number of OpenCL devices corresponding to *d3d_object*.

num_devices returns the number of OpenCL devices available that correspond to *d3d_object*. If *num_devices* is `NULL`, this argument is ignored.

clGetDeviceIDsFromD3D10KHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise it may return

- `CL_INVALID_PLATFORM` if *platform* is not a valid platform.
- `CL_INVALID_VALUE` if *d3d_device_source* is not a valid value, *d3d_device_set* is not a valid value, *num_entries* is equal to zero and *devices* is not `NULL`, or if both *num_devices* and *devices* are `NULL`.
- `CL_DEVICE_NOT_FOUND` if no OpenCL devices that correspond to *d3d_object* were found.

Table 30. Direct3D 10 object types that may be used by **clGetDeviceIDsFromD3D10KHR**

cl_d3d10_device_source_khr	**Type of** *d3d_object*
CL_D3D10_DEVICE_KHR	ID3D10Device *
CL_D3D10_DXGI_ADAPTER_KHR	IDXGIAdapter *

Table 31. Sets of devices queriable using **clGetDeviceIDsFromD3D10KHR**

cl_d3d10_device_set_khr	**Devices returned in** *devices*
CL_PREFERRED_DEVICES_FOR_D3D10_KHR	The preferred OpenCL devices associated with the specified Direct3D object.
CL_ALL_DEVICES_FOR_D3D10_KHR	All OpenCL devices which may interoperate with the specified Direct3D object. Performance of sharing data on these devices may be considerably less than on the preferred devices.

13.7.2. Lifetime of Shared Objects

An OpenCL memory object created from a Direct3D 10 resource remains valid as long as the corresponding Direct3D 10 resource has not been deleted. If the Direct3D 10 resource is deleted through the Direct3D 10 API, subsequent use of the OpenCL memory object will result in undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

The successful creation of a cl_context against a Direct3D 10 device specified via the context create parameter CL_CONTEXT_D3D10_DEVICE_KHR will increment the internal Direct3D reference count on the specified Direct3D 10 device. The internal Direct3D reference count on that Direct3D 10 device will be decremented when the OpenCL reference count on the returned OpenCL context drops to zero.

The OpenCL context and corresponding command-queues are dependent on the existence of the Direct3D 10 device from which the OpenCL context was created. If the Direct3D 10 device is deleted through the Direct3D 10 API, subsequent use of the OpenCL context will result in undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

13.7.3. Sharing Direct3D 10 Buffer Resources as OpenCL Buffer Objects

The function

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

creates an OpenCL buffer object from a Direct3D 10 buffer.

context is a valid OpenCL context created from a Direct3D 10 device.

flags is a bit-field that is used to specify usage information. Refer to *table 5.3* for a description of *flags*. Only `CL_MEM_READ_ONLY`, `CL_MEM_WRITE_ONLY` and `CL_MEM_READ_WRITE` values specified in *table 5.3* can be used.

resource is a pointer to the Direct3D 10 buffer to share.

errcode_ret will return an appropriate error code. If *errcode_ret* is `NULL`, no error code is returned.

clCreateFromD3D10BufferKHR returns a valid non-zero OpenCL buffer object and *errcode_ret* is set to `CL_SUCCESS` if the buffer object is created successfully. Otherwise, it returns a `NULL` value with one of the following error values returned in *errcode_ret*:

- `CL_INVALID_CONTEXT` if *context* is not a valid context.
- `CL_INVALID_VALUE` if values specified in *flags* are not valid.
- `CL_INVALID_D3D10_RESOURCE_KHR` if *resource* is not a Direct3D 10 buffer resource, if *resource* was created with the D3D10_USAGE flag D3D10_USAGE_IMMUTABLE, if a `cl_mem` from *resource* has already been created using **clCreateFromD3D10BufferKHR**, or if *context* was not created against the same Direct3D 10 device from which *resource* was created.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

The size of the returned OpenCL buffer object is the same as the size of *resource*. This call will increment the internal Direct3D reference count on *resource*. The internal Direct3D reference count on *resource* will be decremented when the OpenCL reference count on the returned OpenCL memory object drops to zero.

13.7.4. Sharing Direct3D 10 Texture and Resources as OpenCL Image Objects

The function

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

creates an OpenCL 2D image object from a subresource of a Direct3D 10 2D texture.

context is a valid OpenCL context created from a Direct3D 10 device.

flags is a bit-field that is used to specify usage information. Refer to *table 5.3* for a description of *flags*. Only `CL_MEM_READ_ONLY`, `CL_MEM_WRITE_ONLY` and `CL_MEM_READ_WRITE` values specified in *table 5.3* can be used.

resource is a pointer to the Direct3D 10 2D texture to share.

subresource is the subresource of *resource* to share.

errcode_ret will return an appropriate error code. If *errcode_ret* is NULL, no error code is returned.

clCreateFromD3D10Texture2DKHR returns a valid non-zero OpenCL image object and *errcode_ret* is set to CL_SUCCESS if the image object is created successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context.
- CL_INVALID_VALUE if values specified in *flags* are not valid or if *subresource* is not a valid subresource index for *resource*.
- CL_INVALID_D3D10_RESOURCE_KHR if *resource* is not a Direct3D 10 texture resource, if *resource* was created with the D3D10_USAGE flag D3D10_USAGE_IMMUTABLE, if *resource* is a multisampled texture, if a cl_mem from subresource *subresource* of *resource* has already been created using **clCreateFromD3D10Texture2DKHR**, or if *context* was not created against the same Direct3D 10 device from which *resource* was created.
- CL_INVALID_IMAGE_FORMAT_DESCRIPTOR if the Direct3D 10 texture format of *resource* is not listed in the table *Direct3D 10 formats and corresponding OpenCL image formats* or if the Direct3D 10 texture format of *resource* does not map to a supported OpenCL image format.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The width and height of the returned OpenCL 2D image object are determined by the width and height of subresource *subresource* of *resource*. The channel type and order of the returned OpenCL 2D image object is determined by the format of *resource* by the table *Direct3D 10 formats and corresponding OpenCL image formats*.

This call will increment the internal Direct3D reference count on *resource*. The internal Direct3D reference count on *resource* will be decremented when the OpenCL reference count on the returned OpenCL memory object drops to zero.

The function

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

creates an OpenCL 3D image object from a subresource of a Direct3D 10 3D texture.

context is a valid OpenCL context created from a Direct3D 10 device.

flags is a bit-field that is used to specify usage information. Refer to table 5.3 for a description of *flags*. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* can be used.

resource is a pointer to the Direct3D 10 3D texture to share.

subresource is the subresource of *resource* to share.

errcode_ret will return an appropriate error code. If *errcode_ret* is `NULL`, no error code is returned.

clCreateFromD3D10Texture3DKHR returns a valid non-zero OpenCL image object and *errcode_ret* is set to `CL_SUCCESS` if the image object is created successfully. Otherwise, it returns a `NULL` value with one of the following error values returned in *errcode_ret*:

- `CL_INVALID_CONTEXT` if *context* is not a valid context.
- `CL_INVALID_VALUE` if values specified in *flags* are not valid or if *subresource* is not a valid subresource index for *resource*.
- `CL_INVALID_D3D10_RESOURCE_KHR` if *resource* is not a Direct3D 10 texture resource, if *resource* was created with the D3D10_USAGE flag D3D10_USAGE_IMMUTABLE, if *resource* is a multisampled texture, if a `cl_mem` from subresource *subresource* of *resource* has already been created using **clCreateFromD3D10Texture3DKHR**, or if *context* was not created against the same Direct3D 10 device from which *resource* was created.
- `CL_INVALID_IMAGE_FORMAT_DESCRIPTOR` if the Direct3D 10 texture format of *resource* is not listed in the table *Direct3D 10 formats and corresponding OpenCL image formats* or if the Direct3D 10 texture format of *resource* does not map to a supported OpenCL image format.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

The width, height and depth of the returned OpenCL 3D image object are determined by the width, height and depth of subresource *subresource* of *resource*. The channel type and order of the returned OpenCL 3D image object is determined by the format of *resource* by the table *Direct3D 10 formats and corresponding OpenCL image formats*.

This call will increment the internal Direct3D reference count on *resource*. The internal Direct3D reference count on *resource* will be decremented when the OpenCL reference count on the returned OpenCL memory object drops to zero.

Table 32. Direct3D 10 formats and corresponding OpenCL image formats

DXGI format	CL image format (channel order, channel data type)
DXGI_FORMAT_R32G32B32A32_FLOAT	`CL_RGBA`, `CL_FLOAT`
DXGI_FORMAT_R32G32B32A32_UINT	`CL_RGBA`, `CL_UNSIGNED_INT32`
DXGI_FORMAT_R32G32B32A32_SINT	`CL_RGBA`, `CL_SIGNED_INT32`
DXGI_FORMAT_R16G16B16A16_FLOAT	`CL_RGBA`, `CL_HALF_FLOAT`
DXGI_FORMAT_R16G16B16A16_UNORM	`CL_RGBA`, `CL_UNORM_INT16`
DXGI_FORMAT_R16G16B16A16_UINT	`CL_RGBA`, `CL_UNSIGNED_INT16`
DXGI_FORMAT_R16G16B16A16_SNORM	`CL_RGBA`, `CL_SNORM_INT16`
DXGI_FORMAT_R16G16B16A16_SINT	`CL_RGBA`, `CL_SIGNED_INT16`

DXGI format	CL image format (channel order, channel data type)
DXGI_FORMAT_B8G8R8A8_UNORM	`CL_BGRA`, `CL_UNORM_INT8`
DXGI_FORMAT_R8G8B8A8_UNORM	`CL_RGBA`, `CL_UNORM_INT8`
DXGI_FORMAT_R8G8B8A8_UINT	`CL_RGBA`, `CL_UNSIGNED_INT8`
DXGI_FORMAT_R8G8B8A8_SNORM	`CL_RGBA`, `CL_SNORM_INT8`
DXGI_FORMAT_R8G8B8A8_SINT	`CL_RGBA`, `CL_SIGNED_INT8`
DXGI_FORMAT_R32G32_FLOAT	`CL_RG`, `CL_FLOAT`
DXGI_FORMAT_R32G32_UINT	`CL_RG`, `CL_UNSIGNED_INT32`
DXGI_FORMAT_R32G32_SINT	`CL_RG`, `CL_SIGNED_INT32`
DXGI_FORMAT_R16G16_FLOAT	`CL_RG`, `CL_HALF_FLOAT`
DXGI_FORMAT_R16G16_UNORM	`CL_RG`, `CL_UNORM_INT16`
DXGI_FORMAT_R16G16_UINT	`CL_RG`, `CL_UNSIGNED_INT16`
DXGI_FORMAT_R16G16_SNORM	`CL_RG`, `CL_SNORM_INT16`
DXGI_FORMAT_R16G16_SINT	`CL_RG`, `CL_SIGNED_INT16`
DXGI_FORMAT_R8G8_UNORM	`CL_RG`, `CL_UNORM_INT8`
DXGI_FORMAT_R8G8_UINT	`CL_RG`, `CL_UNSIGNED_INT8`
DXGI_FORMAT_R8G8_SNORM	`CL_RG`, `CL_SNORM_INT8`
DXGI_FORMAT_R8G8_SINT	`CL_RG`, `CL_SIGNED_INT8`
DXGI_FORMAT_R32_FLOAT	`CL_R`, `CL_FLOAT`
DXGI_FORMAT_R32_UINT	`CL_R`, `CL_UNSIGNED_INT32`
DXGI_FORMAT_R32_SINT	`CL_R`, `CL_SIGNED_INT32`
DXGI_FORMAT_R16_FLOAT	`CL_R`, `CL_HALF_FLOAT`
DXGI_FORMAT_R16_UNORM	`CL_R`, `CL_UNORM_INT16`
DXGI_FORMAT_R16_UINT	`CL_R`, `CL_UNSIGNED_INT16`
DXGI_FORMAT_R16_SNORM	`CL_R`, `CL_SNORM_INT16`
DXGI_FORMAT_R16_SINT	`CL_R`, `CL_SIGNED_INT16`
DXGI_FORMAT_R8_UNORM	`CL_R`, `CL_UNORM_INT8`
DXGI_FORMAT_R8_UINT	`CL_R`, `CL_UNSIGNED_INT8`
DXGI_FORMAT_R8_SNORM	`CL_R`, `CL_SNORM_INT8`
DXGI_FORMAT_R8_SINT	`CL_R`, `CL_SIGNED_INT8`

13.7.5. Querying Direct3D properties of memory objects created from Direct3D 10 resources

Properties of Direct3D 10 objects may be queried using **clGetMemObjectInfo** and **clGetImageInfo** with *param_name* CL_MEM_D3D10_RESOURCE_KHR and CL_IMAGE_D3D10_SUBRESOURCE_KHR respectively as described in *sections 5.4.3* and *5.3.6*.

13.7.6. Sharing memory objects created from Direct3D 10 resources between Direct3D 10 and OpenCL contexts

The function

```
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);
```

is used to acquire OpenCL memory objects that have been created from Direct3D 10 resources. The Direct3D 10 objects are acquired by the OpenCL context associated with *command_queue* and can therefore be used by all command-queues associated with the OpenCL context.

OpenCL memory objects created from Direct3D 10 resources must be acquired before they can be used by any OpenCL commands queued to a command-queue. If an OpenCL memory object created from a Direct3D 10 resource is used while it is not currently acquired by OpenCL, the behavior is undefined. Implementations may fail the execution of commands attempting to use that OpenCL memory object and set their associated event's execution status to CL_D3D10_RESOURCE_NOT_ACQUIRED_KHR.

If CL_CONTEXT_INTEROP_USER_SYNC is not specified as CL_TRUE during context creation, **clEnqueueAcquireD3D10ObjectsKHR** provides the synchronization guarantee that any Direct3D 10 calls involving the interop device(s) used in the OpenCL context made before **clEnqueueAcquireD3D10ObjectsKHR** is called will complete executing before *event* reports completion and before the execution of any subsequent OpenCL work issued in *command_queue* begins. If the context was created with properties specifying CL_CONTEXT_INTEROP_USER_SYNC as CL_TRUE, the user is responsible for guaranteeing that any Direct3D 10 calls involving the interop device(s) used in the OpenCL context made before **clEnqueueAcquireD3D10ObjectsKHR** is called have completed before calling **clEnqueueAcquireD3D10ObjectsKHR**.

command_queue is a valid command-queue.

num_objects is the number of memory objects to be acquired in *mem_objects*.

mem_objects is a pointer to a list of OpenCL memory objects that were created from Direct3D 10 resources.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is NULL, then this particular command does not wait on any event to complete. If *event_wait_list* is NULL, *num_events_in_wait_list* must be 0. If *event_wait_list* is not NULL, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points.

event returns an event object that identifies this command and can be used to query or wait for this command to complete. If *event* is NULL or the enqueue is unsuccessful, no event will be created and therefore it will not be possible to query the status of this command or to wait for this command to complete. If *event_wait_list* and *event* are not NULL, *event* must not refer to an element of the *event_wait_list* array.

clEnqueueAcquireD3D10ObjectsKHR returns CL_SUCCESS if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is NULL then the function does nothing and returns CL_SUCCESS. Otherwise it returns one of the following errors:

- CL_INVALID_VALUE if *num_objects* is zero and *mem_objects* is not a NULL value or if *num_objects* > 0 and *mem_objects* is NULL.
- CL_INVALID_MEM_OBJECT if memory objects in *mem_objects* are not valid OpenCL memory objects or if memory objects in *mem_objects* have not been created from Direct3D 10 resources.
- CL_INVALID_COMMAND_QUEUE if *command_queue* is not a valid command-queue.
- CL_INVALID_CONTEXT if context associated with *command_queue* was not created from an Direct3D 10 context.
- CL_D3D10_RESOURCE_ALREADY_ACQUIRED_KHR if memory objects in *mem_objects* have previously been acquired using **clEnqueueAcquireD3D10ObjectsKHR** but have not been released using **clEnqueueReleaseD3D10ObjectsKHR**.
- CL_INVALID_EVENT_WAIT_LIST if *event_wait_list* is NULL and *num_events_in_wait_list* > 0, or *event_wait_list* is not NULL and *num_events_in_wait_list* is 0, or if event objects in *event_wait_list* are not valid events.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

```
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);
```

is used to release OpenCL memory objects that have been created from Direct3D 10 resources. The Direct3D 10 objects are released by the OpenCL context associated with *command_queue*.

OpenCL memory objects created from Direct3D 10 resources which have been acquired by OpenCL must be released by OpenCL before they may be accessed by Direct3D 10. Accessing a Direct3D 10 resource while its corresponding OpenCL memory object is acquired is in error and will result in undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

If `CL_CONTEXT_INTEROP_USER_SYNC` is not specified as `CL_TRUE` during context creation, **clEnqueueReleaseD3D10ObjectsKHR** provides the synchronization guarantee that any calls to Direct3D 10 calls involving the interop device(s) used in the OpenCL context made after the call to **clEnqueueReleaseD3D10ObjectsKHR** will not start executing until after all events in *event_wait_list* are complete and all work already submitted to *command_queue* completes execution. If the context was created with properties specifying `CL_CONTEXT_INTEROP_USER_SYNC` as `CL_TRUE`, the user is responsible for guaranteeing that any Direct3D 10 calls involving the interop device(s) used in the OpenCL context made after **clEnqueueReleaseD3D10ObjectsKHR** will not start executing until after event returned by **clEnqueueReleaseD3D10ObjectsKHR** reports completion.

num_objects is the number of memory objects to be released in *mem_objects*.

mem_objects is a pointer to a list of OpenCL memory objects that were created from Direct3D 10 resources.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is `NULL`, then this particular command does not wait on any event to complete. If *event_wait_list* is `NULL`, *num_events_in_wait_list* must be 0. If *event_wait_list* is not `NULL`, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0.

event returns an event object that identifies this command and can be used to query or wait for this command to complete. If *event* is `NULL` or the enqueue is unsuccessful, no event will be created and therefore it will not be possible to query the status of this command or to wait for this command to complete. If *event_wait_list* and *event* are not `NULL`, *event* must not refer to an element of the *event_wait_list* array.

clEnqueueReleaseD3D10ObjectsKHR returns `CL_SUCCESS` if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is `NULL` the function does nothing and returns `CL_SUCCESS`. Otherwise it returns one of the following errors:

- `CL_INVALID_VALUE` if *num_objects* is zero and *mem_objects* is not a `NULL` value or if *num_objects* > 0 and *mem_objects* is `NULL`.
- `CL_INVALID_MEM_OBJECT` if memory objects in *mem_objects* are not valid OpenCL memory objects or if memory objects in *mem_objects* have not been created from Direct3D 10 resources.
- `CL_INVALID_COMMAND_QUEUE` if *command_queue* is not a valid command-queue.
- `CL_INVALID_CONTEXT` if context associated with *command_queue* was not created from a Direct3D 10 device.
- `CL_D3D10_RESOURCE_NOT_ACQUIRED_KHR` if memory objects in *mem_objects* have not previously been acquired using **clEnqueueAcquireD3D10ObjectsKHR**, or have been released using **clEnqueueReleaseD3D10ObjectsKHR** since the last time that they were acquired.

- `CL_INVALID_EVENT_WAIT_LIST` if *event_wait_list* is `NULL` and *num_events_in_wait_list* > 0, or *event_wait_list* is not `NULL` and *num_events_in_wait_list*> is 0, or if event objects in *event_wait_list* are not valid events.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

13.7.7. Event Command Types for Sharing memory objects that map to Direct3D 10 objects

The following table describes the event command types for the OpenCL commands to acquire and release OpenCL memory objects that have been created from Direct3D 10 objects:

Table 33. List of supported event command types

Events Created By	Event Command Type
clEnqueueAcquireD3D10ObjectsKHR	`CL_COMMAND_ACQUIRE_D3D10_OBJECTS_KHR`
clEnqueueReleaseD3D10ObjectsKHR	`CL_COMMAND_RELEASE_D3D10_OBJECTS_KHR`

13.8. Issues

1. Should this extension be KHR or EXT?

   PROPOSED: KHR. If this extension is to be approved by Khronos then it should be KHR, otherwise EXT. Not all platforms can support this extension, but that is also true of OpenGL interop.

   RESOLVED: KHR.

2. Requiring SharedHandle on ID3D10Resource

   Requiring this can largely simplify things at the DDI level and make some implementations faster. However, the DirectX spec only defines the shared handle for a subset of the resources we would like to support:

   ```
   D3D10_RESOURCE_MISC_SHARED - Enables the sharing of resource data between
   two or more Direct3D devices.
   The only resources that can be shared are 2D non-mipmapped textures.
   ```

   PROPOSED A: Add wording to the spec about some implementations needing the resource setup as shared:

   “Some implementations may require the resource to be shared on the D3D10 side of the API”

   If we do that, do we need another enum to describe this failure case?

   PROPOSED B: Require that all implementations support both shared and non-shared resources. The restrictions prohibiting multisample textures and the flag D3D10_USAGE_IMMUTABLE guarantee software access to all shareable resources.

RESOLVED: Require that implementations support both D3D10_RESOURCE_MISC_SHARED being set and not set. Add the query for `CL_CONTEXT_D3D10_PREFER_SHARED_RESOURCES_KHR` to determine on a per-context basis which method will be faster.

3. Texture1D support

   There is not a matching CL type, so do we want to support this and map to buffer or Texture2D?

   RESOLVED: We will not add support for ID3D10Texture1D objects unless a corresponding OpenCL 1D Image type is created.

4. CL/D3D10 queries

   The GL interop has **clGetGLObjectInfo** and **clGetGLTextureInfo**. It is unclear if these are needed on the D3D10 interop side since the D3D10 spec makes these queries trivial on the D3D10 object itself. Also, not all of the semantics of the GL call map across.

   PROPOSED: Add the **clGetMemObjectInfo** and **clGetImageInfo** parameter names `CL_MEM_D3D10_RESOURCE_KHR` and `CL_IMAGE_D3D10_SUBRESOURCE_KHR` to query the D3D10 resource from which a `cl_mem` was created. From this data, any D3D10 side information may be queried using the D3D10 API.

   RESOLVED: We will use **clGetMemObjectInfo** and **clGetImageInfo** to access this information.

Chapter 14. Creating OpenCL Memory Objects from Direct3D 11 Buffers and Textures

14.1. Overview

This section describes the **cl_khr_d3d11_sharing** extension. The goal of this extension is to provide interoperability between OpenCL and Direct3D 11.

14.2. General Information

14.2.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

14.3. New Procedures and Functions

```
cl_int clGetDeviceIDsFromD3D11KHR(cl_platform_id platform,
                                  cl_d3d11_device_source_khr d3d_device_source,
                                  void *d3d_object,
                                  cl_d3d11_device_set_khr d3d_device_set,
                                  cl_uint num_entries,
                                  cl_device_id *devices,
                                  cl_uint *num_devices);

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

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

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

cl_int clEnqueueAcquireD3D11ObjectsKHR(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 clEnqueueReleaseD3D11ObjectsKHR(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);
```

14.4. New Tokens

Accepted as a Direct3D 11 device source in the *d3d_device_source* parameter of **clGetDeviceIDsFromD3D11KHR**:

```
CL_D3D11_DEVICE_KHR
CL_D3D11_DXGI_ADAPTER_KHR
```

Accepted as a set of Direct3D 11 devices in the _d3d_device_set_parameter of **clGetDeviceIDsFromD3D11KHR**:

```
CL_PREFERRED_DEVICES_FOR_D3D11_KHR
CL_ALL_DEVICES_FOR_D3D11_KHR
```

Accepted as a property name in the *properties* parameter of **clCreateContext** and **clCreateContextFromType**:

```
CL_CONTEXT_D3D11_DEVICE_KHR
```

Accepted as a property name in the *param_name* parameter of **clGetContextInfo**:

```
CL_CONTEXT_D3D11_PREFER_SHARED_RESOURCES_KHR
```

Accepted as the property being queried in the *param_name* parameter of **clGetMemObjectInfo**:

```
CL_MEM_D3D11_RESOURCE_KHR
```

Accepted as the property being queried in the *param_name* parameter of **clGetImageInfo**:

```
CL_IMAGE_D3D11_SUBRESOURCE_KHR
```

Returned in the *param_value* parameter of **clGetEventInfo** when *param_name* is `CL_EVENT_COMMAND_TYPE`:

```
CL_COMMAND_ACQUIRE_D3D11_OBJECTS_KHR
CL_COMMAND_RELEASE_D3D11_OBJECTS_KHR
```

Returned by **clCreateContext** and **clCreateContextFromType** if the Direct3D 11 device specified for interoperability is not compatible with the devices against which the context is to be created:

```
CL_INVALID_D3D11_DEVICE_KHR
```

Returned by **clCreateFromD3D11BufferKHR** when *resource* is not a Direct3D 11 buffer object, and by **clCreateFromD3D11Texture2DKHR** and **clCreateFromD3D11Texture3DKHR** when *resource* is not a Direct3D 11 texture object.

```
CL_INVALID_D3D11_RESOURCE_KHR
```

Returned by **clEnqueueAcquireD3D11ObjectsKHR** when any of *mem_objects* are currently acquired by OpenCL:

```
CL_D3D11_RESOURCE_ALREADY_ACQUIRED_KHR
```

Returned by **clEnqueueReleaseD3D11ObjectsKHR** when any of *mem_objects* are not currently acquired by OpenCL:

```
CL_D3D11_RESOURCE_NOT_ACQUIRED_KHR
```

14.5. Additions to Chapter 4 of the OpenCL 2.2 Specification

In *section 4.4*, replace the description of *properties* under **clCreateContext** with:

"*properties* specifies a list of context property names and their corresponding values. Each property is followed immediately by the corresponding desired value. The list is terminated with zero. If a property is not specified in *properties*, then its default value (listed in *table 4.5*) is used (it is said to be specified implicitly). If *properties* is `NULL` or empty (points to a list whose first value is zero), all attributes take on their default values."

Add the following to *table 4.5*:

cl_context_properties enum	Property value	Description
CL_CONTEXT_D3D11_DEVICE_KHR	ID3D11Device *	Specifies the ID3D11Device * to use for Direct3D 11 interoperability. The default value is NULL.

Add to the list of errors for **clCreateContext**:

- CL_INVALID_D3D11_DEVICE_KHR if the value of the property CL_CONTEXT_D3D11_DEVICE_KHR is non-NULL and does not specify a valid Direct3D 11 device with which the *cl_device_ids* against which this context is to be created may interoperate.
- CL_INVALID_OPERATION if Direct3D 11 interoperability is specified by setting CL_INVALID_D3D11_DEVICE_KHR to a non-NULL value, and interoperability with another graphics API is also specified.

Add to the list of errors for **clCreateContextFromType** the same new errors described above for **clCreateContext**.

Add the following row to *table 4.6*:

cl_context_info	Return Type	Information returned in param_value
CL_CONTEXT_D3D11_PREFER_SHARED_RESOURCES_KHR	cl_bool	Returns CL_TRUE if Direct3D 11 resources created as shared by setting *MiscFlags* to include D3D11_RESOURCE_MISC_SHARED will perform faster when shared with OpenCL, compared with resources which have not set this flag. Otherwise returns CL_FALSE.

14.6. Additions to Chapter 5 of the OpenCL 2.2 Specification

Add to the list of errors for **clGetMemObjectInfo**:

- CL_INVALID_D3D11_RESOURCE_KHR if *param_name* is CL_MEM_D3D11_RESOURCE_KHR and *memobj* was not created by the function **clCreateFromD3D11BufferKHR**, **clCreateFromD3D11Texture2DKHR**, or **clCreateFromD3D11Texture3DKHR**.

Extend *table 5.12* to include the following entry.

cl_mem_info	Return type	Info. returned in *param_value*
CL_MEM_D3D11_RESOURCE_KHR	ID3D11Resource *	If *memobj* was created using **clCreateFromD3D11BufferKHR**, **clCreateFromD3D11Texture2DKHR**, or **clCreateFromD3D11Texture3DKHR**, returns the *resource* argument specified when *memobj* was created.

Add to the list of errors for **clGetImageInfo**:

- CL_INVALID_D3D11_RESOURCE_KHR if *param_name* is CL_IMAGE_D3D11_SUBRESOURCE_KHR and *image* was not created by the function **clCreateFromD3D11Texture2DKHR**, or **clCreateFromD3D11Texture3DKHR**.

Extend *table 5.9* to include the following entry.

cl_image_info	Return type	Info. returned in *param_value*
CL_IMAGE_D3D11_SUBRESOURCE_KHR	UINT	If *image* was created using **clCreateFromD3D11Texture2DKHR**, or **clCreateFromD3D11Texture3DKHR**, returns the *subresource* argument specified when *image* was created.

Add to *table 5.22* in the **Info returned in param_value** column for *cl_event_info* = CL_EVENT_COMMAND_TYPE:

```
CL_COMMAND_ACQUIRE_D3D11_OBJECTS_KHR
CL_COMMAND_RELEASE_D3D11_OBJECTS_KHR
```

14.7. Sharing Memory Objects with Direct3D 11 Resources

This section discusses OpenCL functions that allow applications to use Direct3D 11 resources as OpenCL memory objects. This allows efficient sharing of data between OpenCL and Direct3D 11. The OpenCL API may be used to execute kernels that read and/or write memory objects that are also Direct3D 11 resources. An OpenCL image object may be created from a Direct3D 11 texture resource. An OpenCL buffer object may be created from a Direct3D 11 buffer resource. OpenCL memory objects may be created from Direct3D 11 objects if and only if the OpenCL context has been created from a Direct3D 11 device.

14.7.1. Querying OpenCL Devices Corresponding to Direct3D 11 Devices

The OpenCL devices corresponding to a Direct3D 11 device may be queried. The OpenCL devices corresponding to a DXGI adapter may also be queried. The OpenCL devices corresponding to a Direct3D 11 device will be a subset of the OpenCL devices corresponding to the DXGI adapter against which the Direct3D 11 device was created.

The OpenCL devices corresponding to a Direct3D 11 device or a DXGI device may be queried using the function

```
cl_int clGetDeviceIDsFromD3D11KHR(
    cl_platform_id platform,
    cl_d3d11_device_source_khr d3d_device_source,
    void* d3d_object,
    cl_d3d11_device_set_khr d3d_device_set,
    cl_uint num_entries,
    cl_device_id* devices,
    cl_uint* num_devices);
```

platform refers to the platform ID returned by **clGetPlatformIDs**.

d3d_device_source specifies the type of *d3d_object*, and must be one of the values shown in the table below.

d3d_object specifies the object whose corresponding OpenCL devices are being queried. The type of *d3d_object* must be as specified in the table below.

d3d_device_set specifies the set of devices to return, and must be one of the values shown in the table below.

num_entries is the number of `cl_device_id` entries that can be added to *devices*. If *devices* is not `NULL` then *num_entries* must be greater than zero.

devices returns a list of OpenCL devices found. The `cl_device_id` values returned in *devices* can be used to identify a specific OpenCL device. If *devices* is `NULL`, this argument is ignored. The number of OpenCL devices returned is the minimum of the value specified by *num_entries* and the number of OpenCL devices corresponding to *d3d_object*.

num_devices returns the number of OpenCL devices available that correspond to *d3d_object*. If *num_devices* is `NULL`, this argument is ignored.

clGetDeviceIDsFromD3D11KHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise it may return

- `CL_INVALID_PLATFORM` if *platform* is not a valid platform.
- `CL_INVALID_VALUE` if *d3d_device_source* is not a valid value, *d3d_device_set* is not a valid value, *num_entries* is equal to zero and *devices* is not `NULL`, or if both *num_devices* and *devices* are `NULL`.
- `CL_DEVICE_NOT_FOUND` if no OpenCL devices that correspond to *d3d_object* were found.

Table 34. Direct3D 11 object types that may be used by **clGetDeviceIDsFromD3D11KHR**

cl_d3d11_device_source_khr	**Type of** ***d3d_object***
CL_D3D11_DEVICE_KHR	ID3D11Device *
CL_D3D11_DXGI_ADAPTER_KHR	IDXGIAdapter *

Table 35. Sets of devices queriable using **clGetDeviceIDsFromD3D11KHR**

cl_d3d11_device_set_khr	**Devices returned in** ***devices***
CL_PREFERRED_DEVICES_FOR_D3D11_KHR	The preferred OpenCL devices associated with the specified Direct3D object.
CL_ALL_DEVICES_FOR_D3D11_KHR	All OpenCL devices which may interoperate with the specified Direct3D object. Performance of sharing data on these devices may be considerably less than on the preferred devices.

14.7.2. Lifetime of Shared Objects

An OpenCL memory object created from a Direct3D 11 resource remains valid as long as the corresponding Direct3D 11 resource has not been deleted. If the Direct3D 11 resource is deleted through the Direct3D 11 API, subsequent use of the OpenCL memory object will result in undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

The successful creation of a cl_context against a Direct3D 11 device specified via the context create parameter CL_CONTEXT_D3D11_DEVICE_KHR will increment the internal Direct3D reference count on the specified Direct3D 11 device. The internal Direct3D reference count on that Direct3D 11 device will be decremented when the OpenCL reference count on the returned OpenCL context drops to zero.

The OpenCL context and corresponding command-queues are dependent on the existence of the Direct3D 11 device from which the OpenCL context was created. If the Direct3D 11 device is deleted through the Direct3D 11 API, subsequent use of the OpenCL context will result in undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

14.7.3. Sharing Direct3D 11 Buffer Resources as OpenCL Buffer Objects

The function

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

creates an OpenCL buffer object from a Direct3D 11 buffer.

context is a valid OpenCL context created from a Direct3D 11 device.

flags is a bit-field that is used to specify usage information. Refer to table 5.3 for a description of *flags*. Only `CL_MEM_READ_ONLY`, `CL_MEM_WRITE_ONLY` and `CL_MEM_READ_WRITE` values specified in *table 5.3* can be used.

resource is a pointer to the Direct3D 11 buffer to share.

errcode_ret will return an appropriate error code. If *errcode_ret* is `NULL`, no error code is returned.

clCreateFromD3D11BufferKHR returns a valid non-zero OpenCL buffer object and *errcode_ret* is set to `CL_SUCCESS` if the buffer object is created successfully. Otherwise, it returns a `NULL` value with one of the following error values returned in *errcode_ret*:

- `CL_INVALID_CONTEXT` if *context* is not a valid context.
- `CL_INVALID_VALUE` if values specified in *flags* are not valid.
- `CL_INVALID_D3D11_RESOURCE_KHR` if *resource* is not a Direct3D 11 buffer resource, if *resource* was created with the D3D11_USAGE flag D3D11_USAGE_IMMUTABLE, if a `cl_mem` from *resource* has already been created using **clCreateFromD3D11BufferKHR**, or if *context* was not created against the same Direct3D 11 device from which *resource* was created.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

The size of the returned OpenCL buffer object is the same as the size of *resource*. This call will increment the internal Direct3D reference count on *resource*. The internal Direct3D reference count on *resource* will be decremented when the OpenCL reference count on the returned OpenCL memory object drops to zero.

14.7.4. Sharing Direct3D 11 Texture and Resources as OpenCL Image Objects

The function

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

creates an OpenCL 2D image object from a subresource of a Direct3D 11 2D texture.

context is a valid OpenCL context created from a Direct3D 11 device.

flags is a bit-field that is used to specify usage information. Refer to *table 5.3* for a description of *flags*. Only `CL_MEM_READ_ONLY`, `CL_MEM_WRITE_ONLY` and `CL_MEM_READ_WRITE` values specified in *table 5.3* can be used.

resource is a pointer to the Direct3D 11 2D texture to share.

subresource is the subresource of *resource* to share.

errcode_ret will return an appropriate error code. If *errcode_ret* is `NULL`, no error code is returned.

clCreateFromD3D11Texture2DKHR returns a valid non-zero OpenCL image object and *errcode_ret* is set to `CL_SUCCESS` if the image object is created successfully. Otherwise, it returns a `NULL` value with one of the following error values returned in *errcode_ret*:

- `CL_INVALID_CONTEXT` if *context* is not a valid context.
- `CL_INVALID_VALUE` if values specified in *flags* are not valid or if *subresource* is not a valid subresource index for *resource*.
- `CL_INVALID_D3D11_RESOURCE_KHR` if *resource* is not a Direct3D 11 texture resource, if *resource* was created with the D3D11_USAGE flag D3D11_USAGE_IMMUTABLE, if *resource* is a multisampled texture, if a `cl_mem` from subresource *subresource* of *resource* has already been created using **clCreateFromD3D11Texture2DKHR**, or if *context* was not created against the same Direct3D 11 device from which *resource* was created.
- `CL_INVALID_IMAGE_FORMAT_DESCRIPTOR` if the Direct3D 11 texture format of *resource* is not listed in the table *Direct3D 11 formats and corresponding OpenCL image formats* or if the Direct3D 11 texture format of *resource* does not map to a supported OpenCL image format.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

The width and height of the returned OpenCL 2D image object are determined by the width and height of subresource *subresource* of *resource*. The channel type and order of the returned OpenCL 2D image object is determined by the format of *resource* by the table *Direct3D 11 formats and corresponding OpenCL image formats*.

This call will increment the internal Direct3D reference count on *resource*. The internal Direct3D reference count on *resource* will be decremented when the OpenCL reference count on the returned OpenCL memory object drops to zero.

The function

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

creates an OpenCL 3D image object from a subresource of a Direct3D 11 3D texture.

context is a valid OpenCL context created from a Direct3D 11 device.

flags is a bit-field that is used to specify usage information. Refer to *table 5.3* for a description of *flags*. Only `CL_MEM_READ_ONLY`, `CL_MEM_WRITE_ONLY` and `CL_MEM_READ_WRITE` values specified in *table 5.3* can be used.

resource is a pointer to the Direct3D 11 3D texture to share.

subresource is the subresource of *resource* to share.

errcode_ret will return an appropriate error code. If *errcode_ret* is `NULL`, no error code is returned.

clCreateFromD3D11Texture3DKHR returns a valid non-zero OpenCL image object and *errcode_ret* is set to `CL_SUCCESS` if the image object is created successfully. Otherwise, it returns a `NULL` value with one of the following error values returned in *errcode_ret*:

- `CL_INVALID_CONTEXT` if *context* is not a valid context.
- `CL_INVALID_VALUE` if values specified in *flags* are not valid or if *subresource* is not a valid subresource index for *resource*.
- `CL_INVALID_D3D11_RESOURCE_KHR` if *resource* is not a Direct3D 11 texture resource, if *resource* was created with the D3D11_USAGE flag D3D11_USAGE_IMMUTABLE, if *resource* is a multisampled texture, if a `cl_mem` from subresource *subresource* of *resource* has already been created using **clCreateFromD3D11Texture3DKHR**, or if *context* was not created against the same Direct3D 11 device from which *resource* was created.
- `CL_INVALID_IMAGE_FORMAT_DESCRIPTOR` if the Direct3D 11 texture format of *resource* is not listed in the table *Direct3D 11 formats and corresponding OpenCL image formats* or if the Direct3D 11 texture format of *resource* does not map to a supported OpenCL image format.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

The width, height and depth of the returned OpenCL 3D image object are determined by the width, height and depth of subresource *subresource* of *resource*. The channel type and order of the returned OpenCL 3D image object is determined by the format of *resource* by the table *Direct3D 11 formats and corresponding OpenCL image formats*.

This call will increment the internal Direct3D reference count on *resource*. The internal Direct3D reference count on *resource* will be decremented when the OpenCL reference count on the returned OpenCL memory object drops to zero.

Table 36. Direct3D 11 formats and corresponding OpenCL image formats

DXGI format	CL image format (channel order, channel data type)
DXGI_FORMAT_R32G32B32A32_FLOAT	`CL_RGBA`, `CL_FLOAT`
DXGI_FORMAT_R32G32B32A32_UINT	`CL_RGBA`, `CL_UNSIGNED_INT32`
DXGI_FORMAT_R32G32B32A32_SINT	`CL_RGBA`, `CL_SIGNED_INT32`
DXGI_FORMAT_R16G16B16A16_FLOAT	`CL_RGBA`, `CL_HALF_FLOAT`
DXGI_FORMAT_R16G16B16A16_UNORM	`CL_RGBA`, `CL_UNORM_INT16`
DXGI_FORMAT_R16G16B16A16_UINT	`CL_RGBA`, `CL_UNSIGNED_INT16`
DXGI_FORMAT_R16G16B16A16_SNORM	`CL_RGBA`, `CL_SNORM_INT16`
DXGI_FORMAT_R16G16B16A16_SINT	`CL_RGBA`, `CL_SIGNED_INT16`

DXGI format	CL image format (channel order, channel data type)
DXGI_FORMAT_B8G8R8A8_UNORM	CL_BGRA, CL_UNORM_INT8
DXGI_FORMAT_R8G8B8A8_UNORM	CL_RGBA, CL_UNORM_INT8
DXGI_FORMAT_R8G8B8A8_UINT	CL_RGBA, CL_UNSIGNED_INT8
DXGI_FORMAT_R8G8B8A8_SNORM	CL_RGBA, CL_SNORM_INT8
DXGI_FORMAT_R8G8B8A8_SINT	CL_RGBA, CL_SIGNED_INT8
DXGI_FORMAT_R32G32_FLOAT	CL_RG, CL_FLOAT
DXGI_FORMAT_R32G32_UINT	CL_RG, CL_UNSIGNED_INT32
DXGI_FORMAT_R32G32_SINT	CL_RG, CL_SIGNED_INT32
DXGI_FORMAT_R16G16_FLOAT	CL_RG, CL_HALF_FLOAT
DXGI_FORMAT_R16G16_UNORM	CL_RG, CL_UNORM_INT16
DXGI_FORMAT_R16G16_UINT	CL_RG, CL_UNSIGNED_INT16
DXGI_FORMAT_R16G16_SNORM	CL_RG, CL_SNORM_INT16
DXGI_FORMAT_R16G16_SINT	CL_RG, CL_SIGNED_INT16
DXGI_FORMAT_R8G8_UNORM	CL_RG, CL_UNORM_INT8
DXGI_FORMAT_R8G8_UINT	CL_RG, CL_UNSIGNED_INT8
DXGI_FORMAT_R8G8_SNORM	CL_RG, CL_SNORM_INT8
DXGI_FORMAT_R8G8_SINT	CL_RG, CL_SIGNED_INT8
DXGI_FORMAT_R32_FLOAT	CL_R, CL_FLOAT
DXGI_FORMAT_R32_UINT	CL_R, CL_UNSIGNED_INT32
DXGI_FORMAT_R32_SINT	CL_R, CL_SIGNED_INT32
DXGI_FORMAT_R16_FLOAT	CL_R, CL_HALF_FLOAT
DXGI_FORMAT_R16_UNORM	CL_R, CL_UNORM_INT16
DXGI_FORMAT_R16_UINT	CL_R, CL_UNSIGNED_INT16
DXGI_FORMAT_R16_SNORM	CL_R, CL_SNORM_INT16
DXGI_FORMAT_R16_SINT	CL_R, CL_SIGNED_INT16
DXGI_FORMAT_R8_UNORM	CL_R, CL_UNORM_INT8
DXGI_FORMAT_R8_UINT	CL_R, CL_UNSIGNED_INT8
DXGI_FORMAT_R8_SNORM	CL_R, CL_SNORM_INT8
DXGI_FORMAT_R8_SINT	CL_R, CL_SIGNED_INT8

14.7.5. Querying Direct3D properties of memory objects created from Direct3D 11 resources

Properties of Direct3D 11 objects may be queried using **clGetMemObjectInfo** and **clGetImageInfo** with *param_name* CL_MEM_D3D11_RESOURCE_KHR and CL_IMAGE_D3D11_SUBRESOURCE_KHR respectively as described in *sections 5.4.3* and *5.3.6*.

14.7.6. Sharing memory objects created from Direct3D 11 resources between Direct3D 11 and OpenCL contexts

The function

```
cl_int clEnqueueAcquireD3D11ObjectsKHR(
    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);
```

is used to acquire OpenCL memory objects that have been created from Direct3D 11 resources. The Direct3D 11 objects are acquired by the OpenCL context associated with *command_queue* and can therefore be used by all command-queues associated with the OpenCL context.

OpenCL memory objects created from Direct3D 11 resources must be acquired before they can be used by any OpenCL commands queued to a command-queue. If an OpenCL memory object created from a Direct3D 11 resource is used while it is not currently acquired by OpenCL, the behavior is undefined. Implementations may fail the execution of commands attempting to use that OpenCL memory object and set their associated event's execution status to CL_D3D11_RESOURCE_NOT_ACQUIRED_KHR.

If CL_CONTEXT_INTEROP_USER_SYNC is not specified as CL_TRUE during context creation, **clEnqueueAcquireD3D11ObjectsKHR** provides the synchronization guarantee that any Direct3D 11 calls involving the interop device(s) used in the OpenCL context made before **clEnqueueAcquireD3D11ObjectsKHR** is called will complete executing before *event* reports completion and before the execution of any subsequent OpenCL work issued in *command_queue* begins. If the context was created with properties specifying CL_CONTEXT_INTEROP_USER_SYNC as CL_TRUE, the user is responsible for guaranteeing that any Direct3D 11 calls involving the interop device(s) used in the OpenCL context made before **clEnqueueAcquireD3D11ObjectsKHR** is called have completed before calling **clEnqueueAcquireD3D11ObjectsKHR**.

command_queue is a valid command-queue.

num_objects is the number of memory objects to be acquired in *mem_objects*.

mem_objects is a pointer to a list of OpenCL memory objects that were created from Direct3D 11 resources.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is `NULL`, then this particular command does not wait on any event to complete. If *event_wait_list* is `NULL`, *num_events_in_wait_list* must be 0. If *event_wait_list* is not `NULL`, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points.

event returns an event object that identifies this command and can be used to query or wait for this command to complete. If *event* is `NULL` or the enqueue is unsuccessful, no event will be created and therefore it will not be possible to query the status of this command or to wait for this command to complete. If *event_wait_list* and *event* are not `NULL`, *event* must not refer to an element of the *event_wait_list* array.

clEnqueueAcquireD3D11ObjectsKHR returns `CL_SUCCESS` if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is `NULL` then the function does nothing and returns `CL_SUCCESS`. Otherwise it returns one of the following errors:

- `CL_INVALID_VALUE` if *num_objects* is zero and *mem_objects* is not a `NULL` value or if *num_objects* > 0 and *mem_objects* is `NULL`.
- `CL_INVALID_MEM_OBJECT` if memory objects in *mem_objects* are not valid OpenCL memory objects or if memory objects in *mem_objects* have not been created from Direct3D 11 resources.
- `CL_INVALID_COMMAND_QUEUE` if *command_queue* is not a valid command-queue.
- `CL_INVALID_CONTEXT` if context associated with *command_queue* was not created from an Direct3D 11 context.
- `CL_D3D11_RESOURCE_ALREADY_ACQUIRED_KHR` if memory objects in *mem_objects* have previously been acquired using **clEnqueueAcquireD3D11ObjectsKHR** but have not been released using **clEnqueueReleaseD3D11ObjectsKHR**.
- `CL_INVALID_EVENT_WAIT_LIST` if *event_wait_list* is `NULL` and *num_events_in_wait_list* > 0, or *event_wait_list* is not `NULL` and *num_events_in_wait_list* is 0, or if event objects in *event_wait_list* are not valid events.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

```
cl_int clEnqueueReleaseD3D11ObjectsKHR(
    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);
```

is used to release OpenCL memory objects that have been created from Direct3D 11 resources. The Direct3D 11 objects are released by the OpenCL context associated with *command_queue*.

OpenCL memory objects created from Direct3D 11 resources which have been acquired by OpenCL must be released by OpenCL before they may be accessed by Direct3D 11. Accessing a Direct3D 11 resource while its corresponding OpenCL memory object is acquired is in error and will result in undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

If `CL_CONTEXT_INTEROP_USER_SYNC` is not specified as `CL_TRUE` during context creation, **clEnqueueReleaseD3D11ObjectsKHR** provides the synchronization guarantee that any calls to Direct3D 11 calls involving the interop device(s) used in the OpenCL context made after the call to **clEnqueueReleaseD3D11ObjectsKHR** will not start executing until after all events in *event_wait_list* are complete and all work already submitted to *command_queue* completes execution. If the context was created with properties specifying `CL_CONTEXT_INTEROP_USER_SYNC` as `CL_TRUE`, the user is responsible for guaranteeing that any Direct3D 11 calls involving the interop device(s) used in the OpenCL context made after **clEnqueueReleaseD3D11ObjectsKHR** will not start executing until after event returned by **clEnqueueReleaseD3D11ObjectsKHR** reports completion.

num_objects is the number of memory objects to be released in *mem_objects*.

mem_objects is a pointer to a list of OpenCL memory objects that were created from Direct3D 11 resources.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is `NULL`, then this particular command does not wait on any event to complete. If *event_wait_list* is `NULL`, *num_events_in_wait_list* must be 0. If *event_wait_list* is not `NULL`, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0.

event returns an event object that identifies this command and can be used to query or wait for this command to complete. If *event* is `NULL` or the enqueue is unsuccessful, no event will be created and therefore it will not be possible to query the status of this command or to wait for this command to complete. If *event_wait_list* and *event* are not `NULL`, *event* must not refer to an element of the *event_wait_list* array.

clEnqueueReleaseD3D11ObjectsKHR returns `CL_SUCCESS` if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is `NULL` the function does nothing and returns `CL_SUCCESS`. Otherwise it returns one of the following errors:

- `CL_INVALID_VALUE` if *num_objects* is zero and *mem_objects* is not a `NULL` value or if *num_objects* > 0 and *mem_objects* is `NULL`.
- `CL_INVALID_MEM_OBJECT` if memory objects in *mem_objects* are not valid OpenCL memory objects or if memory objects in *mem_objects* have not been created from Direct3D 11 resources.
- `CL_INVALID_COMMAND_QUEUE` if *command_queue* is not a valid command-queue.
- `CL_INVALID_CONTEXT` if context associated with *command_queue* was not created from a Direct3D 11 device.
- `CL_D3D11_RESOURCE_NOT_ACQUIRED_KHR` if memory objects in *mem_objects* have not previously been acquired using **clEnqueueAcquireD3D11ObjectsKHR**, or have been released using **clEnqueueReleaseD3D11ObjectsKHR** since the last time that they were acquired.

- CL_INVALID_EVENT_WAIT_LIST if *event_wait_list* is NULL and *num_events_in_wait_list* > 0, or *event_wait_list* is not NULL and *num_events_in_wait_list*> is 0, or if event objects in *event_wait_list* are not valid events.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

14.7.7. Event Command Types for Sharing memory objects that map to Direct3D 11 objects

The following table describes the event command types for the OpenCL commands to acquire and release OpenCL memory objects that have been created from Direct3D 11 objects:

Table 37. List of supported event command types

Events Created By	Event Command Type
clEnqueueAcquireD3D11ObjectsKHR	CL_COMMAND_ACQUIRE_D3D11_OBJECTS_KHR
clEnqueueReleaseD3D11ObjectsKHR	CL_COMMAND_RELEASE_D3D11_OBJECTS_KHR

Chapter 15. Creating OpenCL Memory Objects from DirectX 9 Media Surfaces

15.1. Overview

This section describes the **cl_khr_dx9_media_sharing** extension. The goal of this extension is to allow applications to use media surfaces as OpenCL memory objects. This allows efficient sharing of data between OpenCL and selected adapter APIs (only DX9 for now). If this extension is supported, an OpenCL image object can be created from a media surface and the OpenCL API can be used to execute kernels that read and/or write memory objects that are media surfaces. Note that OpenCL memory objects may be created from the adapter media surface if and only if the OpenCL context has been created from that adapter.

15.2. General Information

15.2.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

15.3. New Procedures and Functions

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

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

cl_int clEnqueueAcquireDX9MediaSurfacesKHR(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 clEnqueueReleaseDX9MediaSurfacesKHR(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);
```

15.4. New Tokens

Accepted by the *media_adapter_type* parameter of **clGetDeviceIDsFromDX9MediaAdapterKHR**:

```
CL_ADAPTER_D3D9_KHR
CL_ADAPTER_D3D9EX_KHR
CL_ADAPTER_DXVA_KHR
```

Accepted by the *media_adapter_set* parameter of **clGetDeviceIDsFromDX9MediaAdapterKHR**:

```
CL_PREFERRED_DEVICES_FOR_DX9_MEDIA_ADAPTER_KHR
CL_ALL_DEVICES_FOR_DX9_MEDIA_ADAPTER_KHR
```

Accepted as a property name in the *properties* parameter of **clCreateContext** and **clCreateContextFromType**:

```
CL_CONTEXT_ADAPTER_D3D9_KHR
CL_CONTEXT_ADAPTER_D3D9EX_KHR
CL_CONTEXT_ADAPTER_DXVA_KHR
```

Accepted as the property being queried in the *param_name* parameter of **clGetMemObjectInfo**:

```
CL_MEM_DX9_MEDIA_ADAPTER_TYPE_KHR
CL_MEM_DX9_MEDIA_SURFACE_INFO_KHR
```

Accepted as the property being queried in the *param_name* parameter of **clGetImageInfo**:

```
CL_IMAGE_DX9_MEDIA_PLANE_KHR
```

Returned in the *param_value* parameter of **clGetEventInfo** when *param_name* is `CL_EVENT_COMMAND_TYPE`:

```
CL_COMMAND_ACQUIRE_DX9_MEDIA_SURFACES_KHR
```

```
CL_COMMAND_RELEASE_DX9_MEDIA_SURFACES_KHR
```

Returned by **clCreateContext** and **clCreateContextFromType** if the media adapter specified for interoperability is not compatible with the devices against which the context is to be created:

```
CL_INVALID_DX9_MEDIA_ADAPTER_KHR
```

Returned by **clCreateFromDX9MediaSurfaceKHR** when *adapter_type* is set to a media adapter and the *surface_info* does not reference a media surface of the required type, or if *adapter_type* is set to a media adapter type and *surface_info* does not contain a valid reference to a media surface on that adapter, by **clGetMemObjectInfo** when *param_name* is a surface or handle when the image was not created from an appropriate media surface, and from **clGetImageInfo** when *param_name* is `CL_IMAGE_DX9_MEDIA_PLANE_KHR` and image was not created from an appropriate media surface.

```
CL_INVALID_DX9_MEDIA_SURFACE_KHR
```

Returned by **clEnqueueAcquireDX9MediaSurfacesKHR** when any of *mem_objects* are currently acquired by OpenCL:

```
CL_DX9_MEDIA_SURFACE_ALREADY_ACQUIRED_KHR
```

Returned by **clEnqueueReleaseDX9MediaSurfacesKHR** when any of *mem_objects* are not currently acquired by OpenCL:

```
CL_DX9_MEDIA_SURFACE_NOT_ACQUIRED_KHR
```

15.5. Additions to Chapter 4 of the OpenCL 2.2 Specification

In *section 4.4*, replace the description of *properties* under **clCreateContext** with:

“_properties_ specifies a list of context property names and their corresponding values. Each property is followed immediately by the corresponding desired value. The list is terminated with zero. If a property is not specified in *properties*, then its default value (listed in *table 4.5*) is used (it is said to be specified implicitly). If *properties* is `NULL` or empty (points to a list whose first value is zero), all attributes take on their default values.”

Add the following to *table 4.5*:

cl_context_properties enum	**Property value**	**Description**
`CL_CONTEXT_ADAPTER_D3D9_KHR`	IDirect3DDevice9 *	Specifies an IDirect3DDevice9 to use for D3D9 interop.

cl_context_properties enum	Property value	Description
CL_CONTEXT_ADAPTER_D3D9EX_KHR	IDirect3DDeviceEx*	Specifies an IDirect3DDevice9Ex to use for D3D9 interop.
CL_CONTEXT_ADAPTER_DXVA_KHR	IDXVAHD_Device *	Specifies an IDXVAHD_Device to use for DXVA interop.

Add to the list of errors for **clCreateContext**:

- CL_INVALID_DX9_MEDIA_ADAPTER_KHR if any of the values of the properties CL_CONTEXT_ADAPTER_D3D9_KHR, CL_CONTEXT_ADAPTER_D3D9EX_KHR or CL_CONTEXT_ADAPTER_DXVA_KHR is non-NULL and does not specify a valid media adapter with which the *cl_device_ids* against which this context is to be created may interoperate.

Add to the list of errors for **clCreateContextFromType** the same new errors described above for **clCreateContext**.

15.6. Additions to Chapter 5 of the OpenCL 2.2 Specification

Add to the list of errors for **clGetMemObjectInfo**:

- CL_INVALID_DX9_MEDIA_SURFACE_KHR if *param_name* is CL_MEM_DX9_MEDIA_SURFACE_INFO_KHR and *memobj* was not created by the function **clCreateFromDX9MediaSurfaceKHR** from a Direct3D9 surface.

Extend *table 5.12* to include the following entry:

cl_mem_info	Return type	Info. returned in *param_value*
CL_MEM_DX9_MEDIA_ADAPTER_TYPE_KHR	cl_dx9_media_adapter_type_khr	Returns the cl_dx9_media_adapter_type_khr argument value specified when *memobj* is created using **clCreateFromDX9MediaSurfaceKHR**.
CL_MEM_DX9_MEDIA_SURFACE_INFO_KHR	cl_dx9_surface_info_khr	Returns the cl_dx9_surface_info_khr argument value specified when *memobj* is created using **clCreateFromDX9MediaSurfaceKHR**.

Add to the list of errors for **clGetImageInfo**:

- CL_INVALID_DX9_MEDIA_SURFACE_KHR if *param_name* is CL_IMAGE_DX9_MEDIA_PLANE_KHR and *image* was not created by the function **clCreateFromDX9MediaSurfaceKHR**.

Extend *table 5.9* to include the following entry.

cl_image_info	**Return type**	**Info. returned in *param_value***
`CL_IMAGE_DX9_MEDIA_PLANE_KHR`	`cl_uint`	Returns the *plane* argument value specified when *memobj* is created using **clCreateFromDX9MediaSurfaceKHR**.

Add to *table 5.22* in the **Info returned in param_value** column for *cl_event_info* = `CL_EVENT_COMMAND_TYPE`:

```
CL_COMMAND_ACQUIRE_DX9_MEDIA_SURFACES_KHR
CL_COMMAND_RELEASE_DX9_MEDIA_SURFACES_KHR
```

15.7. Sharing Media Surfaces with OpenCL

This section discusses OpenCL functions that allow applications to use media surfaces as OpenCL memory objects. This allows efficient sharing of data between OpenCL and media surface APIs. The OpenCL API may be used to execute kernels that read and/or write memory objects that are also media surfaces. An OpenCL image object may be created from a media surface. OpenCL memory objects may be created from media surfaces if and only if the OpenCL context has been created from a media adapter.

15.7.1. Querying OpenCL Devices corresponding to Media Adapters

Media adapters are an abstraction associated with devices that provide media capabilities.

The function

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

queries a media adapter for any associated OpenCL devices. Adapters with associated OpenCL devices can enable media surface sharing between the two.

platform refers to the platform ID returned by **clGetPlatformIDs**.

num_media_adapters specifies the number of media adapters.

media_adapters_type is an array of *num_media_adapters* entries. Each entry specifies the type of media adapter and must be one of the values described in the table below.

Table 38. DirectX 9 object types that may be used by **clGetDeviceIDsFromDX9MediaAdapterKHR**

`cl_dx9_media_adapter_type_khr`	**Type of media adapter**
`CL_ADAPTER_D3D9_KHR`	IDirect3DDevice9 *
`CL_ADAPTER_D3D9EX_KHR`	IDirect3DDevice9Ex *
`CL_ADAPTER_DXVA_KHR`	IDXVAHD_Device *

Table 39. Sets of devices queriable using **clGetDeviceIDsFromDX9MediaAdapterKHR**

`cl_dx9_media_adapter_set_khr`	**Description**
`CL_PREFERRED_DEVICES_FOR_DX9_MEDIA_ADAPTER_KHR`	The preferred OpenCL devices associated with the media adapter.
`CL_ALL_DEVICES_FOR_DX9_MEDIA_ADAPTER_KHR`	All OpenCL devices that may interoperate with the media adapter

media_adapters is an array of *num_media_adapters* entries. Each entry specifies the actual adapter whose type is specified by *media_adapter_type*. The *media_adapters* must be one of the types described in the table *cl_dx9_media_adapter_type_khr values*. *media_adapter_set* specifies the set of adapters to return and must be one of the values described in the table <<[[cl_khr_dx9_media_sharing-media-adapter-sets,*cl_dx9_media_adapter_set_khr values*>>.

num_entries is the number of `cl_device_id` entries that can be added to *devices*. If *devices* is not `NULL`, the *num_entries* must be greater than zero.

devices returns a list of OpenCL devices found that support the list of media adapters specified. The `cl_device_id` values returned in *devices* can be used to identify a specific OpenCL device. If *devices* argument is `NULL`, this argument is ignored. The number of OpenCL devices returned is the minimum of the value specified by *num_entries* or the number of OpenCL devices whose type matches *device_type*.

num_devices returns the number of OpenCL devices. If *num_devices* is `NULL`, this argument is ignored.

clGetDeviceIDsFromDX9MediaAdapterKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns one of the following errors:

- `CL_INVALID_PLATFORM` if *platform* is not a valid platform.
- `CL_INVALID_VALUE` if *num_media_adapters* is zero or if *media_adapters_type* is `NULL` or if *media_adapters* is `NULL`.
- `CL_INVALID_VALUE` if any of the entries in *media_adapters_type* or *media_adapters* is not a valid value.

- CL_INVALID_VALUE if *media_adapter_set* is not a valid value.
- CL_INVALID_VALUE if *num_entries* is equal to zero and *devices* is not NULL or if both *num_devices* and *devices* are NULL.
- CL_DEVICE_NOT_FOUND if no OpenCL devices that correspond to adapters specified in *media_adapters* and *media_adapters_type* were found.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

15.7.2. Creating Media Resources as OpenCL Image Objects

The function

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

creates an OpenCL image object from a media surface.

context is a valid OpenCL context created from a media adapter.

flags is a bit-field that is used to specify usage information. Refer to *table 5.3* for a description of flags. Only CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE values specified in *table 5.3* can be used.

adapter_type is a value from enumeration of supported adapters described in the table *cl_dx9_media_adapter_type_khr values*. The type of *surface_info* is determined by the adapter type. The implementation does not need to support all adapter types. This approach provides flexibility to support additional adapter types in the future. Supported adapter types are CL_ADAPTER_D3D9_KHR, CL_ADAPTER_D3D9EX_KHR and CL_ADAPTER_DXVA_KHR.

If *adapter_type* is CL_ADAPTER_D3D9_KHR, CL_ADAPTER_D3D9EX_KHR and CL_ADAPTER_DXVA_KHR, the *surface_info* points to the following structure:

```
typedef struct cl_dx9_surface_info_khr {
    IDirect3DSurface9*    resource;
    HANDLE                shared_handle;
} cl_dx9_surface_info_khr;
```

For DX9 surfaces, we need both the handle to the resource and the resource itself to have a sufficient amount of information to eliminate a copy of the surface for sharing in cases where this is possible. Elimination of the copy is driver dependent. *shared_handle* may be `NULL` and this may result in sub-optimal performance.

surface_info is a pointer to one of the structures defined in the *adapter_type* description above passed in as a void *.

plane is the plane of resource to share for planar surface formats. For planar formats, we use the plane parameter to obtain a handle to thie specific plane (Y, U or V for example). For non-planar formats used by media, *plane* must be 0.

errcode_ret will return an appropriate error code. If *errcode_ret* is `NULL`, no error code is returned.

clCreateFromDX9MediaSurfaceKHR returns a valid non-zero 2D image object and *errcode_ret* is set to `CL_SUCCESS` if the 2D image object is created successfully. Otherwise it returns a `NULL` value with one of the following error values returned in *errcode_ret*:

- `CL_INVALID_CONTEXT` if *context* is not a valid context.
- `CL_INVALID_VALUE` if values specified in *flags* are not valid or if *plane* is not a valid plane of *resource* specified in *surface_info*.
- `CL_INVALID_DX9_MEDIA_SURFACE_KHR` if *resource* specified in *surface_info* is not a valid resource or is not associated with *adapter_type* (e.g., *adapter_type* is set to `CL_ADAPTER_D3D9_KHR` and *resource* is not a Direct3D 9 surface created in D3DPOOL_DEFAULT).
- `CL_INVALID_DX9_MEDIA_SURFACE_KHR` if *shared_handle* specified in *surface_info* is not `NULL` or a valid handle value.
- `CL_INVALID_IMAGE_FORMAT_DESCRIPTOR` if the texture format of *resource* is not listed in *YUV FourCC codes and corresponding OpenCL image format* or *Direct3D formats and corresponding OpenCL image formats*.
- `CL_INVALID_OPERATION` if there are no devices in *context* that support *adapter_type*.
- `CL_OUT_OF_RESOURCES` if there is a failure to allocate resources required by the OpenCL implementation on the device.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

The width and height of the returned OpenCL 2D image object are determined by the width and height of the plane of resource. The channel type and order of the returned image object is determined by the format and plane of resource and are described in the table *YUV FourCC codes and corresponding OpenCL image format* or *Direct3D formats and corresponding OpenCL image formats*.

This call will increment the internal media surface count on *resource*. The internal media surface reference count on *resource* will be decremented when the OpenCL reference count on the returned OpenCL memory object drops to zero.

15.7.3. Querying Media Surface Properties of Memory Objects created from Media Surfaces

Properties of media surface objects may be queried using **clGetMemObjectInfo** and **clGetImageInfo** with *param_name* CL_MEM_DX9_MEDIA_ADAPTER_TYPE_KHR, CL_MEM_DX9_MEDIA_SURFACE_INFO_KHR and CL_IMAGE_DX9_MEDIA_PLANE_KHR as described in *sections 5.4.3* and *5.3.6*.

15.7.4. Sharing Memory Objects created from Media Surfaces between a Media Adapter and OpenCL

The function

```
cl_int clEnqueueAcquireDX9MediaSurfacesKHR(
    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);
```

is used to acquire OpenCL memory objects that have been created from a media surface. The media surfaces are acquired by the OpenCL context associated with *command_queue* and can therefore be used by all command-queues associated with the OpenCL context.

OpenCL memory objects created from media surfaces must be acquired before they can be used by any OpenCL commands queued to a command-queue. If an OpenCL memory object created from a media surface is used while it is not currently acquired by OpenCL, the call attempting to use that OpenCL memory object will return CL_DX9_MEDIA_SURFACE_NOT_ACQUIRED_KHR.

If CL_CONTEXT_INTEROP_USER_SYNC is not specified as CL_TRUE during context creation, **clEnqueueAcquireDX9MediaSurfacesKHR** provides the synchronization guarantee that any media adapter API calls involving the interop device(s) used in the OpenCL context made before **clEnqueueAcquireDX9MediaSurfacesKHR** is called will complete executing before *event* reports completion and before the execution of any subsequent OpenCL work issued in *command_queue* begins. If the context was created with properties specifying CL_CONTEXT_INTEROP_USER_SYNC as CL_TRUE, the user is responsible for guaranteeing that any media adapter API calls involving the interop device(s) used in the OpenCL context made before **clEnqueueAcquireDX9MediaSurfacesKHR** is called have completed before calling **clEnqueueAcquireDX9MediaSurfacesKHR** .

command_queue is a valid command-queue.

num_objects is the number of memory objects to be acquired in *mem_objects*.

mem_objects is a pointer to a list of OpenCL memory objects that were created from media surfaces.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is NULL, then this particular command does not wait on any event to complete. If *event_wait_list* is NULL, *num_events_in_wait_list* must be 0. If *event_wait_list* is not `NULL`, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points.

event returns an event object that identifies this command and can be used to query or wait for this command to complete. If *event* is `NULL` or the enqueue is unsuccessful, no event will be created and therefore it will not be possible to query the status of this command or to wait for this command to complete. If *event_wait_list* and *event* are not `NULL`, *event* must not refer to an element of the *event_wait_list* array.

clEnqueueAcquireDX9MediaSurfacesKHR returns `CL_SUCCESS` if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is `NULL` then the function does nothing and returns `CL_SUCCESS`. Otherwise it returns one of the following errors:

- `CL_INVALID_VALUE` if *num_objects* is zero and *mem_objects* is not a `NULL` value or if *num_objects* > 0 and *mem_objects* is `NULL`.
- `CL_INVALID_MEM_OBJECT` if memory objects in *mem_objects* are not valid OpenCL memory objects or if memory objects in *mem_objects* have not been created from media surfaces.
- `CL_INVALID_COMMAND_QUEUE` if *command_queue* is not a valid command-queue.
- `CL_INVALID_CONTEXT` if context associated with *command_queue* was not created from a device that can share the media surface referenced by *mem_objects*.
- `CL_DX9_MEDIA_SURFACE_ALREADY_ACQUIRED_KHR` if memory objects in *mem_objects* have previously been acquired using **clEnqueueAcquireDX9MediaSurfacesKHR** but have not been released using **clEnqueueReleaseDX9MediaSurfacesKHR**.
- `CL_INVALID_EVENT_WAIT_LIST` if *event_wait_list* is `NULL` and *num_events_in_wait_list* > 0, or *event_wait_list* is not `NULL` and *num_events_in_wait_list* is 0, or if event objects in *event_wait_list* are not valid events.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

```
cl_int clEnqueueReleaseDX9MediaSurfacesKHR(
    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);
```

is used to release OpenCL memory objects that have been created from media surfaces. The media surfaces are released by the OpenCL context associated with *command_queue*.

OpenCL memory objects created from media surfaces which have been acquired by OpenCL must be released by OpenCL before they may be accessed by the media adapter API. Accessing a media surface while its corresponding OpenCL memory object is acquired is in error and will result in undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

If `CL_CONTEXT_INTEROP_USER_SYNC` is not specified as `CL_TRUE` during context creation, **clEnqueueReleaseDX9MediaSurfacesKHR** provides the synchronization guarantee that any calls to media adapter APIs involving the interop device(s) used in the OpenCL context made after the call to **clEnqueueReleaseDX9MediaSurfacesKHR** will not start executing until after all events in *event_wait_list* are complete and all work already submitted to *command_queue* completes execution. If the context was created with properties specifying `CL_CONTEXT_INTEROP_USER_SYNC` as `CL_TRUE`, the user is responsible for guaranteeing that any media adapter API calls involving the interop device(s) used in the OpenCL context made after **clEnqueueReleaseDX9MediaSurfacesKHR** will not start executing until after event returned by **clEnqueueReleaseDX9MediaSurfacesKHR** reports completion.

num_objects is the number of memory objects to be released in *mem_objects*.

mem_objects is a pointer to a list of OpenCL memory objects that were created from media surfaces.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is `NULL`, then this particular command does not wait on any event to complete. If *event_wait_list* is `NULL`, *num_events_in_wait_list* must be 0. If *event_wait_list* is not `NULL`, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0.

event returns an event object that identifies this command and can be used to query or wait for this command to complete. If *event* is `NULL` or the enqueue is unsuccessful, no event will be created and therefore it will not be possible to query the status of this command or to wait for this command to complete. If *event_wait_list* and *event* are not `NULL`, *event* must not refer to an element of the *event_wait_list* array.

clEnqueueReleaseDX9MediaSurfacesKHR returns `CL_SUCCESS` if the function is executed successfully. If *num_objects* is 0 and <*mem_objects*> is `NULL` the function does nothing and returns `CL_SUCCESS`. Otherwise it returns one of the following errors:

- `CL_INVALID_VALUE` if *num_objects* is zero and *mem_objects* is not a `NULL` value or if *num_objects* > 0 and *mem_objects* is `NULL`.
- `CL_INVALID_MEM_OBJECT` if memory objects in *mem_objects* are not valid OpenCL memory objects or if memory objects in *mem_objects* have not been created from valid media surfaces.
- `CL_INVALID_COMMAND_QUEUE` if *command_queue* is not a valid command-queue.
- `CL_INVALID_CONTEXT` if context associated with *command_queue* was not created from a media object.
- `CL_DX9_MEDIA_SURFACE_NOT_ACQUIRED_KHR` if memory objects in *mem_objects* have not previously been acquired using **clEnqueueAcquireDX9MediaSurfacesKHR**, or have been released using **clEnqueueReleaseDX9MediaSurfacesKHR** since the last time that they were acquired.
- `CL_INVALID_EVENT_WAIT_LIST` if *event_wait_list* is `NULL` and *num_events_in_wait_list* > 0, or *event_wait_list* is not `NULL` and *num_events_in_wait_list*> is 0, or if event objects in *event_wait_list* are not valid events.

- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

15.7.5. Event Command Types for Sharing Memory Objects created from Media Surfaces

The following table describes the event command types for the OpenCL commands to acquire and release OpenCL memory objects that have been created from media surfaces:

Table 40. List of supported event command types

Events Created By	Event Command Type
clEnqueueAcquireDX9MediaSurfacesKHR	CL_COMMAND_ACQUIRE_DX9_MEDIA_SURFACES_KHR
clEnqueueReleaseDX9MediaSurfacesKHR	CL_COMMAND_RELEASE_DX9_MEDIA_SURFACES_KHR

15.7.6. Surface formats for Media Surface Sharing

This section includes the D3D surface formats that are supported when the adapter type is one of the Direct 3D lineage . Using a D3D surface format not listed here is an error. To extend the use of this extension to support media adapters beyond DirectX 9 tables similar to the ones in this section will need to be defined for the surface formats supported by the new media adapter. All implementations that support this extension are required to support the NV12 surface format, the other surface formats supported are the same surface formats that the adapter you are sharing with supports as long as they are listed in the table *YUV FourCC codes and corresponding OpenCL image format* or in the table *Direct3D formats and corresponding OpenCL image formats*.

Table 41. YUV FourCC codes and corresponding OpenCL image format

FOUR CC code	CL image format (channel order, channel data type)
FOURCC('N','V','1','2'), Plane 0	CL_R, CL_UNORM_INT8
FOURCC('N','V','1','2'), Plane 1	CL_RG, CL_UNORM_INT8
FOURCC('Y','V','1','2'), Plane 0	CL_R, CL_UNORM_INT8
FOURCC('Y','V','1','2'), Plane 1	CL_R, CL_UNORM_INT8
FOURCC('Y','V','1','2'), Plane 2	CL_R, CL_UNORM_INT8

In the table *YUV FourCC codes and corresponding OpenCL image format* above, NV12 Plane 0 corresponds to the luminance (Y) channel and Plane 1 corresponds to the UV channels. The YV12 Plane 0 corresponds to the Y channel, Plane 1 corresponds to the V channel and Plane 2 corresponds to the U channel. Note that the YUV formats map to CL_R and CL_RG but do not perform any YUV to RGB conversion and vice-versa.

Table 42. Direct3D formats and corresponding OpenCL image formats

D3D format	CL image format (channel order, channel data type)
D3DFMT_R32F	CL_R, CL_FLOAT
D3DFMT_R16F	CL_R, CL_HALF_FLOAT
D3DFMT_L16	CL_R, CL_UNORM_INT16
D3DFMT_A8	CL_A, CL_UNORM_INT8
D3DFMT_L8	CL_R, CL_UNORM_INT8
D3DFMT_G32R32F	CL_RG, CL_FLOAT
D3DFMT_G16R16F	CL_RG, CL_HALF_FLOAT
D3DFMT_G16R16	CL_RG, CL_UNORM_INT16
D3DFMT_A8L8	CL_RG, CL_UNORM_INT8
D3DFMT_A32B32G32R32F	CL_RGBA, CL_FLOAT
D3DFMT_A16B16G16R16F	CL_RGBA, CL_HALF_FLOAT
D3DFMT_A16B16G16R16	CL_RGBA, CL_UNORM_INT16
D3DFMT_A8B8G8R8	CL_RGBA, CL_UNORM_INT8
D3DFMT_X8B8G8R8	CL_RGBA, CL_UNORM_INT8
D3DFMT_A8R8G8B8	CL_BGRA, CL_UNORM_INT8
D3DFMT_X8R8G8B8	CL_BGRA, CL_UNORM_INT8

Note: The D3D9 format names in the table above seem to imply that the order of the color channels are switched relative to OpenCL but this is not the case. For example, the layout of channels for each pixel for D3DFMT_A32FB32FG32FR32F is the same as CL_RGBA, CL_FLOAT.

Chapter 16. Depth Images

This section describes the **cl_khr_depth_images** extension.

This extension adds support for depth images.

This extension became a core feature in OpenCL 2.0.

16.1. General Information

16.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

16.2. Additions to Chapter 5 of the OpenCL 1.2 Specification

This extension adds the following new image formats for depth images to *tables 5.6 and 5.7* of the OpenCL 1.2 specification.

Enum values that can be specified in channel_order
`CL_DEPTH`. This format can only be used if channel data type = `CL_UNORM_INT16` or `CL_FLOAT`.

Image Channel Data Type	Description
`CL_UNORM_INT16`	Each channel component is a normalized unsigned 16-bit integer value
`CL_FLOAT`	Each channel component is a single precision floating-point value

This extension adds the following new image format to the minimum list of supported image formats described in *table 5.8*:

Table 43. Required Image Formats for **cl_khr_depth_images**

num_channels	channel_order	channel_data_type
1	`CL_DEPTH`	`CL_UNORM_INT16` `CL_FLOAT`

NOTE:

Depth image objects can be initialized, read and written using the appropriate CL APIs i.e. **clEnqueueReadImage**, **clEnqueueWriteImage**, **clEnqueueCopyImage**, **clEnqueueCopyImageToBuffer**, **clEnqueueCopyBufferToImage**, **clEnqueueMapImage** and **clEnqueueFillImage**.

For **clEnqueueFillImage**, the fill color is a 4-component value where the R component refers to the depth value if the image format is `CL_DEPTH`. The fill color will be converted to the appropriate image channel format and order associated with image.

Update text that describes arg value argument to **clSetKernelArg** with the following:

If the kernel argument is declared to be of type image2d_depth_t or image2d_array_depth t, the arg_value entry will be a pointer to a depth image or depth image array object.

Add the following error condition for **clSetKernelArg**:

`CL_INVALID_MEM_OBJECT` for an argument declared to be a depth image or a depth image array and the argument value specified in arg_value does not follow the rules described above for a depth memory object or memory array object argument.

16.3. Additions to Chapter 6 of the OpenCL 1.2 Specification

Add the following new data types to *table 6.3* in *section 6.1.3* of the OpenCL 1.2 specification:

Type	Description
image2d_depth_t	A 2D depth image. Refer to *section 6.12.14* for a detailed description of the built-in functions that use this type.
image2d_array_depth_t	A 2D depth image array. Refer to *section 6.12.14* for a detailed description of the built-in functions that use this type.

Add the following to the bulleted list in section 6.12.14.1.1 - Determining the border color:

- If the image channel order is `CL_DEPTH`, the border value is `0.0f`.

Add the following built-in functions to section 6.12.14.2 - Built-in Image Read Functions:

Function	Description
float **read_imagef**(read_only image2d_depth_t *image*, sampler_t *sampler*, int2 *coord*) float **read_imagef**(read_only image2d_depth_t *image*, sampler_t *sampler*, float2 *coord*)	Use the coordinate (*coord.x*, *coord.y*) to do an element lookup in the 2D depth image object specified by *image*. **read_imagef** returns a floating-point value in the range [0.0, 1.0] for depth image objects created with *image_channel_data_type* set to `CL_UNORM_INT16` or `CL_UNORM_INT24`. **read_imagef** returns a floating-point value for depth image objects created with *image_channel_data_type* set to `CL_FLOAT`. The **read_imagef** calls that take integer coordinates must use a sampler with filter mode set to `CLK_FILTER_NEAREST`, normalized coordinates set to `CLK_NORMALIZED_COORDS_FALSE` and addressing mode set to `CLK_ADDRESS_CLAMP_TO_EDGE`, `CLK_ADDRESS_CLAMP` or `CLK_ADDRESS_NONE`; otherwise the values returned are undefined. Values returned by **read_imagef** for depth image objects with *image_channel_data_type* values not specified in the description above are undefined.
float **read_imagef**(read_only image2d_array_depth_t *image*, sampler_t *sampler*, int4 *coord*) float **read_imagef**(read_only image2d_array_depth_t *image*, sampler_t *sampler*, float4 *coord*)	Use *coord.xy* to do an element lookup in the 2D image identified by *coord.z* in the 2D depth image array specified by *image*. **read_imagef** returns a floating-point value in the range [0.0, 1.0] for depth image objects created with *image_channel_data_type* set to `CL_UNORM_INT16` or `CL_UNORM_INT24`. **read_imagef** returns a floating-point value for depth image objects created with *image_channel_data_type* set to `CL_FLOAT`. The **read_imagef** calls that take integer coordinates must use a sampler with filter mode set to `CLK_FILTER_NEAREST`, normalized coordinates set to `CLK_NORMALIZED_COORDS_FALSE` and addressing mode set to `CLK_ADDRESS_CLAMP_TO_EDGE`, `CLK_ADDRESS_CLAMP` or `CLK_ADDRESS_NONE`; otherwise the values returned are undefined. Values returned by **read_imagef** for image objects with *image_channel_data_type* values not specified in the description above are undefined.

Add the following built-in functions to section 6.12.14.3 - Built-in Image Sampler-less Read Functions:

Function	Description
float **read_imagef**(image2d_depth_t *image*, int2 *coord*)	Use the coordinate (*coord.x*, *coord.y*) to do an element lookup in the 2D depth image object specified by *image*. **read_imagef** returns a floating-point value in the range [0.0, 1.0] for depth image objects created with *image_channel_data_type* set to `CL_UNORM_INT16` or `CL_UNORM_INT24`. **read_imagef** returns a floating-point value for depth image objects created with *image_channel_data_type* set to `CL_FLOAT`. Values returned by **read_imagef** for image objects with *image_channel_data_type* values not specified in the description above are undefined.
float **read_imagef**(image2d_array_depth_t *image*, int4 *coord*)	Use *coord.xy* to do an element lookup in the 2D image identified by *coord.z* in the 2D depth image array specified by *image*. **read_imagef** returns a floating-point value in the range [0.0, 1.0] for depth image objects created with *image_channel_data_type* set to `CL_UNORM_INT16` or `CL_UNORM_INT24`. **read_imagef** returns a floating-point value for depth image objects created with *image_channel_data_type* set to `CL_FLOAT`. Values returned by **read_imagef** for image objects with *image_channel_data_type* values not specified in the description above are undefined.

Add the following built-in functions to section 6.12.14.4 – Built-in Image Write Functions:

Function	Description
void **write_imagef**(image2d_depth_t *image*, int2 *coord*, float *depth*)	Write *depth* value to location specified by *coord.xy* in the 2D depth image object specified by *image*. Appropriate data format conversion to the specified image format is done before writing the depth value. *coord.x* and *coord.y* are considered to be unnormalized coordinates, and must be in the range [0, image width-1], and [0, image height-1], respectively. **write_imagef** can only be used with image objects created with *image_channel_data_type* set to `CL_UNORM_INT16`, `CL_UNORM_INT24` or `CL_FLOAT`. Appropriate data format conversion will be done to convert depth value from a floating-point value to actual data format associated with the image. The behavior of **write_imagef**, **write_imagei** and **write_imageui** for image objects created with *image_channel_data_type* values not specified in the description above or with (*x*, *y*) coordinate values that are not in the range [0, image width-1] and [0, image height-1], respectively, is undefined.
void **write_imagef**(image2d_array_depth_t *image*, int4 *coord*, float *depth*)	Write *depth* value to location specified by *coord.xy* in the 2D image identified by *coord.z* in the 2D depth image array specified by *image*. Appropriate data format conversion to the specified image format is done before writing the depth value. *coord.x*, *coord.y* and *coord.z* are considered to be unnormalized coordinates, and must be in the range [0, image width-1], [0, image height-1], and [0, image number of layers-1], respectively. **write_imagef** can only be used with image objects created with *image_channel_data_type* set to `CL_UNORM_INT16`, `CL_UNORM_INT24` or `CL_FLOAT`. Appropriate data format conversion will be done to convert depth valye from a floating-point value to actual data format associated with the image. The behavior of **write_imagef**, **write_imagei** and **write_imageui** for image objects created with *image_channel_data_type* values not specified in the description above or with (*x*, *y*, *z*) coordinate values that are not in the range [0, image width-1], [0, image height-1], [0, image number of layers-1], respectively, is undefined.

Add the following built-in functions to section 6.12.14.5 – Built-in Image Query Functions:

Function	Description
int **get_image_width**(image2d_depth_t *image*) int **get_image_width**(image2d_array_depth_t *image*)	Return the image width in pixels.
int **get_image_height**(image2d_depth_t *image*) int **get_image_height**(image2d_array_depth_t *image*)	Return the image height in pixels.
int **get_image_channel_data_type**(image2d_depth_t *image*) int **get_image_channel_data_type**(image2d_array_depth_t *image*)	Return the channel data type. Valid values are: `CLK_UNORM_INT16` `CLK_FLOAT`
int **get_image_channel_order**(image2d_depth_t *image*) int **get_image_channel_order**(image2d_array_depth_t *image*)	Return the image channel order. Valid values are: `CLK_DEPTH`
int2 **get_image_dim**(image2d_depth_t *image*) int2 **get_image_dim**(image2d_array_depth_t *image*)	Return the 2D image width and height as an int2 type. The width is returned in the *x* component, and the height in the *y* component.
size_t **get_image_array_size**(image2d_array_depth_t *image*)	Return the number of images in the 2D image array.

Add the following text below the table in section 6.12.14.6 - Mapping image channels to color values returned by read_image and color values passed to write_image to image channels:

For `CL_DEPTH` images, a scalar value is returned by **read_imagef** or supplied to **write_imagef**.

Chapter 17. Sharing OpenGL and OpenGL ES Depth and Depth-Stencil Images

This section describes the **cl_khr_gl_depth_images** extension. The **cl_khr_gl_depth_images** extends OpenCL / OpenGL sharing (the cl_khr_gl_sharing_extension) defined in Creating OpenCL Memory Objects from OpenGL Objects to allow an OpenCL image to be created from an OpenGL depth or depth-stencil texture.

17.1. General Information

17.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

17.2. Additions to Chapter 5 of the OpenCL 2.2 Specification

The **cl_khr_gl_depth_images** extension extends OpenCL / OpenGL sharing by allowing an OpenCL depth image to be created from an OpenGL depth or depth-stencil texture. Depth images with an image channel order of CL_DEPTH_STENCIL can only be created using the **clCreateFromGLTexture** API.

This extension adds the following new image format for depth-stencil images to *table 5.6 and 5.7* of the OpenCL 2.2 specification.

Enum values that can be specified in channel_order
CL_DEPTH_STENCIL. This format can only be used if channel data type = CL_UNORM_INT24 or CL_FLOAT.

Image Channel Data Type	Description
CL_UNORM_INT24	Each channel component is a normalized unsigned 24-bit integer value
CL_FLOAT	Each channel component is a single precision floating-point value

This extension adds the following new image format to the minimum list of supported image formats described in *tables 5.8.a* and *5.8.b*.

Table 44. Required Image Formats for **cl_khr_gl_depth_images**

num_channels	channel_order	channel_data_type	read / write

1	CL_DEPTH_STENCIL	CL_UNORM_INT24 CL_FLOAT	read only

For the image format given by channel order of CL_DEPTH_STENCIL and channel data type of CL_UNORM_INT24, the depth is stored as an unsigned normalized 24-bit value.

For the image format given by channel order of CL_DEPTH_STENCIL and channel data type of CL_FLOAT, each pixel is two 32-bit values. The depth is stored as a single precision floating-point value followed by the stencil which is stored as a 8-bit integer value.

The stencil value cannot be read or written using the **read_imagef** and **write_imagef** built-in functions in an OpenCL kernel.

Depth image objects with an image channel order equal to CL_DEPTH_STENCIL cannot be used as arguments to clEnqueueReadImage, clEnqueueWriteImage, clEnqueueCopyImage, clEnqueueCopyImageToBuffer, clEnqueueCopyBufferToImage, clEnqueueMapImage and clEnqueueFillImage and will return a CL_INVALID_OPERATION error.

17.3. Additions to the OpenCL Extension Specification

The following new image formats are added to the table of OpenGL internal formats and corresponding OpenCL internal formats in the OpenCL extension specification. If an OpenGL texture object with an internal format in this table is successfully created by OpenGL, then there is guaranteed to be a mapping to one of the corresponding OpenCL image format(s) in that table.

GL internal format	CL image format (channel order, channel data type)
GL_DEPTH_COMPONENT32F	CL_DEPTH, CL_FLOAT
GL_DEPTH_COMPONENT16	CL_DEPTH, CL_UNORM_INT16
GL_DEPTH24_STENCIL8	CL_DEPTH_STENCIL, CL_UNORM_INT24
GL_DEPTH32F_STENCIL8	CL_DEPTH_STENCIL, CL_FLOAT

Chapter 18. Creating OpenCL Memory Objects from OpenGL MSAA Textures

This extension extends the OpenCL / OpenGL sharing (the cl_khr_gl_sharing_extension) defined in Creating OpenCL Memory Objects from OpenGL Objects to allow an OpenCL image to be created from an OpenGL multi-sampled (a.k.a. MSAA) texture (color or depth).

This extension name is **cl_khr_gl_msaa_sharing**. This extension requires **cl_khr_gl_depth_images**.

18.1. General Information

18.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

18.2. Additions to the OpenCL Extension Specification

Allow *texture_target* argument to **clCreateFromGLTexture** to be GL_TEXTURE_2D_MULTISAMPLE or GL_TEXTURE_2D_MULTISAMPLE_ARRAY.

If *texture_target* is GL_TEXTURE_2D_MULTISAMPLE, **clCreateFromGLTexture** creates an OpenCL 2D multi-sample image object from an OpenGL 2D multi-sample texture.

If *texture_target* is GL_TEXTURE_2D_MULTISAMPLE_ARRAY, **clCreateFromGLTexture** creates an OpenCL 2D multi-sample array image object from an OpenGL 2D multi-sample texture.

Multi-sample OpenCL image objects can only be read from a kernel. Multi-sample OpenCL image objects cannot be used as arguments to clEnqueueReadImage , clEnqueueWriteImage, clEnqueueCopyImage, clEnqueueCopyImageToBuffer, clEnqueueCopyBufferToImage, clEnqueueMapImage and clEnqueueFillImage and will return a CL_INVALID_OPERATION error.

Add the following entry to the table describing OpenGL texture info that may be queried with clGetGLTextureInfo:

cl_gl_texture_info	**Return Type**	**Info. returned in *param_value***
CL_GL_NUM_SAMPLES	GLsizei	The *samples* argument passed to **glTexImage2DMultisample** or **glTexImage3DMultisample**. If *image* is not a MSAA texture, 1 is returned.

18.3. Additions to Chapter 5 of the OpenCL 2.2 Specification

The formats described in tables 5.8.a and 5.8.b of the OpenCL 2.2 specification and the additional formats described in required image formats for cl_khr_gl_depth_images also support OpenCL images created from a OpenGL multi-sampled color or depth texture.

Update text that describes arg value argument to clSetKernelArg with the following:

“If the argument is a multi-sample 2D image, the *arg_value* entry must be a pointer to a multi-sample image object. If the argument is a multi-sample 2D depth image, the *arg_value* entry must be a pointer to a multisample depth image object. If the argument is a multi-sample 2D image array, the *arg_value* entry must be a pointer to a multi-sample image array object. If the argument is a multi-sample 2D depth image array, the *arg_value* entry must be a pointer to a multi-sample depth image array object.”

Updated error code text for clSetKernelArg is:

Add the following text:

“CL_INVALID_MEM_OBJECT for an argument declared to be a multi-sample image, multi-sample image array, multi-sample depth image or a multi-sample depth image array and the argument value specified in *arg_value* does not follow the rules described above for a depth memory object or memory array object argument.”

18.4. Additions to Chapter 6 of the OpenCL 2.2 Specification

Add the following new data types to *table 6.3* in *section 6.1.3* of the OpenCL 2.2 specification:

Type	Description
image2d_msaa_t	A 2D multi-sample color image. Refer to *section 6.13.14* for a detailed description of the built-in functions that use this type.
image2d_array_msaa_t	A 2D multi-sample color image array. Refer to *section 6.13.14* for a detailed description of the built-in functions that use this type.
image2d_msaa_depth_t	A 2D multi-sample depth image. Refer to *section 6.13.14* for a detailed description of the built-in functions that use this type.
image2d_array_msaa_depth_t	A 2D multi-sample depth image array. Refer to *section 6.13.14* for a detailed description of the built-in functions that use this type.

Add the following built-in functions to section 6.13.14.3 — Built-in Image Sampler-less Read Functions:

```
float4 read_imagef(
    image2d_msaa_t image,
    int2 coord,
    int sample)
```

Use the coordinate *(coord.x, coord.y)* and *sample* to do an element lookup in the 2D image object specified by *image*.

read_imagef returns floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16.

read_imagef returns floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to CL_SNORM_INT8, or CL_SNORM_INT16.

read_imagef returns floating-point values for image objects created with *image_channel_data_type* set to CL_HALF_FLOAT or CL_FLOAT.

Values returned by **read_imagef** for image objects with *image_channel_data_type* values not specified in the description above are undefined.

```
int4 read_imagei(image2d_msaa_t image,
                 int2 coord,
                 int sample)

uint4 read_imageui(image2d_msaa_t image,
                   int2 coord,
                   int sample)
```

Use the coordinate *(coord.x, coord.y)* and *sample* to do an element lookup in the 2D image object specified by *image*.

read_imagei and **read_imageui** return unnormalized signed integer and unsigned integer values respectively. Each channel will be stored in a 32-bit integer.

read_imagei can only be used with image objects created with *image_channel_data_type* set to one of the following values:

- CL_SIGNED_INT8,
- CL_SIGNED_INT16, and
- CL_SIGNED_INT32.

If the *image_channel_data_type* is not one of the above values, the values returned by **read_imagei** are undefined.

read_imageui can only be used with image objects created with *image_channel_data_type* set to one of the following values:

* CL_UNSIGNED_INT8,
* CL_UNSIGNED_INT16, and
* CL_UNSIGNED_INT32.

If the *image_channel_data_type* is not one of the above values, the values returned by **read_imageui** are undefined.

```
float4 read_imagef(image2d_array_msaa_t image,
                   int4 coord,
                   int sample)
```

Use *coord.xy* and *sample* to do an element lookup in the 2D image identified by *coord.z* in the 2D image array specified by *image*.

read_imagef returns floating-point values in the range [0.0 … 1.0] for image objects created with *image_channel_data_type* set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16.

read_imagef returns floating-point values in the range [-1.0 … 1.0] for image objects created with *image_channel_data_type* set to CL_SNORM_INT8, or CL_SNORM_INT16.

read_imagef returns floating-point values for image objects created with *image_channel_data_type* set to CL_HALF_FLOAT or CL_FLOAT.

Values returned by **read_imagef** for image objects with *image_channel_data_type* values not specified in the description above are undefined.

```
int4 read_imagei(image2d_array_msaa_t image,
                 int4 coord,
                 int sample)

uint4 read_imageui(image2d_array_msaa_t image,
                   int4 coord,
                   int sample)
```

Use *coord.xy* and *sample* to do an element lookup in the 2D image identified by *coord.z* in the 2D image array specified by *image*.

read_imagei and **read_imageui** return unnormalized signed integer and unsigned integer values respectively. Each channel will be stored in a 32-bit integer.

read_imagei can only be used with image objects created with *image_channel_data_type* set to one of the following values:

* CL_SIGNED_INT8,
* CL_SIGNED_INT16, and
* CL_SIGNED_INT32.

If the *image_channel_data_type* is not one of the above values, the values returned by **read_imagei** are undefined.

read_imageui can only be used with image objects created with *image_channel_data_type* set to one of the following values:

- CL_UNSIGNED_INT8,
- CL_UNSIGNED_INT16, and
- CL_UNSIGNED_INT32.

If the *image_channel_data_type* is not one of the above values, the values returned by **read_imageui** are undefined.

```
float read_imagef(image2d_msaa_depth_t image,
                  int2 coord,
                  int sample)
```

Use the coordinate *(coord.x, coord.y)* and *sample* to do an element lookup in the 2D depth image object specified by *image*.

read_imagef returns a floating-point value in the range [0.0 … 1.0] for depth image objects created with *image_channel_data_type* set to CL_UNORM_INT16 or CL_UNORM_INT24.

read_imagef returns a floating-point value for depth image objects created with *image_channel_data_type* set to CL_FLOAT.

Values returned by **read_imagef** for image objects with *image_channel_data_type* values not specified in the description above are undefined.

```
float read_imagef(image2d_array_msaaa_depth_t image,
                  int4 coord,
                  int sample)
```

Use *coord.xy* and *sample* to do an element lookup in the 2D image identified by *coord.z* in the 2D depth image array specified by *image*.

read_imagef returns a floating-point value in the range [0.0 … 1.0] for depth image objects created with *image_channel_data_type* set to CL_UNORM_INT16 or CL_UNORM_INT24.

read_imagef returns a floating-point value for depth image objects created with *image_channel_data_type* set to CL_FLOAT.

Values returned by **read_imagef** for image objects with *image_channel_data_type* values not specified in the description above are undefined.

Note: When a multisample image is accessed in a kernel, the access takes one vector of integers describing which pixel to fetch and an integer corresponding to the sample numbers describing which sample within the pixel to fetch. sample identifies the sample position in the multi-sample image.

For best performance, we recommend that *sample* be a literal value so it is known at compile time and the OpenCL compiler can perform appropriate optimizations for multi-sample reads on the device.

No standard sampling instructions are allowed on the multisample image. Accessing a coordinate outside the image and/or a sample that is outside the number of samples associated with each pixel in the image is undefined

Add the following built-in functions to section 6.13.14.5 — Built-in Image Query Functions:

```
int get_image_width(image2d_msaa_t image)

int get_image_width(image2d_array_msaa_t image)

int get_image_width(image2d_msaa_depth_t image)

int get_image_width(image2d_array_msaa_depth_t image)
```

Return the image width in pixels.

```
int get_image_height(image2d_msaa_t image)

int get_image_height(image2d_array_msaa_t image)

int get_image_height(image2d_msaa_depth_t image)

int get_image_height(image2d_array_msaa_depth_t image)
```

Return the image height in pixels.

```
int get_image_channel_data_type(image2d_msaa_t image)

int get_image_channel_data_type(image2d_array_msaa_t image)

int get_image_channel_data_type(image2d_msaa_depth_t image)

int get_image_channel_data_type(image2d_array_msaa_depth_t image)
```

Return the channel data type.

```
int get_image_channel_order(image2d_msaa_t image)

int get_image_channel_order(image2d_array_msaa_t image)

int get_image_channel_order(image2d_msaa_depth_t image)
```

```
int get_image_channel_order(image2d_array_msaa_depth_t image)
```

Return the image channel order.

```
int2 get_image_dim(image2d_msaa_t image)

int2 get_image_dim(image2d_array_msaa_t image)

int2 get_image_dim(image2d_msaa_depth_t image)

int2 get_image_dim(image2d_array_msaa_depth_t image)
```

Return the 2D image width and height as an int2 type. The width is returned in the *x* component, and the height in the *y* component.

```
size_t get_image_array_size(image2d_array_msaa_depth_t image)
```

Return the number of images in the 2D image array.

```
int get_image_num_samples(image2d_msaa_t image)

int get_image_num_samples(image2d_array_msaa_t image)

int get_image_num_samples(image2d_msaa_depth_t image)

int get_image_num_samples(image2d_array_msaa_depth_t image)
```

Return the number of samples in the 2D MSAA image

Chapter 19. Creating OpenCL Event Objects from EGL Sync Objects

19.1. Overview

This section describes the **cl_khr_egl_event** extension. This extension allows creating OpenCL event objects linked to EGL fence sync objects, potentially improving efficiency of sharing images and buffers between the two APIs. The companion **EGL_KHR_cl_event** extension provides the complementary functionality of creating an EGL sync object from an OpenCL event object.

19.2. General Information

19.2.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

19.3. New Procedures and Functions

```
cl_event clCreateEventFromEGLSyncKHR(cl_context context,
                                     CLeglSyncKHR sync,
                                     CLeglDisplayKHR display,
                                     cl_int *errcode_ret);
```

19.4. New Tokens

Returned by clCreateEventFromEGLSyncKHR if *sync* is not a valid EGLSyncKHR handle created with respect to EGLDisplay *display*:

```
CL_INVALID_EGL_OBJECT_KHR
```

Returned by **clGetEventInfo** when *param_name* is CL_EVENT_COMMAND_TYPE:

```
CL_COMMAND_EGL_FENCE_SYNC_OBJECT_KHR
```

19.5. Additions to Chapter 5 of the OpenCL 2.2 Specification

Add following to the fourth paragraph of *section 5.11* (prior to the description of **clWaitForEvents**):

"Event objects can also be used to reflect the status of an EGL fence sync object. The sync object in turn refers to a fence command executing in an EGL client API command stream. This provides another method of coordinating sharing of EGL / EGL client API objects with OpenCL. Completion of EGL / EGL client API commands may be determined by placing an EGL fence command after commands using eglCreateSyncKHR, creating an event from the resulting EGL sync object using clCreateEventFromEGLSyncKHR and then specifying it in the *event_wait_list* of a clEnqueueAcquire*** command. This method may be considerably more efficient than calling operations like glFinish, and is referred to as *explicit synchronization*. The application is responsible for ensuring the command stream associated with the EGL fence is flushed to ensure the CL queue is submitted to the device. Explicit synchronization is most useful when an EGL client API context bound to another thread is accessing the memory objects."

Add CL_COMMAND_EGL_FENCE_SYNC_OBJECT_KHR to the valid *param_value* values returned by **clGetEventInfo** for *param_name* CL_EVENT_COMMAND_TYPE (in the third row and third column of *table 5.22*).

Add new *subsection 5.11.2*:

"`**5.11.2 Linking Event Objects to EGL Synchronization Objects**

An event object may be created by linking to an EGL **sync object**. Completion of such an event object is equivalent to waiting for completion of the fence command associated with the linked EGL sync object.

The function

```
cl_event clCreateEventFromEGLSyncKHR(cl_context context,
                                     CLeglSyncKHR sync,
                                     CLeglDisplayKHR display,
                                     cl_int *errcode_ret)
```

creates a linked event object.

context is a valid OpenCL context created from an OpenGL context or share group, using the **cl_khr_gl_sharing** extension.

sync is the name of a sync object of type EGL_SYNC_FENCE_KHR created with respect to EGLDisplay *display*.

clCreateEventFromEGLSyncKHR returns a valid OpenCL event object and *errcode_ret* is set to CL_SUCCESS if the event object is created successfully. Otherwise, it returns a `NULL` value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context, or was not created from a GL context.
- CL_INVALID_EGL_OBJECT_KHR if *sync* is not a valid EGLSyncKHR object of type EGL_SYNC_FENCE_KHR created with respect to EGLDisplay *display*.

The parameters of an event object linked to an EGL sync object will return the following values when queried with **clGetEventInfo**:

- The CL_EVENT_COMMAND_QUEUE of a linked event is `NULL`, because the event is not associated with any OpenCL command-queue.
- The CL_EVENT_COMMAND_TYPE of a linked event is CL_COMMAND_EGL_FENCE_SYNC_OBJECT_KHR, indicating that the event is associated with a EGL sync object, rather than an OpenCL command.
- The CL_EVENT_COMMAND_EXECUTION_STATUS of a linked event is either CL_SUBMITTED, indicating that the fence command associated with the sync object has not yet completed, or CL_COMPLETE, indicating that the fence command has completed.

clCreateEventFromEGLSyncKHR performs an implicit **clRetainEvent** on the returned event object. Creating a linked event object also places a reference on the linked EGL sync object. When the event object is deleted, the reference will be removed from the EGL sync object.

Events returned from **clCreateEventFromEGLSyncKHR** may only be consumed by **clEnqueueAcquire*** commands. Passing such events to any other CL API that enqueues commands will generate a CL_INVALID_EVENT error.`"

19.6. Additions to the OpenCL Extension Specification

Replace the second paragraph of Synchronizing OpenCL and OpenGL Access to Shared Objects with:

"`Prior to calling **clEnqueueAcquireGLObjects**, the application must ensure that any pending EGL or EGL client API operations which access the objects specified in *mem_objects* have completed.

If the **cl_khr_egl_event** extension is supported and the EGL context in question supports fence sync objects, *explicit synchronization* can be achieved as set out in *section 5.7.1*.

If the **cl_khr_egl_event** extension is not supported, completion of EGL client API commands may be determined by issuing and waiting for completion of commands such as glFinish or vgFinish on all client API contexts with pending references to these objects. Some implementations may offer other efficient synchronization methods. If such methods exist they will be described in platform-specific documentation.

Note that no synchronization methods other than glFinish and vgFinish are portable between all EGL client API implementations and all OpenCL implementations. While this is the only way to ensure completion that is portable to all platforms, these are expensive operation and their use should be avoided if the cl_khr_egl_event extension is supported on a platform.`"

19.7. Issues

Most issues are shared with **cl_khr_gl_event** and are resolved as described in that extension.

1. Should we support implicit synchronization?

   RESOLVED: No, as this may be very difficult since the synchronization would not be with EGL, it would be with currently bound EGL client APIs. It would be necessary to know which client APIs might be bound, to validate that they're associated with the EGLDisplay associated with the

OpenCL context, and to reach into each such context.

2. Do we need to have typedefs to use EGL handles in OpenCL?

   RESOLVED Using typedefs for EGL handles.

3. Should we restrict which CL APIs can be used with this cl_event?

   RESOLVED Use is limited to clEnqueueAcquire*** calls only.

4. What is the desired behaviour for this extension when EGLSyncKHR is of a type other than EGL_SYNC_FENCE_KHR?

   RESOLVED This extension only requires support for EGL_SYNC_FENCE_KHR. Support of other types is an implementation choice, and will result in CL_INVALID_EGL_OBJECT_KHR if unsupported.

Chapter 20. Creating OpenCL Memory Objects from EGL Images

20.1. Overview

This section describes the **cl_khr_egl_image** extension. This extension provides a mechanism to creating OpenCL memory objects from from EGLImages.

20.2. General Information

20.2.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

20.3. New Procedures and Functions

```
cl_mem clCreateFromEGLImageKHR(cl_context context,
                               CLeglDisplayKHR display,
                               CLeglImageKHR image,
                               cl_mem_flags flags,
                               const cl_egl_image_properties_khr *properties,
                               cl_int *errcode_ret);

cl_int clEnqueueAcquireEGLObjectsKHR(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 clEnqueueReleaseEGLObjectsKHR(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)
```

20.4. New Tokens

New error codes:

```
CL_EGL_RESOURCE_NOT_ACQUIRED_KHR
```

```
CL_INVALID_EGL_OBJECT_KHR
```

New command types:

```
CL_COMMAND_ACQUIRE_EGL_OBJECTS_KHR
CL_COMMAND_RELEASE_EGL_OBJECTS_KHR
```

20.5. Additions to Chapter 5 of the OpenCL 2.2 Specification

In section 5.2.4, add the following text after the paragraph defining clCreateImage:

"`The function

```
cl_mem clCreateFromEGLImageKHR(cl_context context,
                               CLeglDisplayKHR display,
                               CLeglImageKHR image,
                               cl_mem_flags flags,
                               const cl_egl_image_properties_khr *properties,
                               cl_int *errcode_ret);
```

creates an EGLImage target of type cl_mem from the EGLImage source provided as *image*.

display should be of type EGLDisplay, cast into the type CLeglDisplayKHR.

image should be of type EGLImageKHR, cast into the type CLeglImageKHR. Assuming no errors are generated in this function, the resulting image object will be an EGLImage target of the specified EGLImage *image*. The resulting cl_mem is an image object which may be used normally by all OpenCL operations. This maps to an image2d_t type in OpenCL kernel code.

flags is a bit-field that is used to specify usage information about the memory object being created.

The possible values for *flags* are: CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM _READ_WRITE.

For OpenCL 1.2 *flags* also accepts: CL_MEM_HOST_WRITE_ONLY, CL_MEM_HOST_READ_ONLY or CL_MEM_HOST_NO_ACCESS.

This extension only requires support for CL_MEM _READ_ONLY, and for OpenCL 1.2 CL_MEM_HOST_NO_ACCESS. For OpenCL 1.1, a CL_INVALID_OPERATION will be returned for images which do not support host mapping.

If the value passed in *flags* is not supported by the OpenCL implementation it will return CL_INVALID_VALUE. The accepted *flags* may be dependent upon the texture format used.

properties specifies a list of property names and their corresponding values. Each property name is immediately followed by the corresponding desired value. The list is terminated with 0. No properties are currently supported with this version of the extension. *properties* can be `NULL`.

clCreateFromEGLImageKHR returns a valid non-zero OpenCL image object and *errcode_ret* is set to CL_SUCCESS if the image object is created successfully. Otherwise, it returns a `NULL` value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid OpenCL context.
- CL_INVALID_VALUE if *properties* contains invalid values, if *display* is not a valid display object or if *flags* are not in the set defined above.
- CL_INVALID_EGL_OBJECT_KHR if *image* is not a valid EGLImage object.
- CL_IMAGE_FORMAT_NOT_SUPPORTED if the OpenCL implementation is not able to create a cl_mem compatible with the provided CLeglImageKHR for an implementation-dependent reason (this could be caused by, but not limited to, reasons such as unsupported texture formats, etc).
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_INVALID_OPERATION if there are no devices in *context* that support images (i.e. CL_DEVICE_IMAGE_SUPPORT specified in table 4.3 is CL_FALSE) or if the flags passed are not supported for that image type.`"

20.5.1. Lifetime of Shared Objects

An OpenCL memory object created from an EGL image remains valid according to the lifetime behavior as described in EGL_KHR_image_base.

“Any EGLImage siblings exist in any client API context”

For OpenCL this means that while the application retains a reference on the cl_mem (the EGL sibling), the image remains valid.

20.5.2. Synchronizing OpenCL and EGL Access to Shared Objects

In order to ensure data integrity, the application is responsible for synchronizing access to shared CL/EGL objects by their respective APIs. Failure to provide such synchronization may result in race conditions and other undefined behavior including non-portability between implementations.

Prior to calling clEnqueueAcquireEGLObjectsKHR, the application must ensure that any pending operations which access the objects specified in mem_objects have completed. This may be accomplished in a portable way by ceasing all client operations on the resource, and issuing and waiting for completion of a glFinish command on all GL contexts with pending references to these objects. Implementations may offer more efficient synchronization methods, such as synchronization primitives or fence operations.

Similarly, after calling clEnqueueReleaseEGLImageObjects, the application is responsible for ensuring that any pending OpenCL operations which access the objects specified in mem_objects have completed prior to executing subsequent commands in other APIs which reference these objects. This may be accomplished in a portable way by calling clWaitForEvents with the event object returned by clEnqueueReleaseGLObjects, or by calling clFinish. As above, some implementations may offer more efficient methods.

Attempting to access the data store of an EGLImage object after it has been acquired by OpenCL and before it has been released will result in undefined behavior. Similarly, attempting to access a shared EGLImage object from OpenCL before it has been acquired by the OpenCL command-queue or after it has been released, will result in undefined behavior.

20.5.3. Sharing memory objects created from EGL resources between EGLDisplays and OpenCL contexts

The function

```
cl_int clEnqueueAcquireEGLObjectsKHR(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)
```

is used to acquire OpenCL memory objects that have been created from EGL resources. The EGL objects are acquired by the OpenCL context associated with *command_queue* and can therefore be used by all command-queues associated with the OpenCL context.

OpenCL memory objects created from EGL resources must be acquired before they can be used by any OpenCL commands queued to a command-queue. If an OpenCL memory object created from a EGL resource is used while it is not currently acquired by OpenCL, the behavior is undefined. Implementations may fail the execution of commands attempting to use that OpenCL memory object and set their associated event's execution status to `CL_EGL_RESOURCE_NOT_ACQUIRED_KHR`.

command_queue is a valid command-queue.

num_objects is the number of memory objects to be acquired in *mem_objects*.

mem_objects is a pointer to a list of OpenCL memory objects that were created from EGL resources, within the context associate with command_queue.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is `NULL`, then this particular command does not wait on any event to complete. If *event_wait_list* is `NULL`, *num_events_in_wait_list* must be 0. If *event_wait_list* is not `NULL`, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points.

event returns an event object that identifies this command and can be used to query or wait for this command to complete. If *event* is `NULL` or the enqueue is unsuccessful, no event will be created and therefore it will not be possible to query the status of this command or to wait for this command to complete. If *event_wait_list* and *event* are not NULL, *event* must not refer to an element of the *event_wait_list* array.

clEnqueueAcquireEGLObjectsKHR returns CL_SUCCESS if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is NULL then the function does nothing and returns CL_SUCCESS. Otherwise it returns one of the following errors:

- CL_INVALID_VALUE if *num_objects* is zero and *mem_objects* is not a NULL value or if num_objects > 0 and mem_objects is NULL.
- CL_INVALID_MEM_OBJECT if memory objects in *mem_objects* are not valid OpenCL memory objects in the context associated with *command_queue*.
- CL_INVALID_EGL_OBJECT_KHR if memory objects in *mem_objects* have not been created from EGL resources.
- CL_INVALID_COMMAND_QUEUE if *command_queue* is not a valid command-queue.
- CL_INVALID_EVENT_WAIT_LIST if *event_wait_list* is NULL and *num_events_in_wait_list* > 0, or *event_wait_list* is not NULL and *num_events_in_wait_list* is 0, or if event objects in *event_wait_list* are not valid events.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

```
cl_int clEnqueueReleaseEGLObjectsKHR(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)
```

is used to release OpenCL memory objects that have been created from EGL resources. The EGL objects are released by the OpenCL context associated with <command_queue>.

OpenCL memory objects created from EGL resources which have been acquired by OpenCL must be released by OpenCL before they may be accessed by EGL or by EGL client APIs. Accessing a EGL resource while its corresponding OpenCL memory object is acquired is in error and will result in undefined behavior, including but not limited to possible OpenCL errors, data corruption, and program termination.

command_queue is a valid command-queue.

num_objects is the number of memory objects to be acquired in *mem_objects*.

mem_objects is a pointer to a list of OpenCL memory objects that were created from EGL resources, within the context associate with command_queue.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed. If *event_wait_list* is `NULL`, then this particular command does not wait on any event to complete. If *event_wait_list* is `NULL`, *num_events_in_wait_list* must be 0. If *event_wait_list* is not `NULL`, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points.

event returns an event object that identifies this command and can be used to query or wait for this command to complete. If *event* is `NULL` or the enqueue is unsuccessful, no event will be created and therefore it will not be possible to query the status of this command or to wait for this command to complete. If *event_wait_list* and *event* are not `NULL`, *event* must not refer to an element of the *event_wait_list* array.

clEnqueueReleaseEGLObjectsKHR returns CL_SUCCESS if the function is executed successfully. If *num_objects* is 0 and *mem_objects* is `NULL` then the function does nothing and returns CL_SUCCESS. Otherwise it returns one of the following errors:

- CL_INVALID_VALUE if *num_objects* is zero and *mem_objects* is not a `NULL` value or if num_objects > 0 and mem_objects is `NULL`.
- CL_INVALID_MEM_OBJECT if memory objects in *mem_objects* are not valid OpenCL memory objects in the context associated with *command_queue*.
- CL_INVALID_EGL_OBJECT_KHR if memory objects in *mem_objects* have not been created from EGL resources.
- CL_INVALID_COMMAND_QUEUE if *command_queue* is not a valid command-queue.
- CL_INVALID_EVENT_WAIT_LIST if *event_wait_list* is `NULL` and *num_events_in_wait_list* > 0, or *event_wait_list* is not `NULL` and *num_events_in_wait_list* is 0, or if event objects in *event_wait_list* are not valid events.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

20.5.4. Event Command Types for Sharing memory objects created from EGL resources

The following table describes the event command types for the OpenCL commands to acquire and release OpenCL memory objects that have been created from EGL resources:

Table 45. List of supported event command types

Events Created By	Event Command Type
clEnqueueAcquireEGLObjectsKHR	`CL_COMMAND_ACQUIRE_EGL_OBJECTS_KHR`
clEnqueueReleaseEGLObjectsKHR	`CL_COMMAND_RELEASE_EGL_OBJECTS_KHR`

20.6. Issues

1. This extension does not support reference counting of the images, so the onus is on the application to behave sensibly and not release the underlying cl_mem object while the EGLImage is still being used.
2. In order to ensure data integrity, the application is responsible for synchronizing access to shared CL/EGL image objects by their respective APIs. Failure to provide such synchronization may result in race conditions and other undefined behavior. This may be accomplished by calling clWaitForEvents with the event objects returned by any OpenCL commands which use the shared image object or by calling clFinish.
3. Currently CL_MEM_READ_ONLY is the only supported flag for *flags*.

   RESOLVED: Implementation will now return an error if writing to a shared object that is not supported rather than disallowing it entirely.
4. Currently restricted to 2D image objects.
5. What should happen for YUV color-space conversion, multi plane images, and chroma-siting, and channel mapping?

   RESOLVED: YUV is no longer explicitly described in this extension. Before this removal the behavior was dependent on the platform. This extension explicitly leaves the YUV layout to the platform and EGLImage source extension (i.e. is implementation specific). Colorspace conversion must be applied by the application using a color conversion matrix.

   The expected extension path if YUV color-space conversion is to be supported is to introduce a YUV image type and provide overloaded versions of the read_image built-in functions.

   Getting image information for a YUV image should return the original image size (non quantized size) when all of Y U and V are present in the image. If the planes have been separated then the actual dimensionality of the separated plane should be reported. For example with YUV 4:2:0 (NV12) with a YUV image of 256x256, the Y only image would return 256x256 whereas the UV only image would return 128x128.
6. Should an attribute list be used instead?

   RESOLVED: function has been changed to use an attribute list.
7. What should happen for EGLImage extensions which introduce formats without a mapping to an OpenCL image channel data type or channel order?

   RESOLVED: This extension does not define those formats. It is expected that as additional EGL extensions are added to create EGL images from other sources, an extension to CL will be introduced where needed to represent those image types.
8. What are the guarantees to synchronization behavior provided by the implementation?

   The basic portable form of synchronization is to use a clFinish, as is the case for GL interop. In addition implementations which support the synchronization extensions cl_khr_egl_event and EGL_KHR_cl_event can interoperate more efficiently as described in those extensions.

Chapter 21. Creating a 2D Image From A Buffer

This section describes the **cl_khr_image2d_from_buffer** extension.

This extension allows a 2D image to be created from an existing OpenCL buffer memory object.

This extension became a core feature in OpenCL 2.0.

21.1. General Information

21.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

21.2. Additions to Chapter 4 of the OpenCL 1.2 Specification

The following table entry describes the additions to *table 4.3,* which allows applications to query the configuration information using **clGetDeviceInfo** for an OpenCL device that supports creating a 2D image from a buffer.

Device Info	Return Type	Description
`CL_DEVICE_IMAGE_PITCH_ALIGNMENT_KHR`	`cl_uint`	The row pitch alignment size in pixels for images created from a buffer. The value returned must be a power of 2. If the device does not support images, this value should be 0.
`CL_DEVICE_IMAGE_BASE_ADDRESS_ALIGNMENT_KHR`	`cl_uint`	This query should be used when an image is created from a buffer which was created using `CL_MEM_USE_HOST_PTR`. The value returned must be a power of 2. This query specifies the minimum alignment in pixels of the *host_ptr* specified to **clCreateBuffer**. If the device does not support images, this value should be 0.

21.3. Additions to Chapter 5 of the OpenCL 1.2 Specification

Add to Section 5.3.1: Creating Image Objects:

A 2D image can be created from a buffer by specifying a *buffer* object in the *image_desc* passed to **clCreateImage** for an *image_type* equal to `CL_MEM_OBJECT_IMAGE2D`. When the 2D image from buffer is created, the client must specify the width, height and image format (i.e. channel order and channel data type). If these are not specified, **clCreateImage** returns a NULL value with *errcode_ret* set to `CL_INVALID_IMAGE_FORMAT_DESCRIPTOR`. The pitch can be optionally specified. If the pitch is not specified, the pitch is computed as width × bytes per pixel based on the image format.

The pitch specified (or computed if pitch specified is 0) must be a multiple of the maximum of the `CL_DEVICE_IMAGE_PITCH_ALIGNMENT_KHR` value for all devices in the context associated with the *buffer* that support images. Otherwise, **clCreateImage** returns a NULL value with *errcode_ret* set to `CL_INVALID_IMAGE_FORMAT_DESCRIPTOR`.

If the *buffer* was created with `CL_MEM_USE_HOST_PTR`, the *host_ptr* specified to **clCreateBuffer** must be aligned to the maximum of the `CL_DEVICE_IMAGE_BASE_ADDRESS_ALIGNMENT_KHR` value for all devices in the context associated with the *buffer* that support images. Otherwise, **clCreateImage** returns a NULL value with *errcode_ret* set to `CL_INVALID_IMAGE_FORMAT_DESCRIPTOR`.

The minimum list of supported image formats described in *table 5.8* of the OpenCL 1.2 specification must be supported for 2D images created from a buffer.

The OpenCL runtime APIs that operate on images (i.e. **clEnqueueReadImage**, **clEnqueueWriteImage**, **clEnqueueFillImage**, **clEnqueueCopyImage**, **clEnqueueCopyImageToBuffer**, **clEnqueueCopyBufferToImage** and **clEnqueueMapImage**) are supported for a 2D image created from a buffer.

When the contents of a buffer object data store are modified, those changes are reflected in the contents of the 2D image object and vice-versa at corresponding synchronization points. The *image_height* × *image_row_pitch* specified in *image_desc* must be less than or equal to the size of the buffer object data store.

> Concurrent reading from, writing to, and copying between both a buffer object and the 2D image object associated with the buffer object is undefined. Only reading from both a buffer object and 2D image object associated with the buffer object is defined. A 2D image and a 2D image created from a buffer use the same image type in OpenCL C (`image2d_t`). The image built-ins functions described in *section 6.12.14.2*, *6.12.14.3*, *6.12.14.4* and *6.12.14.5* for `image2d_t` behave the same way for a 2D image and a 2D image from a buffer.

Chapter 22. Local and Private Memory Initialization

Memory is allocated in various forms in OpenCL both explicitly (global memory) or implicitly (local, private memory). This allocation so far does not provide a straightforward mechanism to initialize the memory on allocation. In other words what is lacking is the equivalent of calloc for the currently supported malloc like capability. This functionality is useful for a variety of reasons including ease of debugging, application controlled limiting of visibility to previous contents of memory and in some cases, optimization.

This extension adds support for initializing local and private memory before a kernel begins execution. This extension name is **cl_khr_initialize_memory**.

22.1. General Information

22.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

22.2. Additions to Chapter 4 of the OpenCL 2.2 Specification

Add a new context property to *table 4.5* in *section 4.4*.

Table 46. List of supported context creation properties by **clCreateContext**

Context Property	Property value	Description
`CL_CONTEXT_MEMORY_INITIALIZE_KHR`	`cl_context_memory_initialize_khr`	Describes which memory types for the context must be initialized. This is a bit-field, where the following values are currently supported: `CL_CONTEXT_MEMORY_INITIALIZE_LOCAL_KHR` — Initialize local memory to zeros. `CL_CONTEXT_MEMORY_INITIALIZE_PRIVATE_KHR` — Initialize private memory to zeros.

22.3. Additions to Chapter 6 of the OpenCL 2.2 Specification

Updates to *section 6.9* — Restrictions

If the context is created with `CL_CONTEXT_MEMORY_INITIALIZE_KHR`, appropriate memory locations as specified by the bit-field is initialized with zeroes, prior to the start of execution of any kernel. The driver chooses when, prior to kernel execution, the initialization of local and/or private memory is performed. The only requirement is there should be no values set from outside the context, which can be read during a kernel execution.

Chapter 23. Terminating OpenCL Contexts

Today, OpenCL provides an API to release a context. This operation is done only after all queues, memory object, programs and kernels are released, which in turn might wait for all ongoing operations to complete. However, there are cases in which a fast release is required, or release operation cannot be done, as commands are stuck in mid execution. An example of the first case can be program termination due to exception, or quick shutdown due to low power. Examples of the second case are when a kernel is running too long, or gets stuck, or it may result from user action which makes the results of the computation unnecessary.

In many cases, the driver or the device is capable of speeding up the closure of ongoing operations when the results are no longer required in a much more expedient manner than waiting for all previously enqueued operations to finish.

This extension implements a new query to check whether a device can terminate an OpenCL context and adds an API to terminate a context.

The extension name is **cl_khr_terminate_context**.

23.1. General Information

23.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

23.2. Additions to Chapter 4 of the OpenCL 2.2 Specification

Add a new device property to *table 4.3* in *section 4.2*.

Table 47. List of supported param_names by **clGetDeviceInfo**

Device Info	Return Type	Description
`CL_DEVICE_TERMINATE_CAPABILITY_KHR`	`cl_device_terminate_capability_khr`	Describes the termination capability of the OpenCL device. This is a bit-field, where the following values are currently supported: `CL_DEVICE_TERMINATE_CAPABILITY_CONTEXT_KHR` - Indicates that context termination is supported.

Add a new context property to *table 4.5* in *section 4.4*.

Table 48. List of supported context creation properties by **clCreateContext**

Context Property	Property value	Description
CL_CONTEXT_TERMINATE_KHR	cl_bool	Specifies whether the context can be terminated. The default value is CL_FALSE.

CL_CONTEXT_TERMINATE_KHR can be specified in the context properties only if all devices associated with the context support the ability to support context termination (i.e. CL_DEVICE_TERMINATE_CAPABILITY_CONTEXT_KHR is set for CL_DEVICE_TERMINATE_CAPABILITY_KHR). Otherwise, context creation fails with error code of CL_INVALID_PROPERTY.

The new function

```
cl_int clTerminateContextKHR(
    cl_context context);
```

terminates all pending work associated with the context and renders all data owned by the context invalid. It is the responsibility of the application to release all objects associated with the context being terminated.

When a context is terminated:

- The execution status of enqueued commands will be CL_CONTEXT_TERMINATED_KHR. Event objects can be queried using **clGetEventInfo**. Event callbacks can be registered and registered event callbacks will be called with *event_command_status* set to CL_CONTEXT_TERMINATED_KHR. **clWaitForEvents** will return as immediately for commands associated with event objects specified in event_list. The status of user events can be set. Event objects can be retained and released. **clGetEventProfilingInfo** returns CL_PROFILING_INFO_NOT_AVAILABLE.
- The context is considered to be terminated. A callback function registered when the context was created will be called. Only queries, retain and release operations can be performed on the context. All other APIs that use a context as an argument will return CL_CONTEXT_TERMINATED_KHR.
- The contents of the memory regions of the memory objects is undefined. Queries, registering a destructor callback, retain and release operations can be performed on the memory objects.
- Once a context has been terminated, all OpenCL API calls that create objects or enqueue commands will return CL_CONTEXT_TERMINATED_KHR. APIs that release OpenCL objects will continue to operate as though **clTerminateContextKHR** was not called.
- The behavior of callbacks will remain unchanged, and will report appropriate error, if executing after termination of context. This behavior is similar to enqueued commands, after the command-queue has become invalid.

clTerminateContextKHR returns CL_SUCCESS if the function is executed successfully. Otherwise, it returns one of the following errors:

- CL_INVALID_CONTEXT if *context* is not a valid OpenCL context.
- CL_CONTEXT_TERMINATED_KHR if *context* has already been terminated.
- CL_INVALID_OPERATION if *context* was not created with CL_CONTEXT_TERMINATE_KHR set to CL_TRUE.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL

implementation on the device.

- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

An implementation that supports this extension must be able to terminate commands currently executing on devices or queued across all command-queues associated with the context that is being terminated. The implementation cannot implement this extension by waiting for currently executing (or queued) commands to finish execution on devices associated with this context (i.e. doing a **clFinish**).

Chapter 24. Standard Portable Intermediate Representation Binaries

This extension adds the ability to create an OpenCL program object from a Standard Portable Intermediate Representation (SPIR) instance. A SPIR instance is a vendor-neutral non-source representation for OpenCL C programs.

The extension name is **cl_khr_spir**. This extension has been superseded by the SPIR-V intermediate representation, which is supported by the **cl_khr_il_program** extension, and is a core feature in OpenCL 2.1.

24.1. General Information

24.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

24.2. Additions to Chapter 4 of the OpenCL 2.2 Specification

Add a new device property to *table 4.3* in *section 4.2*:

Table 49. List of supported param_names by **clGetDeviceInfo**

Device Info	Return Type	Description
`CL_DEVICE_SPIR_VERSIONS`	`char`[]	A space separated list of SPIR versions supported by the device. For example, returning `"1.2"` in this query implies that SPIR version 1.2 is supported by the implementation.

24.3. Additions to Chapter 5 of the OpenCL 2.2 Specification

Additions to *section 5.8.1* — Creating Program Objects:

"**clCreateProgramWithBinary** can be used to load a SPIR binary. Once a program object has been created from a SPIR binary, **clBuildProgram** can be called to build a program executable or **clCompileProgram** can be called to compile the SPIR binary."

Modify the `CL_PROGRAM_BINARY_TYPE` entry in *table 5.14* for **clGetProgramBuildInfo** to add a potential value `CL_PROGRAM_BINARY_TYPE_INTERMEDIATE`:

Table 50. List of supported param_names by **clGetProgramBuildInfo**

Program Build Info	Return Type	Description
`CL_PROGRAM_BINARY_TYPE`	`cl_program_binary_type`	`CL_PROGRAM_BINARY_TYPE_INTERMEDIATE` — An intermediate (non-source) representation for the program is loaded as a binary. The program must be further processed with **clCompileProgram** or **clBuildProgram**. If processed with **clCompileProgram**, the result will be a binary of type `CL_PROGRAM_BINARY_TYPE_COMPILED_OBJECT` or `CL_PROGRAM_BINARY_TYPE_LIBRARY`. If processed with **clBuildProgram**, the result will be a binary of type `CL_PROGRAM_BINARY_TYPE_EXECUTABLE`.

Additions to *section 5.8.4* — Compiler Options:

"The compile option `-x spir` must be specified to indicate that the binary is in SPIR format, and the compile option `-spir-std` must be used to specify the version of the SPIR specification that describes the format and meaning of the binary. For example, if the binary is as described in SPIR version 1.2, then `-spir-std=1.2` must be specified. Failing to specify these compile options may result in implementation-defined behavior."

Additions to *section 5.8.5* — Separate Compilation and Linking of Programs:

Replace this error for **clCompileProgram**:

- `CL_INVALID_OPERATION` if *program* has no source or IL available, i.e. it has not been created with **clCreateProgramWithSource** or **clCreateProgramWithIL**.

with:

- `CL_INVALID_OPERATION` if *program* has no source or IL available, i.e. it has not been created with **clCreateProgramWithSource** or **clCreateProgramWithIL** or **clCreateProgramWithBinary** where `-x spir` is present in *options*.

Additions to *section 5.9.3* — Kernel Object Queries:

Modify following text in **clGetKernelArgInfo** from:

"Kernel argument information is only available if the program object associated with *kernel* is created with **clCreateProgramWithSource** and the program executable is built with the -cl-kernel -arg-info option specified in *options* argument to **clBuildProgram** or **clCompileProgram**."

to:

"Kernel argument information is only available if the program object associated with *kernel* is created with **clCreateProgramWithSource** and the program executable is built with the `-cl-kernel-arg-info option` specified in *options* argument to **clBuildProgram** or **clCompileProgram**, or if the program object associated with *kernel* is created with **clCreateProgramWithBinary** and the program executable is built with the `-cl-kernel-arg-info` and `-x spir` options specified in *options* argument to **clBuildProgram** or **clCompileProgram**."

Chapter 25. Intermediate Language Programs

This section describes the **cl_khr_il_program** extension.

This extension adds the ability to create programs with intermediate language (IL), usually SPIR-V. Further information about the format and contents of SPIR-V may be found in the SPIR-V Specification. Information about how SPIR-V modules behave in the OpenCL environment may be found in the OpenCL SPIR-V Environment Specification.

This functionality described by this extension is a core feature in OpenCL 2.1.

25.1. General Information

25.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

25.2. New Procedures and Functions

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

25.3. New Tokens

Accepted as a new *param_name* argument to **clGetDeviceInfo**:

```
CL_DEVICE_IL_VERSION_KHR
```

Accepted as a new *param_name* argument to **clGetProgramInfo**:

```
CL_PROGRAM_IL_KHR
```

25.4. Additions to Chapter 3 of the OpenCL 2.0 Specification

In section 3.1, replace the fourth paragraph with:

"Programmers provide programs in the form of intermediate language binaries (usually SPIR-V), OpenCL C source strings, or implementation-defined binary objects. The OpenCL platform provides a compiler to translate programs represented as intermediate language binaries or OpenCL C source strings into device program executables. The compiler may be *online* or *offline*. An *online compiler* is available during host program execution using standard APIs. An *offline compiler* is invoked outside of host program control, using platform-specific methods. The OpenCL runtime allows developers to get a previously compiled device program executable and to load and execute a previously compiled device program executable."

25.5. Additions to Chapter 4 of the OpenCL 2.0 Specification

Add to Table 4.3 - OpenCL Device Queries:

Table 4.3 List of supported param_names by **clGetDeviceInfo**

Device Info	Return Type	Description
`CL_DEVICE_IL_VERSION_KHR`	char[]	The intermediate languages that are be supported by **clCreateProgramWithILKHR** for this device. Returns a space separated list of IL version strings of the form: <IL_Prefix>_<Major_version>.<Minor_version> A device that supports the **cl_khr_il_program** extension must support the “SPIR-V” IL prefix.

25.6. Additions to Chapter 5 of the OpenCL 2.0 Specification

Add to Section 5.8.1: Creating Program Objects:

"The function

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

creates a new program object for *context* using the *length* bytes of intermediate language pointed to by *il*.

context must be a valid OpenCL context.

il is a pointer to a *length*-byte block of memory containing intermediate langage.

length is the length of the block of memory pointed to by *il*.

errcode_ret will return an appropriate error code. If *errcode_ret* is NULL, no error code is returned.

clCreateProgramWithILKHR returns a valid non-zero program object and *errcode_ret* is set to `CL_SUCCESS` if the program object is created successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- `CL_INVALID_CONTEXT` if *context* is not a valid context
- `CL_INVALID_VALUE` if *il* is NULL or if *length* is zero.
- `CL_INVALID_VALUE` if the *length*-byte block of memory pointed to by *il* does not contain well-formed intermediate language.
- `CL_OUT_OF_RESOURCES` if there is a failure to allocate resources required by the OpenCL implementation on the device.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host."

Add to Section 5.8.2: Building Program Executables:

Add the following to the description of the *options* parameter to **clBuildProgram**:

"Certain options are ignored when *program* is created with IL."

Additionally, replace the error:

- `CL_INVALID_OPERATION` if *program* was not created with **clCreateProgramWithSource** or **clCreateProgramWithBinary**.

with:

- `CL_INVALID_OPERATION` if *program* was not created with **clCreateProgramWithSource**, **clCreateProgramWithILKHR** or **clCreateProgramWithBinary**.

Add to Section 5.8.3: Separate Compilation and Linking of Programs:

Add the following to the description of the *options* parameter to **clCompileProgram**:

"Certain options are ignored when *program* is created with IL."

Additionally, replace the error:

- `CL_INVALID_OPERATION` if *program* has no source i.e. it has not been created with **clCreateProgramWithSource**.

with:

- `CL_INVALID_OPERATION` if *program* was not created with **clCreateProgramWithSource** or **clCreateProgramWithILKHR**.

Add to Section 5.8.4.1: Preprocessor Options,
Add to Section 5.8.4.2: Math Intrinsic Options (for -cl-single-precision-constant-only),
Add to Section 5.8.4.3: Optimization Options,
Add to Section 5.8.4.4: Options to Request or Suppress Warnings, and
Add to Section 5.8.4.5: Options Controlling the OpenCL C Version:

"These options are ignored for programs created with IL."

Change one entry and add one new entry to Table 5.17 **clGetProgramInfo** parameter queries:

Table 5.17 List of supported param_names by **clGetProgramInfo**

Program Info	Return Type	Description
`CL_PROGRAM_SOURCE`	`char`[]	Return the program source code specified by **clCreateProgramWithSource**. The source string returned is a concatenation of all source strings specified to **clCreateProgramWithSource** with a null terminator. The concatenation strips any nulls in the original source strings. If program is created using **clCreateProgramWithBinary**, **clCreateProgramWithBuiltInKernels**, or **clCreateProgramWithILKHR** a null string or the appropriate program source code is returned depending on whether or not the program source code is stored in the binary. The actual number of characters that represents the program source code including the null terminator is returned in *param_value_size_ret*.
`CL_PROGRAM_IL_KHR`	`unsigned char`[]	Returns the program IL for programs created with **clCreateProgramWithILKHR**. If program is created with **clCreateProgramWithSource**, **clCreateProgramWithBinary**, or **clCreateProgramWithBuiltInKernels**, the memory pointed to by *param_value* will be unchanged and *param_value_size_ret* will be set to zero.

Chapter 26. Creating Command-Queues with Properties

26.1. Overview

The section describes the **cl_khr_create_command_queue** extension.

This extension allows OpenCL 1.x devices to support an equivalent of the **clCreateCommandQueueWithProperties** API that was added in OpenCL 2.0. This allows OpenCL 1.x devices to support other optional extensions or features that use the **clCreateCommandQueueWithProperties** API to specify additional command-queue properties that cannot be specified using the OpenCL 1.x **clCreateCommandQueue** API.

No new command-queue properties are required by this extension. Applications may use the existing `CL_DEVICE_QUEUE_PROPERTIES` query to determine command-queue properties that are supported by the device.

OpenCL 2.x devices may support this extension for compatibility. In this scenario, the function added by this extension will have the same capabilities as the core **clCreateCommandQueueWithProperties** API. Applications that only target OpenCL 2.x devices should use the core OpenCL 2.x **clCreateCommandQueueWithProperties** API instead of this extension API.

26.2. General Information

26.2.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

26.3. New API Functions

```
cl_command_queue clCreateCommandQueueWithPropertiesKHR(
    cl_context context,
    cl_device_id device,
    const cl_queue_properties_khr *properties,
    cl_int *errcode_ret);
```

26.4. New API Types

```
typedef cl_properties cl_queue_properties_khr;
```

26.5. Modifications to the OpenCL 1.2 Specification

(Add to Table 5.2 for `CL_QUEUE_PROPERTIES` in Section 5.1)

Table 5.2 List of supported param_names by **clGetCommandQueueInfo**

Queue Info	Return Type	Description
`CL_QUEUE_PROPERTIES`	`cl_command_queue_properties`	Returns the currently specified properties for the command-queue. These properties are specified by the *properties* argument in **clCreateCommandQueue**, or by the `CL_QUEUE_PROPERTIES` property value in **clCreateCommandQueueWithPropertiesKHR**.

(Add a new Section 5.1.1, Creating Command-Queues With Properties)

The function

```
cl_command_queue clCreateCommandQueueWithPropertiesKHR(
    cl_context context,
    cl_device_id device,
    const cl_queue_properties_khr* properties,
    cl_int* errcode_ret);
```

allows creation of a command-queue from an array of properties for the specified device.

context must be a valid OpenCL context.

device must be a device or sub-device associated with *context*. It can either be in the list of devices and sub-devices specified when *context* is created using **clCreateContext** or be a root device with the same device type as specified when *context* is created using **clCreateContextFromType**.

properties specifies a list of properties for the command-queue and their corresponding values. Each property name is immediately followed by the corresponding desired value. The list is terminated with 0. The list of supported properties is described in the table below. If a supported property and its value is not specified in *properties*, its default value will be used. *properties* can be NULL in which case the default values for supported command-queue properties will be used.

Table X.Y List of supported param_names by **clCreateCommandQueueWithPropertiesKHR**

Queue Properties	Property Value	Description
CL_QUEUE_PROPERTIES	cl_bitfield	This is a bitfield and can be set to a combination of the following values: CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE - Determines whether the commands queued in the command-queue are executed in-order or out-of-order. If set, the commands in the command-queue are executed out-of-order. Otherwise, commands are executed in-order. CL_QUEUE_PROFILING_ENABLE - Enable or disable profiling of commands in the command-queue. If set, the profiling of commands is enabled. Otherwise, profiling of commands is disabled. If CL_QUEUE_PROPERTIES is not specified an in-order command-queue that does not support profiling of commands is created for the specified device.

errcode_ret will return an appropriate error code. If *errcode_ret* is NULL, no error code is returned.

clCreateCommandQueueWithPropertiesKHR returns a valid non-zero command-queue and *errcode_ret* is set to CL_SUCCESS if the command-queue is created successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context.
- CL_INVALID_DEVICE if *device* is not a valid device or is not associated with *context*.
- CL_INVALID_VALUE if values specified in *properties* are not valid.
- CL_INVALID_QUEUE_PROPERTIES if values specified in *properties* are valid but are not supported by the device.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL

implementation on the host.

Chapter 27. Device Enqueue Local Argument Types

This extension allows arguments to blocks that are passed to the **enqueue_kernel** built-in function to be pointers to any type (built-in or user-defined) in local memory, instead of requiring arguments to blocks to be pointers to void in local memory.

The name of this extension is **cl_khr_device_enqueue_local_arg_types**.

27.1. General Information

27.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

27.2. Additions to Chapter 6 of the OpenCL 2.0 C Specification

Modify the second paragraph of Section 6.13.17: Enqueuing Kernels:

"The following table describes the list of built-in functions that can be used to enqueue a kernel. We use the generic type name gentype to indicate 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, which can be used as the type of the pointee of the arguments of the kernel enqueue functions listed in table 6.31."

Then, replace all occurrences of local void * in table 6.31 with local gentype *. For example:

```
int enqueue_kernel(queue_t queue,
                   kernel_enqueue_flags_t flags,
                   const ndrange_t ndrange,
                   void (^block)(local gentype *, ...),
                   uint size0, ... )
```

Additionally, replace all occurrences of local void* in table 6.33 with local gentype *. For example:

```
uint get_kernel_work_group_size(
                    void (^block)(local gentype *, ...))
```

Chapter 28. Sub-groups

This section describes the **cl_khr_subgroups** extension.

This extension adds support for implementation-controlled groups of work items, known as sub-groups. Sub-groups behave similarly to work-groups and have their own sets of built-ins and synchronization primitives. Sub-groups within a work-group are independent, may make forward progress with respect to each other, and may map to optimized hardware structures where that makes sense.

Sub-groups were promoted to a core feature in OpenCL 2.1, however note that:

- The sub-group OpenCL C built-in functions described by this extension must still be accessed as an OpenCL C extension in OpenCL 2.1.
- Sub-group independent forward progress is an optional device property in OpenCL 2.1, see `CL_DEVICE_SUB_GROUP_INDEPENDENT_FORWARD_PROGRESS`.

28.1. General Information

28.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

28.2. Additions to Chapter 3 of the OpenCL 2.0 Specification

28.3. Additions to section 3.2 — Execution Model

Within a work-group work-items may be divided into sub-groups. The mapping of work-items to sub-groups is implementation-defined and may be queried at runtime. While sub-groups may be used in multi-dimensional work-groups, each sub-group is 1-dimensional and any given work-item may query which sub-group it is a member of.

Work items are mapped into sub-groups through a combination of compile-time decisions and the parameters of the dispatch. The mapping to sub-groups is invariant for the duration of a kernel's execution, across dispatches of a given kernel with the same launch parameters, and from one work-group to another within the dispatch (excluding the trailing edge work-groups in the presence of non-uniform work-group sizes). In addition, all sub-groups within a work-group will be the same size, apart from the sub-group with the maximum index which may be smaller if the size of the work-group is not evenly divisible by the size of the sub-group.

Sub-groups execute concurrently within a given work-group and make independent forward progress with respect to each other even in the absence of work-group barrier operations. Sub-groups are able to internally synchronize using barrier operations without synchronizing with each other.

In the degenerate case, with the extension enabled, a single sub-group must be supported for each work-group. In this situation all sub-group scope functions alias their work-group level equivalents.

28.4. Additions to Chapter 5 of the OpenCL 2.0 Specification

The function

```
cl_int clGetKernelSubGroupInfoKHR(
    cl_kernel in_kernel,
    cl_device_id in_device,
    cl_kernel_sub_group_info param_name,
    size_t input_value_size,
    const void* input_value,
    size_t param_value_size,
    void* param_value,
    size_t* param_value_size_ret);
```

returns information about the kernel object.

kernel specifies the kernel object being queried.

device identifies a specific device in the list of devices associated with *kernel*. The list of devices is the list of devices in the OpenCL context that is associated with *kernel*. If the list of devices associated with *kernel* is a single device, *device* can be a `NULL` value.

param_name specifies the information to query. The list of supported *param_name* types and the information returned in *param_value* by **clGetKernelSubGroupInfoKHR** is described in the Kernel Object Sub-group Queries table.

input_value_size is used to specify the size in bytes of memory pointed to by *input_value*. This size must be == size of input type as described in the table below.

input_value is a pointer to memory where the appropriate parameterization of the query is passed from. If *input_value* is `NULL`, it is ignored.

param_value is a pointer to memory where the appropriate result being queried is returned. If *param_value* is `NULL`, it is ignored.

param_value_size is used to specify the size in bytes of memory pointed to by *param_value*. This size must be ≥ size of return type as described in the Kernel Object Sub-group Queries table.

param_value_size_ret returns the actual size in bytes of data being queried by *param_name*. If *param_value_size_ret* is `NULL`, it is ignored.

Table 51. List of supported param_names by **clGetKernelSubGroupInfoKHR**

Kernel Sub-group Info	Input Type	Return Type	Description
CL_KERNEL_MAX_SUB_GROUP_SIZE_FOR_NDRANGE_KHR	size_t*	size_t	Returns the maximum sub-group size for this kernel. All sub-groups must be the same size, while the last sub-group in any work-group (i.e. the sub-group with the maximum index) could be the same or smaller size. The *input_value* must be an array of size_t values corresponding to the local work size parameter of the intended dispatch. The number of dimensions in the ND-range will be inferred from the value specified for *input_value_size*.

Kernel Sub-group Info	Input Type	Return Type	Description
CL_KERNEL_SUB_GROUP_COUNT_FOR_NDRANGE_KHR	size_t*	size_t	Returns the number of sub-groups that will be present in each work-group for a given local work size. All workgroups, apart from the last work-group in each dimension in the presence of non-uniform work-group sizes, will have the same number of sub-groups. The *input_value* must be an array of size_t values corresponding to the local work size parameter of the intended dispatch. The number of dimensions in the ND-range will be inferred from the value specified for *input_value_size*.

clGetKernelSubGroupInfoKHR returns CL_SUCCESS if the function is executed successfully. Otherwise, it returns one of the following errors:

- CL_INVALID_DEVICE if *device* is not in the list of devices associated with *kernel* or if *device* is NULL but there is more than one device associated with *kernel*.
- CL_INVALID_VALUE if *param_name* is not valid, or if size in bytes specified by *param_value_size* is < size of return type as described in the Kernel Object Sub-group Queries table and *param_value* is not NULL.
- CL_INVALID_VALUE if *param_name* is CL_KERNEL_MAX_SUB_GROUP_SIZE_FOR_NDRANGE_KHR and the size in bytes specified by *input_value_size* is not valid or if *input_value* is NULL.
- CL_INVALID_KERNEL if *kernel* is a not a valid kernel object.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

28.5. Additions to Chapter 6 of the OpenCL 2.0 C Specification

28.5.1. Additions to section 6.13.1 — Work Item Functions

Function	Description
uint **get_sub_group_size** ()	Returns the number of work items in the sub-group. This value is no more than the maximum sub-group size and is implementation-defined based on a combination of the compiled kernel and the dispatch dimensions. This will be a constant value for the lifetime of the sub-group.
uint **get_max_sub_group_size** ()	Returns the maximum size of a sub-group within the dispatch. This value will be invariant for a given set of dispatch dimensions and a kernel object compiled for a given device.
uint **get_num_sub_groups** ()	Returns the number of sub-groups that the current work-group is divided into. This number will be constant for the duration of a work-group's execution. If the kernel is executed with a non-uniform work-group size (i.e. the global_work_size values specified to **clEnqueueNDRangeKernel** are not evenly divisible by the local_work_size values for any dimension, calls to this built-in from some work-groups may return different values than calls to this built-in from other work-groups.
uint **get_enqueued_num_sub_groups** ()	Returns the same value as that returned by **get_num_sub_groups** if the kernel is executed with a uniform work-group size. If the kernel is executed with a non-uniform work-group size, returns the number of sub-groups in each of the work-groups that make up the uniform region of the global range.
uint **get_sub_group_id** ()	**get_sub_group_id** returns the sub-group ID which is a number from 0 .. **get_num_sub_groups**() - 1. For **clEnqueueTask**, this returns 0.

Function	Description
uint **get_sub_group_local_id** ()	Returns the unique work item ID within the current sub-group. The mapping from **get_local_id**(*dimindx*) to **get_sub_group_local_id** will be invariant for the lifetime of the work-group.

28.5.2. Additions to section 6.13.8 — Synchronization Functions

Function	Description
void **sub_group_barrier** (cl_mem_fence_flags *flags*) void **sub_group_barrier** (cl_mem_fence_flags *flags*, memory_scope *scope*)	All work items in a sub-group executing the kernel on a processor must execute this function before any are allowed to continue execution beyond the sub-group barrier. This function must be encountered by all work items in a sub-group executing the kernel. These rules apply to ND-ranges implemented with uniform and non-uniform work-groups. If **sub_group_barrier** is inside a conditional statement, then all work items within the sub-group must enter the conditional if any work item in the sub-group enters the conditional statement and executes the sub_group_barrier. If **sub_group_barrier** is inside a loop, all work items within the sub-group must execute the sub_group_barrier for each iteration of the loop before any are allowed to continue execution beyond the sub_group_barrier. The **sub_group_barrier** function also queues a memory fence (reads and writes) to ensure correct ordering of memory operations to local or global memory. The flags argument specifies the memory address space and can be set to a combination of the following values: CLK_LOCAL_MEM_FENCE - The **sub_group_barrier** function will either flush any variables stored in local memory or queue a memory fence to ensure correct ordering of memory operations to local memory. CLK_GLOBAL_MEM_FENCE — The **sub_group_barrier** function will queue a memory fence to ensure correct ordering of memory operations to global memory. This can be useful when work items, for example, write to buffer objects and then want to read the updated data from these buffer objects. CLK_IMAGE_MEM_FENCE — The **sub_group_barrier** function will queue a memory fence to ensure correct ordering of memory operations to image objects. This can be useful when work items, for example, write to image objects and then want to read the updated data from these image objects.

28.5.3. Additions to section 6.13.11 — Atomic Functions

Add the following new value to the enumerated type `memory_scope` defined in *section 6.13.11.4*.

```
memory_scope_sub_group
```

The `memory_scope_sub_group` specifies that the memory ordering constraints given by `memory_order` apply to work items in a sub-group. This memory scope can be used when performing atomic operations to global or local memory.

28.5.4. Add a new section 6.13.X — Sub-group Functions

The table below describes OpenCL C programming language built-in functions that operate on a sub-group level. These built-in functions must be encountered by all work items in the sub-group executing the kernel. For the functions below, the generic type name `gentype` may be the one of the supported built-in scalar data types `int`, `uint`, `long`, `ulong`, `float`, `double` (if double precision is supported), or `half` (if half precision is supported).

Function	Description
int **sub_group_all** (int *predicate*)	Evaluates *predicate* for all work items in the sub-group and returns a non-zero value if *predicate* evaluates to non-zero for all work items in the sub-group.
int **sub_group_any** (int *predicate*)	Evaluates *predicate* for all work items in the sub-group and returns a non-zero value if *predicate* evaluates to non-zero for any work items in the sub-group.
gentype **sub_group_broadcast** (gentype *x*, uint *sub_group_local_id*)	Broadcast the value of *x* for work item identified by *sub_group_local_id* (value returned by **get_sub_group_local_id**) to all work items in the sub-group. *sub_group_local_id* must be the same value for all work items in the sub-group.
gentype **sub_group_reduce_<op>** (gentype *x*)	Return result of reduction operation specified by **<op>** for all values of *x* specified by work items in a sub-group.
gentype **sub_group_scan_exclusive_<op>** (gentype *x*)	Do an exclusive scan operation specified by **<op>** of all values specified by work items in a sub-group. The scan results are returned for each work item. The scan order is defined by increasing sub-group local ID within the sub-group.

Function	Description
gentype **sub_group_scan_inclusive_<op>** (gentype *x*)	Do an inclusive scan operation specified by **<op>** of all values specified by work items in a sub-group. The scan results are returned for each work item. The scan order is defined by increasing sub-group local ID within the sub-group.

The <op> in **sub_group_reduce_<op>**, **sub_group_scan_inclusive_<op>** and **sub_group_scan_exclusive_<op>** defines the operator and can be **add**, **min** or **max**.

The exclusive scan operation takes a binary operator **op** with an identity I and *n* (where *n* is the size of the sub-group) elements [a_0, a_1, … a_{n-1}] and returns [I, a_0, (a_0 **op** a_1), … (a_0 **op** a_1 **op** … **op** a_{n-2})].

The inclusive scan operation takes a binary operator **op** with *n* (where *n* is the size of the sub-group) elements [a_0, a_1, … a_{n-1}] and returns [a_0, (a_0 **op** a_1), … (a_0 **op** a_1 **op** … **op** a_{n-1})].

If **op** = **add**, the identity I is 0. If **op** = **min**, the identity I is `INT_MAX`, `UINT_MAX`, `LONG_MAX`, `ULONG_MAX`, for `int`, `uint`, `long`, `ulong` types and is `+INF` for floating-point types. Similarly if **op** = max, the identity I is `INT_MIN`, 0, `LONG_MIN`, 0 and `-INF`.

The order of floating-point operations is not guaranteed for the **sub_group_reduce_<op>**, **sub_group_scan_inclusive_<op>** and **sub_group_scan_exclusive_<op>** built-in functions that operate on `half`, `float` and `double` data types. The order of these floating-point operations is also non-deterministic for a given sub-group.

28.5.5. Additions to section 6.13.16 — Pipe Functions

The OpenCL C programming language implements the following built-in pipe functions that operate at a sub-group level. These built-in functions must be encountered by all work items in a sub-group executing the kernel with the same argument values; otherwise the behavior is undefined. We use the generic type name `gentype` to indicate 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 can be used as the type for the arguments to the pipe functions listed in *table 6.29*.

Function	Description
reserve_id_t **sub_group_reserve_read_pipe** (read_only pipe gentype *pipe*, uint *num_packets*) reserve_id_t **sub_group_reserve_write_pipe** (write_only pipe gentype *pipe*, uint *num_packets*)	Reserve *num_packets* entries for reading from or writing to *pipe*. Returns a valid non-zero reservation ID if the reservation is successful and 0 otherwise. The reserved pipe entries are referred to by indices that go from 0 … *num_packets* - 1.

Function	Description
void **sub_group_commit_read_pipe** (read_only pipe gentype *pipe*, reserve_id_t *reserve_id*) void **sub_group_commit_write_pipe** (write_only pipe gentype *pipe*, reserve_id_t *reserve_id*)	Indicates that all reads and writes to *num_packets* associated with reservation *reserve_id* are completed.

Note: Reservations made by a sub-group are ordered in the pipe as they are ordered in the program. Reservations made by different sub-groups that belong to the same work-group can be ordered using sub-group synchronization. The order of sub-group based reservations that belong to different work groups is implementation-defined.

28.5.6. Additions to section 6.13.17.6 — Enqueuing Kernels (Kernel Query Functions)

Built-in Function	Description
uint **get_kernel_sub_group_count_for_ndrange** (const ndrange_t *ndrange*, void (^block)(void)); uint **get_kernel_sub_group_count_for_ndrange** (const ndrange_t *ndrange*, void (^block)(local void *, …));	Returns the number of sub-groups in each work-group of the dispatch (except for the last in cases where the global size does not divide cleanly into work-groups) given the combination of the passed ndrange and block. *block* specifies the block to be enqueued.
uint **get_kernel_max_sub_group_size_for_ndrange** (const ndrange_t *ndrange*, void (^block)(void)); uint **get_kernel_max_sub_group_size_for_ndrange** (const ndrange_t *ndrange*, void (^block)(local void *, …));	Returns the maximum sub-group size for a block.

Chapter 29. Mipmaps

This section describes OpenCL support for mipmaps.

There are two optional mipmap extensions. The **cl_khr_mipmap_image** extension adds the ability to create a mip-mapped image, enqueue commands to read/write/copy/map/unmap a region of a mipmapped image, and built-in functions that can be used to read a mip-mapped image in an OpenCL C program. The **cl_khr_mipmap_image_writes** extension adds built-in functions that can be used to write a mip-mapped image in an OpenCL C program. If the **cl_khr_mipmap_image_writes** extension is supported by the OpenCL device, the **cl_khr_mipmap_image** extension must also be supported.

29.1. General Information

29.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

29.2. Additions to Chapter 5 of the OpenCL 2.2 Specification

29.2.1. Additions to section 5.3 — Image Objects

A mip-mapped 1D image, 1D image array, 2D image, 2D image array or 3D image is created by specifying *num_mip_levels* to be a value greater than one in the *image_desc* passed to **clCreateImage**. The dimensions of a mip-mapped image can be a power of two or a non-power of two. Each successively smaller mipmap level is half the size of the previous level. If this half value is a fractional value, it is rounded down to the nearest integer.

Restrictions

The following restrictions apply when mip-mapped images are created with **clCreateImage**:

- `CL_MEM_USE_HOST_PTR` or `CL_MEM_COPY_HOST_PTR` cannot be specified if a mip-mapped image is created.
- The *host_ptr* argument to **clCreateImage** must be a `NULL` value.
- Mip-mapped images cannot be created for `CL_MEM_OBJECT_IMAGE1D_BUFFER` images, depth images or multi-sampled (i.e. msaa) images.

Calls to **clEnqueueReadImage**, **clEnqueueWriteImage** and **clEnqueueMapImage** can be used to read from or write to a specific mip-level of a mip-mapped image. If image argument is a 1D image, *origin*[1] specifies the mip-level to use. If image argument is a 1D image array, *origin*[2] specifies the mip-level to use. If image argument is a 2D image, *origin*[2] specifies the mip-level to use. If image argument is a 2D image array or a 3D image, *origin*[3] specifies the mip-level to use.

Calls to **clEnqueueCopyImage**, **clEnqueueCopyImageToBuffer** and **clEnqueueCopyBufferToImage** can also be used to copy from and to a specific mip-level of a mip-mapped image. If *src_image* argument is a 1D image, *src_origin*[1] specifies the mip-level to use. If *src_image* argument is a 1D image array, *src_origin*[2] specifies the mip-level to use. If *src_image* argument is a 2D image, *src_origin*[2] specifies the mip-level to use. If *src_image* argument is a 2D image array or a 3D image, *src_origin*[3] specifies the mip-level to use. If *dst_image* argument is a 1D image, *dst_origin*[1] specifies the mip-level to use. If *dst_image* argument is a 1D image array, *dst_origin*[2] specifies the mip-level to use. If *dst_image* argument is a 2D image, *dst_origin*[2] specifies the mip-level to use. If *dst_image* argument is a 2D image array or a 3D image, *dst_origin*[3] specifies the mip-level to use.

If the mip level specified is not a valid value, these functions return the error `CL_INVALID_MIP_LEVEL`.

Calls to **clEnqueueFillImage** can be used to write to a specific mip-level of a mip-mapped image. If image argument is a 1D image, origin[1] specifies the mip-level to use. If image argument is a 1D image array, origin[2] specifies the mip-level to use. If image argument is a 2D image, origin[2] specifies the mip-level to use. If image argument is a 2D image array or a 3D image, origin[3] specifies the mip-level to use.

29.2.2. Additions to section 5.7 — Sampler Objects

Add the following sampler properties *to table 5.14* that can be specified when a sampler object is created using **clCreateSamplerWithProperties**.

Sampler Property	Property Value	Default Value
`CL_SAMPLER_MIP_FILTER_MODE_KHR`	`cl_filter_mode`	`CL_FILTER_NEAREST`
`CL_SAMPLER_LOD_MIN_KHR`	`cl_float`	`0.0f`
`CL_SAMPLER_LOD_MAX_KHR`	`cl_float`	`MAXFLOAT`

Note: The sampler properties `CL_SAMPLER_MIP_FILTER_MODE_KHR`, `CL_SAMPLER_LOD_MIN_KHR` and `CL_SAMPLER_LOD_MAX_KHR` cannot be specified with any samplers initialized in the OpenCL program source. Only the default values for these properties will be used. To create a sampler with specific values for these properties, a sampler object must be created with **clCreateSamplerWithProperties** and passed as an argument to a kernel.

29.3. Additions to Chapter 6 of the OpenCL 2.0 Specification

29.3.1. Additions to section 6.13.14 – Image Read, Write and Query Functions

The image read and write functions described in *sections 6.13.14.2*, *6.13.14.3* and *6.13.14.4* read from and write to mip-level 0 if the image argument is a mip-mapped image.

The following new built-in functions are added to *section 6.13.14.2*.

Function	Description
``float4 read_imagef(read_only image2d_t image, sampler_t sampler, float2 coord, float lod) int4 read_imagei(read_only image2d_t image, sampler_t sampler, float2 coord, float lod) uint4 read_imageui(read_only image2d_t image, sampler_t sampler, float2 coord, float lod) float read_imagef(read_only image2d_depth_t image, sampler_t sampler, float2 coord, float lod)```	Use the coordinate *coord.xy* to do an element lookup in the mip-level specified by *lod* in the 2D image object specified by *image*.

Function	Description
`float4 read_imagef(read_only image2d_t image, sampler_t sampler, float2 coord, float2 gradient_x, float2 gradient_y)` `int4 read_imagei(read_only image2d_t image, sampler_t sampler, float2 coord, float2 gradient_x, float2 gradient_y)` `uint4 read_imageui(read_only image2d_t image, sampler_t sampler, float2 coord, float2 gradient_x, float2 gradient_y)` `float read_imagef(read_only image2d_depth_t image, sampler_t sampler, float2 coord, float2 gradient_x, float2 gradient_y)`	Use the gradients to compute the lod and coordinate *coord.xy* to do an element lookup in the mip-level specified by the computed lod in the 2D image object specified by *image*.
`float4 read_imagef(read_only image1d_t image, sampler_t sampler, float coord, float lod)` `int4 read_imagei(read_only image1d_t image, sampler_t sampler, float coord, float lod)` `uint4 read_imageui(read_only image1d_t image, sampler_t sampler, float coord, float lod)`	Use the coordinate *coord* to do an element lookup in the mip-level specified by *lod* in the 1D image object specified by *image*.

Function	Description
<pre>float4 read_imagef(read_only image1d_t image, sampler_t sampler, float coord, float gradient_x, float gradient_y) int4 read_imagei(read_only image1d_t image, sampler_t sampler, float coord, float gradient_x, float gradient_y) uint4 read_imageui(read_only image1d_t image, sampler_t sampler, float coord, float gradient_x, float gradient_y)</pre>	Use the gradients to compute the lod and coordinate *coord* to do an element lookup in the mip-level specified by the computed lod in the 1D image object specified by *image*.
<pre>float4 read_imagef(read_only image3d_t image, sampler_t sampler, float4 coord, float lod) int4 read_imagei(read_only image3d_t image, sampler_t sampler, float4 coord, float lod) uint4 read_imageui(read_only image3d_t image, sampler_t sampler, float4 coord, float lod)</pre>	Use the coordinate *coord.xyz* to do an element lookup in the mip-level specified by *lod* in the 3D image object specified by *image*.

Function	Description
`float4 read_imagef(` `    read_only image3d_t image,` `    sampler_t sampler,` `    float4 coord,` `    float4 gradient_x,` `    float4 gradient_y)` `int4 read_imagei(` `    read_only image3d_t image,` `    sampler_t sampler,` `    float4 coord,` `    float4 gradient_x,` `    float4 gradient_y)` `uint4 read_imageui(` `    read_only image3d_t image,` `    sampler_t sampler,` `    float4 coord,` `    float4 gradient_x,` `    float4 gradient_y)`	Use the gradients to compute the lod and coordinate *coord.xyz* to do an element lookup in the mip-level specified by the computed lod in the 3D image object specified by *image*.
`float4 read_imagef(` `    read_only image1d_array_t image,` `    sampler_t sampler,` `    float2 coord,` `    float lod)` `int4 read_imagei(` `    read_only image1d_array_t image,` `    sampler_t sampler,` `    float2 coord,` `    float lod)` `uint4 read_imageui(` `    read_only image1d_array_t image,` `    sampler_t sampler,` `    float2 coord,` `    float lod)`	Use the coordinate *coord.x* to do an element lookup in the 1D image identified by *coord.x* and mip-level specified by *lod* in the 1D image array specified by *image*.

Function	Description
<pre>float4 read_imagef(read_only image1d_array_t image, sampler_t sampler, float2 coord, float gradient_x, float gradient_y) int4 read_imagei(read_only image1d_array_t image, sampler_t sampler, float2 coord, float gradient_x, float gradient_y) uint4 read_imageui(read_only image1d_array_t image, sampler_t sampler, float2 coord, float gradient_x, float gradient_y)</pre>	Use the gradients to compute the lod and coordinate *coord.x* to do an element lookup in the mip-level specified by the computed lod in the 1D image array specified by *image*.
<pre>float4 read_imagef(read_only image2d_array_t image, sampler_t sampler, float4 coord, float lod) int4 read_imagei(read_only image2d_array_t image, sampler_t sampler, float4 coord, float lod) uint4 read_imageui(read_only image2d_array_t image, sampler_t sampler, float4 coord, float lod) float read_imagef(read_only image2d_array_depth_t image, sampler_t sampler, float4 coord, float lod)</pre>	Use the coordinate *coord.xy* to do an element lookup in the 2D image identified by *coord.z* and mip-level specified by *lod* in the 2D image array specified by *image*.

Function	Description
`float4 read_imagef(read_only image2d_array_t image, sampler_t sampler, float4 coord, float2 gradient_x, float2 gradient_y)` `int4 read_imagei(read_only image2d_array_t image, sampler_t sampler, float4 coord, float2 gradient_x, float2 gradient_y)` `uint4 read_imageui(read_only image2d_array_t image, sampler_t sampler, float4 coord, float2 gradient_x, float2 gradient_y)` `float read_imagef(read_only image2d_array_depth_t image, sampler_t sampler, float4 coord, float2 gradient_x, float2 gradient_y)`	Use the gradients to compute the lod coordinate and *coord.xy* to do an element lookup in the 2D image identified by *coord.z* and mip-level specified by the computed lod in the 2D image array specified by *image*.

`CL_SAMPLER_NORMALIZED_COORDS` must be `CL_TRUE` for built-in functions described in the table above that read from a mip-mapped image; otherwise the behavior is undefined. The value specified in the *lod* argument is clamped to the minimum of (actual number of mip-levels – 1) in the image or value specified for `CL_SAMPLER_LOD_MAX`.

The following new built-in functions are added to *section 6.13.14.4*.

Function	Description
`void write_imagef(` `    write_only image2d_t image,` `    int2 coord,` `    int lod,` `    float4 color)` `void write_imagei(` `    write_only image2d_t image,` `    int2 coord,` `    int lod,` `    int4 color)` `void write_imageui(` `    write_only image2d_t image,` `    int2 coord,` `    int lod,` `    uint4 color)` `void write_imagef(` `    write_only image2d_depth_t image,` `    int2 coord,` `    int lod,` `    float depth)`	Write *color* value to location specified by *coord.xy* in the mip-level specified by *lod* in the 2D image object specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. *coord.x* and *coord.y* are considered to be unnormalized coordinates and must be in the range 0 .. image width of mip-level specified by *lod* – 1, and 0 .. image height of mip-level specified by *lod* – 1. The behavior of **write_imagef**, **write_imagei** and **write_imageui** if (*x*, *y*) coordinate values are not in the range (0 .. image width of the mip-level specified by *lod* – 1, 0 .. image height of the mip-level specified by *lod* – 1) or *lod* value exceeds the (number of mip-levels in the image – 1) is undefined.
`void write_imagef(` `    write_only image1d_t image,` `    int coord,` `    int lod,` `    float4 color)` `void write_imagei(` `    write_only image1d_t image,` `    int coord,` `    int lod,` `    int4 color)` `void write_imageui(` `    write_only image1d_t image,` `    int coord,` `    int lod,` `    uint4 color)`	Write *color* value to location specified by *coord* in the mip-level specified by *lod* in the 1D image object specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. *coord* is considered to be unnormalized coordinates and must be in the range 0 .. image width of the mip-level specified by *lod* – 1. The behavior of **write_imagef**, **write_imagei** and **write_imageui** if coordinate value is not in the range (0 .. image width of the mip-level specified by *lod* – 1) or *lod* value exceeds the (number of mip-levels in the image – 1), is undefined.

Function	Description
<pre>void write_imagef(write_only image1d_array_t image, int2 coord, int lod, float4 color) void write_imagei(write_only image1d_array_t image, int2 coord, int lod, int4 color) void write_imageui(write_only image1d_array_t image, int2 coord, int lod, uint4 color)</pre>	Write *color* value to location specified by *coord.x* in the 1D image identified by *coord.y* and mip-level *lod* in the 1D image array specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. *coord.x* and *coord.y* are considered to be unnormalized coordinates and must be in the range 0 .. image width of the mip-level specified by *lod* – 1 and 0 .. image number of layers – 1. The behavior of **write_imagef**, **write_imagei** and **write_imageui** if (*x*, *y*) coordinate values are not in the range (0 .. image width of the mip-level specified by *lod* – 1, 0 .. image number of layers – 1), respectively or *lod* value exceeds the (number of mip-levels in the image – 1), is undefined.
<pre>void write_imagef(write_only image2d_array_t image, int4 coord, int lod, float4 color) void write_imagei(write_only image2d_array_t image, int4 coord, int lod, int4 color) void write_imageui(write_only image2d_array_t image, int4 coord, int lod, uint4 color) void write_imagef(write_only image2d_array_depth_t image, int4 coord, int lod, float depth)</pre>	Write *color* value to location specified by *coord.xy* in the 2D image identified by *coord.z* and mip-level *lod* in the 2D image array specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. *coord.x*, *coord.y* and *coord.z* are considered to be unnormalized coordinates and must be in the range 0 .. image width of the mip-level specified by *lod* – 1, 0 .. image height – 1 specified by *lod* – 1 and 0 .. image number of layers – 1. The behavior of **write_imagef**, **write_imagei** and **write_imageui** if (*x*, *y*, *z*) coordinate values are not in the range (0 .. image width of the mip-level specified by *lod* – 1, 0 .. image height of the mip-level specified by *lod* – 1, 0 .. image number of layers – 1), respectively or *lod* value exceeds the (number of mip-levels in the image – 1), is undefined.

Function	Description
<pre>void write_imagef(write_only image3d_t image, int4 coord, int lod, float4 color) void write_imagei(write_only image3d_t image, int4 coord, int lod, int4 color) void write_imageui(write_only image3d_t image, int4 coord, int lod, uint4 color)</pre>	Write color value to location specified by *coord.xyz* and mip-level *lod* in the 3D image object specified by *image*. Appropriate data format conversion to the specified image format is done before writing the color value. *coord.x*, *coord.y* and *coord.z* are considered to be unnormalized coordinates and must be in the range 0 .. image width – 1 specified by *lod* – 1, 0 .. image height – 1 specified by *lod* – 1 and 0 .. image depth – 1 specified by *lod* – 1. The behavior of **write_imagef**, **write_imagei** and **write_imageui** if (*x*, *y*, *z*) coordinate values are not in the range (0 .. image width of the mip-level specified by *lod* – 1, 0 .. image height of the mip-level specified by *lod* – 1, 0 .. image depth – 1), respectively or *lod* value exceeds the (number of mip-levels in the image – 1), is undefined.

The following new built-in functions are added to *section 6.13.14.5*.

Function	Description
<pre>int get_image_num_mip_levels(image1d_t image) int get_image_num_mip_levels(image2d_t image) int get_image_num_mip_levels(image3d_t image) int get_image_num_mip_levels(image1d_array_t image) int get_image_num_mip_levels(image2d_array_t image) int get_image_num_mip_levels(image2d_depth_t image) int get_image_num_mip_levels(image2d_array_depth_t image)</pre>	Return the number of mip-levels.

29.4. Additions to Creating OpenCL Memory Objects from OpenGL Objects

If both the **cl_khr_mipmap_image** and **cl_khr_gl_sharing** extensions are supported by the OpenCL device, the **cl_khr_gl_sharing** extension may also be used to create a mipmapped OpenCL image from a mipmapped OpenGL texture.

To create a mipmapped OpenCL image from a mipmapped OpenGL texture, pass a negative value as the *miplevel* argument to **clCreateFromGLTexture**. If *miplevel* is a negative value then an OpenCL mipmapped image object is created from a mipmapped OpenGL texture object, instead of an OpenCL image object for a specific miplevel of the OpenGL texture.

Note: For a detailed description of how the level of detail is computed, please refer to *section 3.9.7* of the OpenGL 3.0 specification.

Chapter 30. sRGB Image Writes

This section describes the **cl_khr_srgb_image_writes** extension.

This extension enables kernels to write to sRGB images using the **write_imagef** built-in function. The sRGB image formats that may be written to will be returned by **clGetSupportedImageFormats**.

When the image is an sRGB image, the **write_imagef** built-in function will perform the linear to sRGB conversion. Only the R, G, and B components are converted from linear to sRGB; the A component is written as-is.

30.1. General Information

30.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

Chapter 31. Priority Hints

This section describes the **cl_khr_priority_hints** extension. This extension adds priority hints for OpenCL, but does not specify the scheduling behavior or minimum guarantees. It is expected that the the user guides associated with each implementation which supports this extension will describe the scheduling behavior guarantees.

31.1. General Information

31.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

31.2. Host-side API modifications

The function **clCreateCommandQueueWithProperties** (Section 5.1) is extended to support a priority value as part of the *properties* argument.

The priority property applies to OpenCL command-queues that belong to the same OpenCL context.

The properties field accepts the `CL_QUEUE_PRIORITY_KHR` property, with a value of type `cl_queue_priority_khr`, which can be one of:

- `CL_QUEUE_PRIORITY_HIGH_KHR`
- `CL_QUEUE_PRIORITY_MED_KHR`
- `CL_QUEUE_PRIORITY_LOW_KHR`

If `CL_QUEUE_PRIORITY_KHR` is not specified then the default priority is `CL_QUEUE_PRIORITY_MED_KHR`.

To the error section for **clCreateCommandQueueWithProperties**, the following is added:

- `CL_INVALID_QUEUE_PROPERTIES` if the `CL_QUEUE_PRIORITY_KHR` property is specified and the queue is a `CL_QUEUE_ON_DEVICE`.

Chapter 32. Throttle Hints

This section describes the **cl_khr_throttle_hints** extension. This extension adds throttle hints for OpenCL, but does not specify the throttling behavior or minimum guarantees. It is expected that the user guide associated with each implementation which supports this extension will describe the throttling behavior guarantees.

Note that the throttle hint is orthogonal to functionality defined in **cl_khr_priority_hints** extension. For example, a task may have high priority (`CL_QUEUE_PRIORITY_HIGH_KHR`) but should at the same time be executed at an optimized throttle setting (`CL_QUEUE_THROTTLE_LOW_KHR`).

32.1. General Information

32.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

32.2. Host-side API modifications

The function **clCreateCommandQueueWithProperties** (Section 5.1) is extended to support a new `CL_QUEUE_THROTTLE_KHR` value as part of the *properties* argument.

The properties field accepts the following values:

- `CL_QUEUE_THROTTLE_HIGH_KHR` (full throttle, i.e., OK to consume more energy)
- `CL_QUEUE_THROTTLE_MED_KHR` (normal throttle)
- `CL_QUEUE_THROTTLE_LOW_KHR` (optimized/lowest energy consumption)

If `CL_QUEUE_THROTTLE_KHR` is not specified then the default priority is `CL_QUEUE_THROTTLE_MED_KHR`.

To the error section for **clCreateCommandQueueWithProperties**, the following is added:

- `CL_INVALID_QUEUE_PROPERTIES` if the `CL_QUEUE_THROTTLE_KHR` property is specified and the queue is a `CL_QUEUE_ON_DEVICE`.

Chapter 33. Named Barriers for Sub-groups

This section describes the **cl_khr_subgroup_named_barrier** extension. This extension adds barrier operations that cover subsets of an OpenCL work-group. Only the OpenCL API changes are described in this section. Please refer to the SPIR-V specification for information about using sub-groups named barriers in the SPIR-V intermediate representation, and to the OpenCL C++ specification for descriptions of the sub-group named barrier built-in functions in the OpenCL C++ kernel language.

33.1. General Information

33.1.1. Version History

Date	Version	Description
2020-04-21	1.0.0	First assigned version.

33.2. Changes to OpenCL specification

Add to *table 4.3*:

Table 52. List of supported param_names by **clGetDeviceInfo**

Device Info	Return Type	Description
`CL_DEVICE_MAX_NAMED_BARRIER_COUNT_KHR`	`cl_uint`	Maximum number of named barriers in a work-group for any given kernel-instance running on the device. The minimum value is 8.

Chapter 34. Extended Async Copies

This section describes the **cl_khr_extended_async_copies** extension. This extension augments built-in asynchronous copy functions to OpenCL C to support more patterns:

1. for async copy between 2D source and 2D destination.
2. for async copy between 3D source and 3D destination.

34.1. General Information

34.1.1. Version History

Date	Version	Description
2020-04-21	0.9.0	First assigned version (provisional).
2021-09-06	0.9.1	Elements-based proposal update.
2021-11-10	1.0.0	First non-provisional version.

34.2. Additions to Chapter 6 of the OpenCL C Specification

The following new built-in functions are added to the *Async Copies from Global to Local Memory, Local to Global Memory, and Prefetch* functions described in *section 6.12.10* and *section 6.13.10* of the OpenCL 1.2 and OpenCL 2.0 C specifications.

Note that **async_work_group_strided_copy** is a special case of **async_work_group_copy_2D2D**, namely one which copies a single column to a single line or vice versa. For example: `async_work_group_strided_copy(dst, src, num_gentypes, src_stride, event)` is equal to `async_work_group_copy_2D2D(dst, 0, src, 0, sizeof(gentype), 1, num_gentypes, src_stride, 1, event)`

The async copy built-in functions described in this section support arbitrary `gentype`-based buffers by casting pointers to `void*`.

These async copy built-in functions do not perform any implicit synchronization of source data such as using a **barrier** before performing the copy.

These async copy built-in functions are performed by all work-items in a work-group and must therefore be encountered by all work-items in a work-group executing the kernel with the same argument values; otherwise the results are undefined.

The *src_offset*, *dst_offset*, *src_total_line_length*, *dst_total_line_length*, *src_total_plane_area* and *dst_total_plane_area* function arguments are expressed in elements.

Both *src_total_line_length* and *dst_total_line_length* describe the number of elements between the beginning of the current line and the beginning of the next line.

Both *src_total_plane_area* and *dst_total_plane_area* describe the number of elements between the beginning of the current plane and the beginning of the next plane.

These async copy built-in functions return an event object that can be used by **wait_group_events** to wait for the async copy to finish. The *event* argument can also be used to associate the async copy with a previous async copy allowing an event to be shared by multiple async copies; otherwise *event* should be zero. If the *event* argument is non-zero, the event object supplied as the *event* argument will be returned.

<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td>

```
event_t async_work_group_copy_2D2D(
    __local void *dst,
    size_t dst_offset,
    const __global void *src,
    size_t src_offset,
    size_t num_bytes_per_element,
    size_t num_elements_per_line,
    size_t num_lines,
    size_t src_total_line_length,
    size_t dst_total_line_length,
    event_t event)

event_t async_work_group_copy_2D2D(
    __global void *dst,
    size_t dst_offset,
    const __local void *src,
    size_t src_offset,
    size_t num_bytes_per_element,
    size_t num_elements_per_line,
    size_t num_lines,
    size_t src_total_line_length,
    size_t dst_total_line_length,
    event_t event)
```

</td><td>Perform an async copy of (num_elements_per_line * num_lines) elements of size num_bytes_per_element from (src + (src_offset * num_bytes_per_element)) to (dst + (dst_offset * num_bytes_per_element)). All pointer arithmetic is performed with implicit casting to <code>char*</code> by the implementation. Each line contains num_elements_per_line elements of size num_bytes_per_element. After each line of transfer, the src address is incremented by src_total_line_length elements (i.e. src_total_line_length * num_bytes_per_element bytes), and the dst address is incremented by dst_total_line_length elements (i.e. dst_total_line_length * num_bytes_per_element bytes), for the next line of transfer.

The behavior of async_work_group_copy_2D2D is undefined if the source or destination addresses exceed the upper bounds of the address space during the copy.

The behavior of async_work_group_copy_2D2D is also undefined if the src_total_line_length or dst_total_line_length values are smaller than num_elements_per_line, i.e. overlapping of lines is undefined.</td></tr>
</table>

Function	Description
`event_t async_work_group_copy_3D3D(__local void *dst, size_t dst_offset, const __global void *src, size_t src_offset, size_t num_bytes_per_element, size_t num_elements_per_line, size_t num_lines, size_t num_planes, size_t src_total_line_length, size_t src_total_plane_area, size_t dst_total_line_length, size_t dst_total_plane_area, event_t event)`  `event_t async_work_group_copy_3D3D(__global void *dst, size_t dst_offset, const __local void *src, size_t src_offset, size_t num_bytes_per_element, size_t num_elements_per_line, size_t num_lines, size_t num_planes, size_t src_total_line_length, size_t src_total_plane_area, size_t dst_total_line_length, size_t dst_total_plane_area, event_t event)`	Perform an async copy of ((*num_elements_per_line* * *num_lines*) * *num_planes*) elements of size *num_bytes_per_element* from (*src* + (*src_offset* * *num_bytes_per_element*)) to (*dst* + (*dst_offset* * *num_bytes_per_element*)), arranged in *num_planes* planes. All pointer arithmetic is performed with implicit casting to `char*` by the implementation. Each plane contains *num_lines* lines. Each line contains *num_elements_per_line* elements. After each line of transfer, the *src* address is incremented by *src_total_line_length* elements (i.e. *src_total_line_length* * *num_bytes_per_element* bytes), and the *dst* address is incremented by *dst_total_line_length* elements (i.e. *dst_total_line_length* * *num_bytes_per_element* bytes), for the next line of transfer. The behavior of **async_work_group_copy_3D3D** is undefined if the source or destination addresses exceed the upper bounds of the address space during the copy. The behavior of **async_work_group_copy_3D3D** is also undefined if the *src_total_line_length* or *dst_total_line_length* values are smaller than *num_elements_per_line*, i.e. overlapping of lines is undefined. The behavior of **async_work_group_copy_3D3D** is also undefined if *src_total_plane_area* is smaller than (*num_lines* * *src_total_line_length*), or *dst_total_plane_area* is smaller than (*num_lines* * *dst_total_line_length*), i.e. overlapping of planes is undefined.

Chapter 35. Async Work-group Copy Fence

This section describes the **cl_khr_async_work_group_copy_fence** extension. The extension adds a new built-in function to OpenCL C to establish a memory synchronization ordering of asynchronous copies.

35.1. General Information

35.1.1. Version History

Date	Version	Description
2020-04-21	0.9.0	First assigned version (provisional).
2021-11-10	1.0.0	First non-provisional version.

35.2. Additions to Chapter 6 of the OpenCL C Specification

The following new built-in function is added to the *Async Copies from Global to Local Memory, Local to Global Memory, and Prefetch* functions described in *section 6.12.10* and *section 6.13.10* of the OpenCL 1.2 and OpenCL 2.0 C specifications:

<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td>

```
void async_work_group_copy_fence(
    cl_mem_fence_flags flags)
```

</td><td>
Orders async copies produced by the work-items of a work-group executing a kernel. Async copies preceding the async_work_group_copy_fence must complete their access to the designated memory or memories, including both reads-from and writes-to it, before async copies following the fence are allowed to start accessing these memories. In other words, every async copy preceding the async_work_group_copy_fence must happen-before every async copy following the fence, with respect to the designated memory or memories.

The flags argument specifies the memory address space and can be set to a combination of the following literal values:

<code>CLK_LOCAL_MEM_FENCE</code>
<code>CLK_GLOBAL_MEM_FENCE</code>

The async fence is performed by all work-items in a work-group and this built-in function must therefore be encountered by all work-items in a work-group executing the kernel with the same argument values; otherwise the results are undefined. This rule applies to ND-ranges implemented with uniform and non-uniform work-groups.
</td></tr>
</table>

Chapter 36. Unique Device Identifiers

This section describes the **cl_khr_device_uuid** extension.

This extension adds the ability to query a universally unique identifier (UUID) for an OpenCL driver and OpenCL device. The UUIDs returned by the query may be used to identify drivers and devices across processes or APIs.

36.1. General Information

36.1.1. Version History

Date	Version	Description
2020-08-27	1.0.0	First assigned version.

36.2. Additions to Chapter 4 of the OpenCL 3.0 API Specification

Add to Table 5 - OpenCL Device Queries:

Table 5. List of supported param_names by **clGetDeviceInfo**

Device Info	Return Type	Description
`CL_DEVICE_UUID_KHR`	`cl_uchar`[`CL_UUID_SIZE_KHR`]	Returns a universally unique identifier (UUID) for the device. Device UUIDs must be immutable for a given device across processes, driver APIs, driver versions, and system reboots.
`CL_DRIVER_UUID_KHR`	`cl_uchar`[`CL_UUID_SIZE_KHR`]	Returns a universally unique identifier (UUID) for the software driver for the device.
`CL_DEVICE_LUID_VALID_KHR`	`cl_bool`	Returns `CL_TRUE` if the device has a valid LUID and `CL_FALSE` otherwise.

Device Info	Return Type	Description
CL_DEVICE_LUID_KHR	cl_uchar[CL_LUID_SIZE_KHR]	Returns a locally unique identifier (LUID) for the device. It is not an error to query CL_DEVICE_LUID_KHR when CL_DEVICE_LUID_VALID_KHR returns CL_FALSE, but in this case the returned LUID value is undefined. When CL_DEVICE_LUID_VALID_KHR returns CL_TRUE, and the OpenCL device is running on the Windows operating system, the returned LUID value can be cast to an LUID object and must be equal to the locally unique identifier of an IDXGIAdapter1 object that corresponds to the OpenCL device.
CL_DEVICE_NODE_MASK_KHR	cl_uint	Returns a node mask for the device. It is not an error to query CL_DEVICE_NODE_MASK_KHR when CL_DEVICE_LUID_VALID_KHR returns CL_FALSE, but in this case the returned node mask is undefined. When CL_DEVICE_LUID_VALID_KHR returns CL_TRUE, the returned node mask must contain exactly one bit. If the OpenCL device is running on an operating system that supports the Direct3D 12 API and the OpenCL device corresponds to an individual device in a linked device adapter, the returned node mask identifies the Direct3D 12 node corresponding to the OpenCL device. Otherwise, the returned node mask must be 1.

While CL_DEVICE_UUID_KHR is specified to remain consistent across driver versions and system reboots, it is not intended to be usable as a serializable persistent identifier for a device. It may change when a device is physically added to, removed from, or moved to a different connector in a system while that system is powered down. Further, there is no reasonable way to verify with conformance testing that a given device retains the same UUID in a given system across all driver versions supported in that system. While implementations should make every effort to report consistent device UUIDs across driver versions, applications should avoid relying on the persistence of this value for uses other than identifying compatible devices for external object sharing purposes.

Chapter 37. Extended versioning

This extension introduces new platform and device queries that return detailed version information to applications. It makes it possible to return the exact revision of the specification or intermediate languages supported by an implementation. It also enables implementations to communicate a version number for each of the extensions they support and remove the requirement for applications to process strings to test for the presence of an extension or intermediate language or built-in kernel.

Extended versioning was promoted to a core feature in OpenCL 3.0, however note that the query for `CL_DEVICE_OPENCL_C_NUMERIC_VERSION_KHR` was replaced by the query for `CL_DEVICE_OPENCL_C_ALL_VERSIONS`.

37.1. General Information

37.1.1. Name Strings

`cl_khr_extended_versioning`

37.1.2. Contributors

Kévin Petit, Arm Ltd.
Ben Ashbaugh, Intel
Alastair Murray, Codeplay Software Ltd.
Einar Hov, Arm Ltd.

37.1.3. Version History

Date	Version	Description
2020-02-12	1.0.0	Initial version.

37.1.4. Dependencies

This extension is written against the OpenCL Specification Version 2.2, Revision 11.

This extension requires OpenCL 1.0.

37.2. New API Types

37.2.1. Version

This extension introduces a new scheme to encode detailed (major, minor, patch/revision) version information into a single 32-bit unsigned integer:

- The major version is using bits 31-22
- The minor version is using bits 21-12

- The patch version is using bits 11-0

This scheme enables two versions to be ordered using the standard C/C++ operators. Macros are provided to extract individual fields or compose a full version from the individual fields.

```
typedef cl_uint cl_version_khr;

#define CL_VERSION_MAJOR_BITS_KHR (10)
#define CL_VERSION_MINOR_BITS_KHR (10)
#define CL_VERSION_PATCH_BITS_KHR (12)

#define CL_VERSION_MAJOR_MASK_KHR ((1 << CL_VERSION_MAJOR_BITS_KHR) - 1)
#define CL_VERSION_MINOR_MASK_KHR ((1 << CL_VERSION_MINOR_BITS_KHR) - 1)
#define CL_VERSION_PATCH_MASK_KHR ((1 << CL_VERSION_PATCH_BITS_KHR) - 1)

#define CL_VERSION_MAJOR_KHR(version) \
        ((version) >> (CL_VERSION_MINOR_BITS_KHR + CL_VERSION_PATCH_BITS_KHR))
#define CL_VERSION_MINOR_KHR(version) \
        (((version) >> CL_VERSION_PATCH_BITS_KHR) & CL_VERSION_MINOR_MASK_KHR)
#define CL_VERSION_PATCH_KHR(version) ((version) & CL_VERSION_PATCH_MASK_KHR)

#define CL_MAKE_VERSION_KHR(major, minor, patch) \
    ((((major) & CL_VERSION_MAJOR_MASK_KHR) << (CL_VERSION_MINOR_BITS_KHR +
CL_VERSION_PATCH_BITS_KHR)) | \
     (((minor) & CL_VERSION_MINOR_MASK_KHR) << CL_VERSION_PATCH_BITS_KHR) | \
     ((patch) & CL_VERSION_PATCH_MASK_KHR))
```

37.2.2. Name and version

This extension adds a structure that can be used to describe a combination of a name alongside a version number:

```
#define CL_NAME_VERSION_MAX_NAME_SIZE_KHR 64

typedef struct _cl_name_version_khr {
    cl_version_khr version;
    char name[CL_NAME_VERSION_MAX_NAME_SIZE_KHR];
} cl_name_version_khr;
```

The `name` field is an array of `CL_NAME_VERSION_MAX_NAME_SIZE_KHR` bytes used as storage for a NUL-terminated string whose maximum length is therefore `CL_NAME_VERSION_MAX_NAME_SIZE_KHR - 1`.

37.3. New API Enums

Accepted value for the *param_name* parameter to **clGetPlatformInfo**:

```
CL_PLATFORM_NUMERIC_VERSION_KHR
```

```
CL_PLATFORM_EXTENSIONS_WITH_VERSION_KHR
```

Accepted value for the *param_name* parameter to **clGetDeviceInfo**:

```
CL_DEVICE_NUMERIC_VERSION_KHR
CL_DEVICE_OPENCL_C_NUMERIC_VERSION_KHR
CL_DEVICE_EXTENSIONS_WITH_VERSION_KHR
CL_DEVICE_ILS_WITH_VERSION_KHR
CL_DEVICE_BUILT_IN_KERNELS_WITH_VERSION_KHR
```

37.4. Modifications to the OpenCL API Specification

(Modify Section 4.1, Querying Platform Info)

(Add the following to Table 3, *List of supported param_names by clGetPlatformInfo*)

Platform Info	Return Type	Description
`CL_PLATFORM_NUMERIC_VERSION_KHR`	`cl_version_khr`	Returns detailed (major, minor, patch) numeric version information. The major and minor version numbers returned must match those returned via `CL_PLATFORM_VERSION`.
`CL_PLATFORM_EXTENSIONS_WITH_VERSION_KHR`	`cl_name_version_khr`[]	Returns an array of description (name and version) structures. The same extension name must not be reported more than once. The list of extensions reported must match the list reported via `CL_PLATFORM_EXTENSIONS`.

(Modify Section 4.2, Querying Devices)

(Add the following to Table 5, *List of supported param_names by clGetDeviceInfo*)

Device Info	Return Type	Description
`CL_DEVICE_NUMERIC_VERSION_KHR`	`cl_version_khr`	Returns detailed (major, minor, patch) numeric version information. The major and minor version numbers returned must match those returned via `CL_DEVICE_VERSION`.

Device Info	Return Type	Description
CL_DEVICE_OPENCL_C_NUMERIC_VERSION_KHR	cl_version_khr	Returns detailed (major, minor, patch) numeric version information. The major and minor version numbers returned must match those returned via CL_DEVICE_OPENCL_C_VERSION.
CL_DEVICE_EXTENSIONS_WITH_VERSION_KHR	cl_name_version_khr[]	Returns an array of description (name and version) structures. The same extension name must not be reported more than once. The list of extensions reported must match the list reported via CL_DEVICE_EXTENSIONS.
CL_DEVICE_ILS_WITH_VERSION_KHR	cl_name_version_khr[]	Returns an array of descriptions (name and version) for all supported Intermediate Languages. Intermediate Languages with the same name may be reported more than once but each name and major/minor version combination may only be reported once. The list of intermediate languages reported must match the list reported via CL_DEVICE_IL_VERSION.
CL_DEVICE_BUILT_IN_KERNELS_WITH_VERSION_KHR	cl_name_version_khr[]	Returns an array of descriptions for the built-in kernels supported by the device. Each built-in kernel may only be reported once. The list of reported kernels must match the list returned via CL_DEVICE_BUILT_IN_KERNELS.

37.5. Conformance tests

1. Each of the new queries described in this extension must be attempted and succeed.
2. It must be verified that the information returned by all queries that extend existing queries is consistent with the information returned by existing queries.
3. Some of the queries introduced by this extension impose uniqueness constraints on the list of returned values. It must be verified that these constraints are satisfied.

37.6. Issues

1. What compatibility policy should we define? e.g. a *revision* has to be backwards-compatible with previous ones

   RESOLVED: No general rules as that wouldn't be testable. Here's a recommended policy:

   - Patch version bump: only clarifications and small/obvious bugfixes.
   - Minor version bump: backwards-compatible changes only.
   - Major version bump: backwards compatibility may break.

2. Do we want versioning for built-in kernels as returned by `CL_DEVICE_BUILT_IN_KERNELS`?

   RESOLVED: No immediate use-case for versioning but being able to get a list of individual kernels without parsing a string is desirable. Adding `CL_DEVICE_BUILT_IN_KERNELS_WITH_VERSION_KHR`.

3. What is the behaviour of the queries that return an array of structures when there are no elements to return?

   RESOLVED: The query succeeds and the size returned is zero.

4. What value should be returned when version information is not available?

   RESOLVED: If a patch version is not available, it should be reported as 0. If no version information is available, 0.0.0 should be reported. These values have been chosen as they are guaranteed to be lower than or equal to any other version.

5. Should we add a query to report SPIR-V extended instruction sets?

   RESOLVED: It is unlikely that we will introduce many SPIR-V extended instruction sets without an accompanying API extension. Decided not to do this.

6. Should the queries for which the old-style query doesn't exist in a given OpenCL version be present (e.g. `CL_DEVICE_BUILT_IN_KERNELS_WITH_VERSION_KHR` prior to OpenCL 2.1 or without support for `cl_khr_il_program` or `CL_DEVICE_OPENCL_C_NUMERIC_VERSION_KHR` on OpenCL 1.0)?

   RESOLVED: All the queries are always present. `CL_DEVICE_BUILT_IN_KERNELS_WITH_VERSION_KHR` returns an empty set when Intermediate Languages are not supported. `CL_DEVICE_OPENCL_C_NUMERIC_VERSION_KHR` always returns 1.0 on an OpenCL 1.0 platform.

7. Is reporting multiple Intermediate Languages with the same name and major/minor versions but differing patch versions allowed?

   RESOLVED: No. This isn't aligned with the intended use for patch versions and makes it harder for implementations to guarantee consistency with the existing IL queries.

Chapter 38. Extended Sub-group Functions

38.1. Overview

This section describes a family of extensions that provide extended sub-group functionality. The extensions in this family are:

- `cl_khr_subgroup_extended_types`
- `cl_khr_subgroup_non_uniform_vote`
- `cl_khr_subgroup_ballot`
- `cl_khr_subgroup_non_uniform_arithmetic`
- `cl_khr_subgroup_shuffle`
- `cl_khr_subgroup_shuffle_relative`
- `cl_khr_subgroup_clustered_reduce`

The functionality added by these extensions includes:

- Additional data type support for sub-group broadcast, scan, and reduction functions;
- The ability to elect a single work item from a sub-group to perform a task;
- The ability to hold votes among work items in a sub-group;
- The ability to collect and operate on ballots from work items in the sub-group;
- The ability to use some sub-group functions, such as any, all, broadcasts, scans, and reductions within non-uniform flow control;
- Additional scan and reduction operators;
- Additional ways to exchange data among work items in a sub-group;
- Clustered reductions, that operate on a subset of work items in the sub-group.

This section describes changes to the OpenCL C Language for these extensions. There are no new API functions or enums added by these extensions.

38.2. General Information

38.2.1. Version History

For all of the extensions described in this section:

Date	Version	Description
2020-12-15	1.0.0	First assigned version.

38.3. Summary of New OpenCL C Functions

```
// These functions are available to devices supporting
// cl_khr_subgroup_extended_types:

// Note: Existing functions supporting additional data types.

gentype sub_group_broadcast( gentype value, uint index )

gentype sub_group_reduce_add( gentype value )
gentype sub_group_reduce_min( gentype value )
gentype sub_group_reduce_max( gentype value )

gentype sub_group_scan_inclusive_add( gentype value )
gentype sub_group_scan_inclusive_min( gentype value )
gentype sub_group_scan_inclusive_max( gentype value )

gentype sub_group_scan_exclusive_add( gentype value )
gentype sub_group_scan_exclusive_min( gentype value )
gentype sub_group_scan_exclusive_max( gentype value )

// These functions are available to devices supporting
// cl_khr_subgroup_non_uniform_vote:

int sub_group_elect()

int sub_group_non_uniform_all( int predicate )
int sub_group_non_uniform_any( int predicate )
int sub_group_non_uniform_all_equal( gentype value )

// These functions are available to devices supporting
// cl_khr_subgroup_ballot:

gentype sub_group_non_uniform_broadcast( gentype value, uint index )
gentype sub_group_broadcast_first( gentype value )

uint4 sub_group_ballot( int predicate )
int   sub_group_inverse_ballot( uint4 value )
int   sub_group_ballot_bit_extract( uint4 value, uint index )
uint  sub_group_ballot_bit_count( uint4 value )
uint  sub_group_ballot_inclusive_scan( uint4 value )
uint  sub_group_ballot_exclusive_scan( uint4 value )
uint  sub_group_ballot_find_lsb( uint4 value )
uint  sub_group_ballot_find_msb( uint4 value )

uint4 get_sub_group_eq_mask()
uint4 get_sub_group_ge_mask()
uint4 get_sub_group_gt_mask()
uint4 get_sub_group_le_mask()
uint4 get_sub_group_lt_mask()
```

```
// These functions are available to devices supporting
// cl_khr_subgroup_non_uniform_arithmetic:

gentype sub_group_non_uniform_reduce_add( gentype value )
gentype sub_group_non_uniform_reduce_mul( gentype value )
gentype sub_group_non_uniform_reduce_min( gentype value )
gentype sub_group_non_uniform_reduce_max( gentype value )
gentype sub_group_non_uniform_reduce_and( gentype value )
gentype sub_group_non_uniform_reduce_or( gentype value )
gentype sub_group_non_uniform_reduce_xor( gentype value )
int     sub_group_non_uniform_reduce_logical_and( int predicate )
int     sub_group_non_uniform_reduce_logical_or( int predicate )
int     sub_group_non_uniform_reduce_logical_xor( int predicate )

gentype sub_group_non_uniform_scan_inclusive_add( gentype value )
gentype sub_group_non_uniform_scan_inclusive_mul( gentype value )
gentype sub_group_non_uniform_scan_inclusive_min( gentype value )
gentype sub_group_non_uniform_scan_inclusive_max( gentype value )
gentype sub_group_non_uniform_scan_inclusive_and( gentype value )
gentype sub_group_non_uniform_scan_inclusive_or( gentype value )
gentype sub_group_non_uniform_scan_inclusive_xor( gentype value )
int     sub_group_non_uniform_scan_inclusive_logical_and( int predicate )
int     sub_group_non_uniform_scan_inclusive_logical_or( int predicate )
int     sub_group_non_uniform_scan_inclusive_logical_xor( int predicate )

gentype sub_group_non_uniform_scan_exclusive_add( gentype value )
gentype sub_group_non_uniform_scan_exclusive_mul( gentype value )
gentype sub_group_non_uniform_scan_exclusive_min( gentype value )
gentype sub_group_non_uniform_scan_exclusive_max( gentype value )
gentype sub_group_non_uniform_scan_exclusive_and( gentype value )
gentype sub_group_non_uniform_scan_exclusive_or( gentype value )
gentype sub_group_non_uniform_scan_exclusive_xor( gentype value )
int     sub_group_non_uniform_scan_exclusive_logical_and( int predicate )
int     sub_group_non_uniform_scan_exclusive_logical_or( int predicate )
int     sub_group_non_uniform_scan_exclusive_logical_xor( int predicate )

// These functions are available to devices supporting
// cl_khr_subgroup_shuffle:

gentype sub_group_shuffle( gentype value, uint index )
gentype sub_group_shuffle_xor( gentype value, uint mask )

// These functions are available to devices supporting
// cl_khr_subgroup_shuffle_relative:

gentype sub_group_shuffle_up( gentype value, uint delta )
gentype sub_group_shuffle_down( gentype value, uint delta )

// These functions are available to devices supporting
// cl_khr_subgroup_clustered_reduce:
```

```
gentype sub_group_clustered_reduce_add( gentype value, uint clustersize )
gentype sub_group_clustered_reduce_mul( gentype value, uint clustersize )
gentype sub_group_clustered_reduce_min( gentype value, uint clustersize )
gentype sub_group_clustered_reduce_max( gentype value, uint clustersize )
gentype sub_group_clustered_reduce_and( gentype value, uint clustersize )
gentype sub_group_clustered_reduce_or( gentype value, uint clustersize )
gentype sub_group_clustered_reduce_xor( gentype value, uint clustersize )
int     sub_group_clustered_reduce_logical_and( int predicate, uint clustersize )
int     sub_group_clustered_reduce_logical_or( int predicate, uint clustersize )
int     sub_group_clustered_reduce_logical_xor( int predicate, uint clustersize )
```

38.4. Extended Types

This section describes functionality added by `cl_khr_subgroup_extended_types`. This extension adds additional supported data types to the existing sub-group broadcast, scan, and reduction functions.

38.4.1. Modify the Existing Section Describing Sub-group Functions

Modify the first paragraph in this section that describes `gentype` type support for the sub-group `broadcast`, `scan`, and `reduction` functions to add scalar `char`, `uchar`, `short`, and `ushort` support, and to additionally add built-in vector type support for `broadcast` specifically. The functions in the table and their descriptions remain unchanged by this extension:

The table below describes OpenCL C programming language built-in functions that operate on a sub-group level. These built-in functions must be encountered by all work items in the sub-group executing the kernel. We use the generic type name `gentype` to indicate the built-in scalar data types `char`, `uchar`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `float`, `double` (if double precision is supported), or `half` (if half precision is supported).

For the `sub_group_broadcast` function, the generic type name `gentype` may additionally be one of the supported built-in vector data types `charn`, `ucharn`, `shortn`, `ushortn`, `intn`, `uintn`, `longn`, `ulongn`, `floatn`, `doublen` (if double precision is supported), or `halfn` (if half precision is supported).

38.5. Votes and Elections

This section describes functionality added by `cl_khr_subgroup_non_uniform_vote`. This extension adds the ability to elect a single work item from a sub-group to perform a task and to hold votes among work items in a sub-group.

38.5.1. Add a new Section 6.15.X - Sub-group Vote and Elect Built-in Functions

The table below describes the OpenCL C programming language built-in functions to elect a single work item in a sub-group to perform a task and to collectively vote to determine a boolean condition for the sub-group. These functions need not be encountered by all work items in a sub-group executing the kernel. For the functions below, the generic type name `gentype` may be the one of the supported built-in scalar data types `char`, `uchar`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `float`, double (if double precision is supported), or half (if half precision is supported).

Function	Description
`int sub_group_elect()`	Elects a single work item in the sub-group to perform a task. This function will return true (nonzero) for the active work item in the sub-group with the smallest sub-group local ID, and false (zero) for all other active work items in the sub-group.
`int sub_group_non_uniform_all( int predicate )`	Examines *predicate* for all active work items in the sub-group and returns a non-zero value if *predicate* is non-zero for all active work items in the sub-group and zero otherwise. Note: This behavior is the same as sub_group_all from cl_khr_subgroups and OpenCL 2.1, except this function need not be encountered by all work items in the sub-group executing the kernel.
`int sub_group_non_uniform_any( int predicate )`	Examines *predicate* for all active work items in the sub-group and returns a non-zero value if *predicate* is non-zero for any active work item in the sub-group and zero otherwise. Note: This behavior is the same as sub_group_any from cl_khr_subgroups and OpenCL 2.1, except this function need not be encountered by all work items in the sub-group executing the kernel.
`int sub_group_non_uniform_all_equal( gentype value )`	Examines *value* for all active work items in the sub-group and returns a non-zero value if *value* is equivalent for all active invocations in the sub-group and zero otherwise. Integer types use a bitwise test for equality. Floating-point types use an ordered floating-point test for equality.

38.6. Ballots

This section describes functionality added by cl_khr_subgroup_ballot. This extension adds the ability to collect and operate on ballots from work items in the sub-group.

38.6.1. Add a new Section 6.15.X - Sub-group Ballot Built-in Functions

The table below describes the OpenCL C programming language built-in functions to allow work items in a sub-group to collect and operate on ballots from work items in the sub-group. These functions need not be encountered by all work items in a sub-group executing the kernel.

For the `sub_group_non_uniform_broadcast` and `sub_group_broadcast_first` functions, the generic type name `gentype` may be one of the supported built-in scalar data types `char`, `uchar`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `float`, `double` (if double precision is supported), or `half` (if half precision is supported).

For the `sub_group_non_uniform_broadcast` function, the generic type name `gentype` may additionally be one of the supported built-in vector data types `char`*n*, `uchar`*n*, `short`*n*, `ushort`*n*, `int`*n*, `uint`*n*, `long`*n*, `ulong`*n*, `float`*n*, `double`*n* (if double precision is supported), or `half`*n* (if half precision is supported).

Function	Description
`gentype sub_group_non_uniform_broadcast( gentype value, uint index )`	Returns *value* for the work item with sub-group local ID equal to *index*. Behavior is undefined when the value of *index* is not equivalent for all active work items in the sub-group. The return value is undefined if the work item with sub-group local ID equal to *index* is inactive or if *index* is greater than or equal to the size of the sub-group.
`gentype sub_group_broadcast_first( gentype value )`	Returns *value* for the work item with the smallest sub-group local ID among active work items in the sub-group.
`uint4 sub_group_ballot( int predicate )`	Returns a bitfield combining the *predicate* values from all work items in the sub-group. Bit zero of the first vector component represents the sub-group local ID zero, with higher-order bits and subsequent vector components representing, in order, increasing sub-group local IDs. The representative bit in the bitfield is set if the work item is active and the *predicate* is non-zero, and is unset otherwise.

Function	Description
`int sub_group_inverse_ballot( uint4 value )`	Returns the predicate value for this work item in the sub-group from the bitfield *value* representing predicate values from all work items in the sub-group. The predicate return value will be non-zero if the bit in the bitfield *value* for this work item is set, and zero otherwise. Behavior is undefined when *value* is not equivalent for all active work items in the sub-group. This is a specialized function that may perform better than the equivalent `sub_group_ballot_bit_extract` on some implementations.
`int sub_group_ballot_bit_extract( uint4 value, uint index )`	Returns the predicate value for the work item with sub-group local ID equal to *index* from the bitfield *value* representing predicate values from all work items in the sub-group. The predicate return value will be non-zero if the bit in the bitfield *value* for the work item with sub-group local ID equal to *index* is set, and zero otherwise. The predicate return value is undefined if the work item with sub-group local ID equal to *index* is greater than or equal to the size of the sub-group.
`uint sub_group_ballot_bit_count( uint4 value )`	Returns the number of bits that are set in the bitfield *value*, only considering the bits in *value* that represent predicate values corresponding to sub-group local IDs less than the maximum sub-group size within the dispatch (as returned by `get_max_sub_group_size`).
`uint sub_group_ballot_inclusive_scan( uint4 value )`	Returns the number of bits that are set in the bitfield *value*, only considering the bits in *value* representing work items with a sub-group local ID less than or equal to this work item's sub-group local ID.
`uint sub_group_ballot_exclusive_scan( uint4 value )`	Returns the number of bits that are set in the bitfield *value*, only considering the bits in *value* representing work items with a sub-group local ID less than this work item's sub-group local ID.

Function	Description
`uint sub_group_ballot_find_lsb( uint4 value )`	Returns the smallest sub-group local ID with a bit set in the bitfield *value*, only considering the bits in *value* that represent predicate values corresponding to sub-group local IDs less than the maximum sub-group size within the dispatch (as returned by `get_max_sub_group_size`). If no bits representing predicate values from all work items in the sub-group are set in the bitfield *value* then the return value is undefined.
`uint sub_group_ballot_find_msb( uint4 value )`	Returns the largest sub-group local ID with a bit set in the bitfield *value*, only considering the bits in *value* that represent predicate values corresponding to sub-group local IDs less than the maximum sub-group size within the dispatch (as returned by `get_max_sub_group_size`). If no bits representing predicate values from all work items in the sub-group are set in the bitfield *value* then the return value is undefined.
`uint4 get_sub_group_eq_mask()`	Generates a bitmask where the bit is set in the bitmask if the bit index equals the sub-group local ID and unset otherwise. Bit zero of the first vector component represents the sub-group local ID zero, with higher-order bits and subsequent vector components representing, in order, increasing sub-group local IDs.
`uint4 get_sub_group_ge_mask()`	Generates a bitmask where the bit is set in the bitmask if the bit index is greater than or equal to the sub-group local ID and less than the maximum sub-group size, and unset otherwise. Bit zero of the first vector component represents the sub-group local ID zero, with higher-order bits and subsequent vector components representing, in order, increasing sub-group local IDs.
`uint4 get_sub_group_gt_mask()`	Generates a bitmask where the bit is set in the bitmask if the bit index is greater than the sub-group local ID and less than the maximum sub-group size, and unset otherwise. Bit zero of the first vector component represents the sub-group local ID zero, with higher-order bits and subsequent vector components representing, in order, increasing sub-group local IDs.

Function	Description
`uint4 get_sub_group_le_mask()`	Generates a bitmask where the bit is set in the bitmask if the bit index is less than or equal to the sub-group local ID and unset otherwise. Bit zero of the first vector component represents the sub-group local ID zero, with higher-order bits and subsequent vector components representing, in order, increasing sub-group local IDs.
`uint4 get_sub_group_lt_mask()`	Generates a bitmask where the bit is set in the bitmask if the bit index is less than the sub-group local ID and unset otherwise. Bit zero of the first vector component represents the sub-group local ID zero, with higher-order bits and subsequent vector components representing, in order, increasing sub-group local IDs.

38.7. Non-Uniform Arithmetic

This section describes functionality added by `cl_khr_subgroup_non_uniform_arithmetic`. This extension adds the ability to use some sub-group functions within non-uniform flow control, including additional scan and reduction operators.

38.7.1. Add a new Section 6.15.X - Non Uniform Sub-group Scan and Reduction Built-in Functions

38.7.1.1. Arithmetic Operations

The table below describes the OpenCL C programming language built-in functions that perform simple arithmetic operations across work items in a sub-group. These functions need not be encountered by all work items in a sub-group executing the kernel. For the functions below, the generic type name `gentype` may be one of the supported built-in scalar data types `char`, `uchar`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `float`, `double` (if double precision is supported), or `half` (if half precision is supported).

Function	Description
`gentype sub_group_non_uniform_reduce_add( gentype value )` `gentype sub_group_non_uniform_reduce_min( gentype value )` `gentype sub_group_non_uniform_reduce_max( gentype value )` `gentype sub_group_non_uniform_reduce_mul( gentype value )`	Returns the summation, multiplication, minimum, or maximum of *value* for all active work items in the sub-group. Note: This behavior is the same as the **add**, **min**, and **max** reduction built-in functions from `cl_khr_subgroups` and OpenCL 2.1, except these functions support additional types and need not be encountered by all work items in the sub-group executing the kernel.
`gentype sub_group_non_uniform_scan_inclusive_add( gentype value )` `gentype sub_group_non_uniform_scan_inclusive_min( gentype value )` `gentype sub_group_non_uniform_scan_inclusive_max( gentype value )` `gentype sub_group_non_uniform_scan_inclusive_mul( gentype value )`	Returns the result of an inclusive scan operation, which is the summation, multiplication, minimum, or maximum of *value* for all active work items in the sub-group with a sub-group local ID less than or equal to this work item's sub-group local ID. Note: This behavior is the same as the **add**, **min**, and **max** inclusive scan built-in functions from `cl_khr_subgroups` and OpenCL 2.1, except these functions support additional types and need not be encountered by all work items in the sub-group executing the kernel.

Function	Description
``gentype sub_group_non_uniform_scan_exclusive_add(gentype value) gentype sub_group_non_uniform_scan_exclusive_min(gentype value) gentype sub_group_non_uniform_scan_exclusive_max(gentype value) gentype sub_group_non_uniform_scan_exclusive_mul(gentype value)```	Returns the result of an exclusive scan operation, which is the summation, multiplication, minimum, or maximum of *value* for all active work items in the sub-group with a sub-group local ID less than this work item's sub-group local ID. If there is no active work item in the sub-group with a sub-group local ID less than this work item's sub-group local ID then an identity value `I` is returned. For **add**, the identity value is `0`. For **min**, the identity value is the largest representable value for integer types, or `+INF` for floating-point types. For **max**, the identity value is the minimum representable value for integer types, or `-INF` for floating-point types. For **mul**, the identity value is `1`. Note: This behavior is the same as the **add**, **min**, and **max** exclusive scan built-in functions from `cl_khr_subgroups` and OpenCL 2.1, except these functions support additional types and need not be encountered by all work items in the sub-group executing the kernel.

Note: The order of floating-point operations is not guaranteed for the sub-group scan and reduction built-in functions that operate on floating-point types, and the order of operations may additionally be non-deterministic for a given sub-group.

38.7.1.2. Bitwise Operations

The table below describes the OpenCL C programming language built-in functions that perform simple bitwise integer operations across work items in a sub-group. These functions need not be encountered by all work items in a sub-group executing the kernel. For the functions below, the generic type name `gentype` may be one of the supported built-in scalar data types `char`, `uchar`, `short`, `ushort`, `int`, `uint`, `long`, and `ulong`.

Function	Description
`gentype sub_group_non_uniform_reduce_and( gentype value )` `gentype sub_group_non_uniform_reduce_or( gentype value )` `gentype sub_group_non_uniform_reduce_xor( gentype value )`	Returns the bitwise **and**, **or**, or **xor** of *value* for all active work items in the sub-group.
`gentype sub_group_non_uniform_scan_inclusive_and( gentype value )` `gentype sub_group_non_uniform_scan_inclusive_or( gentype value )` `gentype sub_group_non_uniform_scan_inclusive_xor( gentype value )`	Returns the result of an inclusive scan operation, which is the bitwise **and**, **or**, or **xor** of *value* for all active work items in the sub-group with a sub-group local ID less than or equal to this work item's sub-group local ID.
`gentype sub_group_non_uniform_scan_exclusive_and( gentype value )` `gentype sub_group_non_uniform_scan_exclusive_or( gentype value )` `gentype sub_group_non_uniform_scan_exclusive_xor( gentype value )`	Returns the result of an exclusive scan operation, which is the bitwise **and**, **or**, or **xor** of *value* for all active work items in the sub-group with a sub-group local ID less than this work item's sub-group local ID. If there is no active work item in the sub-group with a sub-group local ID less than this work item's sub-group local ID then an identity value `I` is returned. For **and**, the identity value is `~0` (all bits set). For **or** and **xor**, the identity value is `0`.

38.7.1.3. Logical Operations

The table below describes the OpenCL C programming language built-in functions that perform simple logical operations across work items in a sub-group. These functions need not be encountered by all work items in a sub-group executing the kernel. For these functions, a non-zero *predicate* argument or return value is logically `true` and a zero *predicate* argument or return value is logically `false`.

Function	Description
`int sub_group_non_uniform_reduce_logical_and( int predicate )` `int sub_group_non_uniform_reduce_logical_or( int predicate )` `int sub_group_non_uniform_reduce_logical_xor( int predicate )`	Returns the logical **and**, **or**, or **xor** of *predicate* for all active work items in the sub-group.

Function	Description
`int sub_group_non_uniform_scan_inclusive_logical_and( int predicate )` `int sub_group_non_uniform_scan_inclusive_logical_or( int predicate )` `int sub_group_non_uniform_scan_inclusive_logical_xor( int predicate )`	Returns the result of an inclusive scan operation, which is the logical **and**, **or**, or **xor** of *predicate* for all active work items in the sub-group with a sub-group local ID less than or equal to this work item's sub-group local ID.
`int sub_group_non_uniform_scan_exclusive_logical_and( int predicate )` `int sub_group_non_uniform_scan_exclusive_logical_or( int predicate )` `int sub_group_non_uniform_scan_exclusive_logical_xor( int predicate )`	Returns the result of an exclusive scan operation, which is the logical **and**, **or**, or **xor** of *predicate* for all active work items in the sub-group with a sub-group local ID less than this work item's sub-group local ID. If there is no active work item in the sub-group with a sub-group local ID less than this work item's sub-group local ID then an identity value `I` is returned. For **and**, the identity value is `true` (non-zero). For **or** and **xor**, the identity value is `false` (zero).

38.8. General Purpose Shuffles

This section describes functionality added by `cl_khr_subgroup_shuffle`. This extension adds additional ways to exchange data among work items in a sub-group.

38.8.1. Add a new Section 6.15.X - Sub-group Shuffle Built-in Functions

The table below describes the OpenCL C programming language built-in functions that allow work items in a sub-group to exchange data. These functions need not be encountered by all work items in a sub-group executing the kernel. For the functions below, the generic type name `gentype` may be one of the supported built-in scalar data types `char`, `uchar`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `float`, `double` (if double precision is supported), or `half` (if half precision is supported).

Function	Description
`gentype sub_group_shuffle( gentype value, uint index )`	Returns *value* for the work item with sub-group local ID equal to *index*. The shuffle *index* need not be the same for all work items in the sub-group. The return value is undefined if the work item with sub-group local ID equal to *index* is inactive or if *index* is greater than or equal to the size of the sub-group.
`gentype sub_group_shuffle_xor( gentype value, uint mask )`	Returns *value* for the work item with sub-group local ID equal to this work item's sub-group local ID xor'd with *mask*. The shuffle *mask* need not be the same for all work items in the sub-group. The return value is undefined if the work item with sub-group local ID equal to the calculated index is inactive or if the calculated index is greater than or equal to the size of the sub-group. This is a specialized function that may perform better than the equivalent `sub_group_shuffle` on some implementations.

38.9. Relative Shuffles

This section describes functionality added by `cl_khr_subgroup_shuffle_relative`. This extension adds specialized ways to exchange data among work items in a sub-group that may perform better on some implementations.

38.9.1. Add a new Section 6.15.X - Sub-group Relative Shuffle Built-in Functions

The table below describes specialized OpenCL C programming language built-in functions that allow work items in a sub-group to exchange data. These functions need not be encountered by all work items in a sub-group executing the kernel. For the functions below, the generic type name `gentype` may be one of the supported built-in scalar data types `char`, `uchar`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `float`, `double` (if double precision is supported), or `half` (if half precision is supported).

Function	Description
`gentype sub_group_shuffle_up( gentype value, uint delta )`	Returns *value* for the work item with sub-group local ID equal to this work item's sub-group local ID minus *delta*. The shuffle *delta* need not be the same for all work items in the sub-group. The return value is undefined if the work item with sub-group local ID equal to the calculated index is inactive, or *delta* is greater than this work item's sub-group local ID. This is a specialized function that may perform better than the equivalent `sub_group_shuffle` on some implementations.
`gentype sub_group_shuffle_down( gentype value, uint delta )`	Returns *value* for the work item with sub-group local ID equal to this work item's sub-group local ID plus *delta*. The shuffle *delta* need not be the same for all work items in the sub-group. The return value is undefined if the work item with sub-group local ID equal to the calculated index is inactive, or this work item's sub-group local ID plus *delta* is greater than or equal to the size of the sub-group. This is a specialized function that may perform better than the equivalent `sub_group_shuffle` on some implementations.

38.10. Clustered Reductions

This section describes functionality added by `cl_khr_subgroup_clustered_reduce`. This extension adds support for clustered reductions that operate on a subset of work items in the sub-group.

38.10.1. Add a new Section 6.15.X - Sub-group Clustered Reduction Built-in Functions

This section describes arithmetic operations that are performed on a subset of work items in a sub-group, referred to as a cluster. A cluster is described by a specified cluster size. Work items in a sub-group are assigned to clusters such that for cluster size *n*, the *n* work items in the sub-group with the smallest sub-group local IDs are assigned to the first cluster, then the *n* remaining work items with the smallest sub-group local IDs are assigned to the next cluster, and so on. Behavior is undefined if the specified cluster size is not an integer constant expression, is not a power-of-two, or is greater than the maximum size of a sub-group within the dispatch.

38.10.1.1. Arithmetic Operations

The table below describes the OpenCL C programming language built-in functions that perform simple arithmetic operations on a cluster of work items in a sub-group. These functions need not be encountered by all work items in a sub-group executing the kernel. For the functions below, the generic type name `gentype` may be one of the supported built-in scalar data types `char`, `uchar`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `float`, `double` (if double precision is supported), or `half` (if half precision is supported).

Function	Description
`gentype sub_group_clustered_reduce_add( gentype value, uint clustersize )` `gentype sub_group_clustered_reduce_mul( gentype value, uint clustersize )` `gentype sub_group_clustered_reduce_min( gentype value, uint clustersize )` `gentype sub_group_clustered_reduce_max( gentype value, uint clustersize )`	Returns the summation, multiplication, minimum, or maximum of *value* for all active work items in the sub-group within a cluster of the specified *clustersize*.

Note: The order of floating-point operations is not guaranteed for the sub-group clustered reduction built-in functions that operate on floating-point types, and the order of operations may additionally be non-deterministic for a given sub-group.

38.10.1.2. Bitwise Operations

The table below describes the OpenCL C programming language built-in functions to perform simple bitwise integer operations across a cluster of work items in a sub-group. These functions need not be encountered by all work items in a sub-group executing the kernel. For the functions below, the generic type name `gentype` may be the one of the supported built-in scalar data types `char`, `uchar`, `short`, `ushort`, `int`, `uint`, `long`, or `ulong`.

Function	Description
`gentype sub_group_clustered_reduce_and( gentype value, uint clustersize )` `gentype sub_group_clustered_reduce_or( gentype value, uint clustersize )` `gentype sub_group_clustered_reduce_xor( gentype value, uint clustersize )`	Returns the bitwise **and**, **or**, or **xor** of *value* for all active work items in the sub-group within a cluster of the specified *clustersize*.

38.10.1.3. Logical Operations

The table below describes the OpenCL C programming language built-in functions to perform simple logical operations across a cluster of work items in a sub-group. These functions need not be encountered by all work items in a sub-group executing the kernel. For these functions, a non-zero *predicate* argument or return value is logically `true` and a zero *predicate* argument or return value is logically `false`.

Function	Description
`int sub_group_clustered_reduce_logical_and( int predicate, uint clustersize )` `int sub_group_clustered_reduce_logical_or( int predicate, uint clustersize )` `int sub_group_clustered_reduce_logical_xor( int predicate, uint clustersize )`	Returns the logical **and**, **or**, or **xor** of *predicate* for all active work items in the sub-group within a cluster of the specified *clustersize*.

38.11. Function Mapping and Capabilities

This section describes a possible mapping between OpenCL built-in functions and SPIR-V instructions and required SPIR-V capabilities.

This section is informational and non-normative.

OpenCL C Function	SPIR-V BuiltIn or Instruction	Enabling SPIR-V Capability
For OpenCL 2.1 or `cl_khr_subgroups`:		
`get_sub_group_size`	**SubgroupSize**	**Kernel**
`get_max_sub_group_size`	**SubgroupMaxSize**	**Kernel**
`get_num_sub_groups`	**NumSubgroups**	**Kernel**
`get_enqueued_num_sub_groups`	**NumEnqueuedSubgroups**	**Kernel**
`get_sub_group_id`	**SubgroupId**	**Kernel**
`get_sub_group_local_id`	**SubgroupLocalInvocationId**	**Kernel**
`sub_group_barrier`	**OpControlBarrier**	None Needed
`sub_group_all`	**OpGroupAll**	**Groups**
`sub_group_any`	**OpGroupAny**	**Groups**
`sub_group_broadcast`	**OpGroupBroadcast**	**Groups**
`sub_group_reduce_add`	**OpGroupIAdd**, **OpGroupFAdd**	**Groups**
`sub_group_reduce_min`	**OpGroupSMin**, **OpGroupUMin**, **OpGroupFMin**	**Groups**
`sub_group_reduce_max`	**OpGroupSMax**, **OpGroupUMax**, **OpGroupFMax**	**Groups**
`sub_group_scan_exclusive_add`	**OpGroupIAdd**, **OpGroupFAdd**	**Groups**
`sub_group_scan_exclusive_min`	**OpGroupSMin**, **OpGroupUMin**, **OpGroupFMin**	**Groups**
`sub_group_scan_exclusive_max`	**OpGroupSMax**, **OpGroupUMax**, **OpGroupFMax**	**Groups**

OpenCL C Function	SPIR-V BuiltIn or Instruction	Enabling SPIR-V Capability
`sub_group_scan_inclusive_add`	**OpGroupIAdd, OpGroupFAdd**	**Groups**
`sub_group_scan_inclusive_min`	**OpGroupSMin, OpGroupUMin, OpGroupFMin**	**Groups**
`sub_group_scan_inclusive_max`	**OpGroupSMax, OpGroupUMax, OpGroupFMax**	**Groups**
`sub_group_reserve_read_pipe`	**OpGroupReserveReadPipePackets**	**Pipes**
`sub_group_reserve_write_pipe`	**OpGroupReserveReadWritePackets**	**Pipes**
`sub_group_commit_read_pipe`	**OpGroupCommitReadPipe**	**Pipes**
`sub_group_commit_write_pipe`	**OpGroupCommitWritePipe**	**Pipes**
`get_kernel_sub_group_count_for_ndrange`	**OpGetKernelNDrangeSubGroupCount**	**DeviceEnqueue**
`get_kernel_max_sub_group_size_for_ndrange`	**OpGetKernelNDrangeMaxSubGroupSize**	**DeviceEnqueue**
For `cl_khr_subgroup_extended_types`: Note: This extension adds new types to uniform sub-group operations.		
`sub_group_broadcast`	**OpGroupBroadcast**	**Groups**
`sub_group_reduce_add`	**OpGroupIAdd, OpGroupFAdd**	**Groups**
`sub_group_reduce_min`	**OpGroupSMin, OpGroupUMin, OpGroupFMin**	**Groups**
`sub_group_reduce_max`	**OpGroupSMax, OpGroupUMax, OpGroupFMax**	**Groups**
`sub_group_scan_exclusive_add`	**OpGroupIAdd, OpGroupFAdd**	**Groups**
`sub_group_scan_exclusive_min`	**OpGroupSMin, OpGroupUMin, OpGroupFMin**	**Groups**
`sub_group_scan_exclusive_max`	**OpGroupSMax, OpGroupUMax, OpGroupFMax**	**Groups**
`sub_group_scan_inclusive_add`	**OpGroupIAdd, OpGroupFAdd**	**Groups**
`sub_group_scan_inclusive_min`	**OpGroupSMin, OpGroupUMin, OpGroupFMin**	**Groups**
`sub_group_scan_inclusive_max`	**OpGroupSMax, OpGroupUMax, OpGroupFMax**	**Groups**
For `cl_khr_subgroup_non_uniform_vote`:		

OpenCL C Function	SPIR-V BuiltIn or Instruction	Enabling SPIR-V Capability
`sub_group_elect`	**OpGroupNonUniformElect**	**GroupNonUniform**
`sub_group_non_uniform_all`	**OpGroupNonUniformAll**	**GroupNonUniformVote**
`sub_group_non_uniform_any`	**OpGroupNonUniformAny**	**GroupNonUniformVote**
`sub_group_non_uniform_all_equal`	**OpGroupNonUniformAllEqual**	**GroupNonUniformVote**
For `cl_khr_subgroup_ballot`:		
`sub_group_non_uniform_broadcast`	**OpGroupNonUniformBroadcast**	**GroupNonUniformBallot**
`sub_group_broadcast_first`	**OpGroupNonUniformBroadcastFirst**	**GroupNonUniformBallot**
`sub_group_ballot`	**OpGroupNonUniformBallot**	**GroupNonUniformBallot**
`sub_group_inverse_ballot`	**OpGroupNonUniformInverseBallot**	**GroupNonUniformBallot**
`sub_group_ballot_bit_extract`	**OpGroupNonUniformBallotBitExtract**	**GroupNonUniformBallot**
`sub_group_ballot_bit_count`	**OpGroupNonUniformBallotBitCount**	**GroupNonUniformBallot**
`sub_group_ballot_inclusive_scan`	**OpGroupNonUniformBallotBitCount**	**GroupNonUniformBallot**
`sub_group_ballot_exclusive_scan`	**OpGroupNonUniformBallotBitCount**	**GroupNonUniformBallot**
`sub_group_ballot_find_lsb`	**OpGroupNonUniformBallotFindLSB**	**GroupNonUniformBallot**
`sub_group_ballot_find_msb`	**OpGroupNonUniformBallotFindMSB**	**GroupNonUniformBallot**
`get_sub_group_eq_mask`	**SubgroupEqMask**	**GroupNonUniformBallot**
`get_sub_group_ge_mask`	**SubgroupGeMask**	**GroupNonUniformBallot**
`get_sub_group_gt_mask`	**SubgroupGtMask**	**GroupNonUniformBallot**
`get_sub_group_le_mask`	**SubgroupLeMask**	**GroupNonUniformBallot**
`get_sub_group_lt_mask`	**SubgroupLtMask**	**GroupNonUniformBallot**
For `cl_khr_subgroup_non_uniform_arithmetic`:		
`sub_group_non_uniform_reduce_add`	**OpGroupNonUniformIAdd**, **OpGroupNonUniformFAdd**	**GroupNonUniformArithmetic**
`sub_group_non_uniform_reduce_mul`	**OpGroupNonUniformIMul**, **OpGroupNonUniformFMul**	**GroupNonUniformArithmetic**

OpenCL C Function	SPIR-V BuiltIn or Instruction	Enabling SPIR-V Capability
sub_group_non_uniform_reduce_min	OpGroupNonUniformSMin, OpGroupNonUniformUMin, OpGroupNonUniformFMin	GroupNonUniformArithmetic
sub_group_non_uniform_reduce_max	OpGroupNonUniformSMax, OpGroupNonUniformUMax, OpGroupNonUniformFMax	GroupNonUniformArithmetic
sub_group_non_uniform_reduce_and	OpGroupNonUniformBitwiseAnd	GroupNonUniformArithmetic
sub_group_non_uniform_reduce_or	OpGroupNonUniformBitwiseOr	GroupNonUniformArithmetic
sub_group_non_uniform_reduce_xor	OpGroupNonUniformBitwiseXor	GroupNonUniformArithmetic
sub_group_non_uniform_reduce_logical_and	OpGroupNonUniformLogicalAnd	GroupNonUniformArithmetic
sub_group_non_uniform_reduce_logical_or	OpGroupNonUniformLogicalOr	GroupNonUniformArithmetic
sub_group_non_uniform_reduce_logical_xor	OpGroupNonUniformLogicalXor	GroupNonUniformArithmetic
sub_group_non_uniform_scan_inclusive_add	OpGroupNonUniformIAdd, OpGroupNonUniformFAdd	GroupNonUniformArithmetic
sub_group_non_uniform_scan_inclusive_mul	OpGroupNonUniformIMul, OpGroupNonUniformFMul	GroupNonUniformArithmetic
sub_group_non_uniform_scan_inclusive_min	OpGroupNonUniformSMin, OpGroupNonUniformUMin, OpGroupNonUniformFMin	GroupNonUniformArithmetic
sub_group_non_uniform_scan_inclusive_max	OpGroupNonUniformSMax, OpGroupNonUniformUMax, OpGroupNonUniformFMax	GroupNonUniformArithmetic
sub_group_non_uniform_scan_inclusive_and	OpGroupNonUniformBitwiseAnd	GroupNonUniformArithmetic
sub_group_non_uniform_scan_inclusive_or	OpGroupNonUniformBitwiseOr	GroupNonUniformArithmetic
sub_group_non_uniform_scan_inclusive_xor	OpGroupNonUniformBitwiseXor	GroupNonUniformArithmetic
sub_group_non_uniform_scan_inclusive_logical_and	OpGroupNonUniformLogicalAnd	GroupNonUniformArithmetic
sub_group_non_uniform_scan_inclusive_logical_or	OpGroupNonUniformLogicalOr	GroupNonUniformArithmetic

OpenCL C Function	SPIR-V BuiltIn or Instruction	Enabling SPIR-V Capability
sub_group_non_uniform_scan_inclusive_logical_xor	**OpGroupNonUniformLogicalXor**	**GroupNonUniformArithmetic**
sub_group_non_uniform_scan_exclusive_add	**OpGroupNonUniformIAdd**, **OpGroupNonUniformFAdd**	**GroupNonUniformArithmetic**
sub_group_non_uniform_scan_exclusive_mul	**OpGroupNonUniformIMul**, **OpGroupNonUniformFMul**	**GroupNonUniformArithmetic**
sub_group_non_uniform_scan_exclusive_min	**OpGroupNonUniformSMin**, **OpGroupNonUniformUMin**, **OpGroupNonUniformFMin**	**GroupNonUniformArithmetic**
sub_group_non_uniform_scan_exclusive_max	**OpGroupNonUniformSMax**, **OpGroupNonUniformUMax**, **OpGroupNonUniformFMax**	**GroupNonUniformArithmetic**
sub_group_non_uniform_scan_exclusive_and	**OpGroupNonUniformBitwiseAnd**	**GroupNonUniformArithmetic**
sub_group_non_uniform_scan_exclusive_or	**OpGroupNonUniformBitwiseOr**	**GroupNonUniformArithmetic**
sub_group_non_uniform_scan_exclusive_xor	**OpGroupNonUniformBitwiseXor**	**GroupNonUniformArithmetic**
sub_group_non_uniform_scan_exclusive_logical_and	**OpGroupNonUniformLogicalAnd**	**GroupNonUniformArithmetic**
sub_group_non_uniform_scan_exclusive_logical_or	**OpGroupNonUniformLogicalOr**	**GroupNonUniformArithmetic**
sub_group_non_uniform_scan_exclusive_logical_xor	**OpGroupNonUniformLogicalXor**	**GroupNonUniformArithmetic**
For cl_khr_subgroup_shuffle:		
sub_group_shuffle	**OpGroupNonUniformShuffle**	**GroupNonUniformShuffle**
sub_group_shuffle_xor	**OpGroupNonUniformShuffleXor**	**GroupNonUniformShuffle**
For cl_khr_subgroup_shuffle_relative:		
sub_group_shuffle_up	**OpGroupNonUniformShuffleUp**	**GroupNonUniformShuffleRelative**
sub_group_shuffle_down	**OpGroupNonUniformShuffleDown**	**GroupNonUniformShuffleRelative**
For cl_khr_subgroup_clustered_reduce:		
sub_group_clustered_reduce_add	**OpGroupNonUniformIAdd**, **OpGroupNonUniformFAdd**	**GroupNonUniformClustered**
sub_group_clustered_reduce_mul	**OpGroupNonUniformIMul**, **OpGroupNonUniformFMul**	**GroupNonUniformClustered**

OpenCL C Function	SPIR-V BuiltIn or Instruction	Enabling SPIR-V Capability
`sub_group_clustered_reduce_min`	**OpGroupNonUniformSMin**, **OpGroupNonUniformUMin**, **OpGroupNonUniformFMin**	**GroupNonUniformClustered**
`sub_group_clustered_reduce_max`	**OpGroupNonUniformSMax**, **OpGroupNonUniformUMax**, **OpGroupNonUniformFMax**	**GroupNonUniformClustered**
`sub_group_clustered_reduce_and`	**OpGroupNonUniformBitwise And**	**GroupNonUniformClustered**
`sub_group_clustered_reduce_or`	**OpGroupNonUniformBitwise Or**	**GroupNonUniformClustered**
`sub_group_clustered_reduce_xor`	**OpGroupNonUniformBitwise Xor**	**GroupNonUniformClustered**
`sub_group_clustered_reduce_logical_and`	**OpGroupNonUniformLogical And**	**GroupNonUniformClustered**
`sub_group_clustered_reduce_logical_or`	**OpGroupNonUniformLogical Or**	**GroupNonUniformClustered**
`sub_group_clustered_reduce_logical_xor`	**OpGroupNonUniformLogical Xor**	**GroupNonUniformClustered**

Chapter 39. PCI Bus Information Query

This extension adds a new query to obtain PCI bus information about an OpenCL device.

Not all OpenCL devices have PCI bus information, either due to the device not being connected to the system through a PCI interface or due to platform specific restrictions and policies. Thus this extension is only expected to be supported by OpenCL devices which can provide the information.

As a consequence, applications should always check for the presence of the extension string for each individual OpenCL device for which they intend to issue the new query for and should not have any assumptions about the availability of the extension on any given platform.

39.1. General Information

39.1.1. Name Strings

`cl_khr_pci_bus_info`

39.1.2. Version History

Date	Version	Description
2021-04-19	1.0.0	Initial version.

39.1.3. Dependencies

This extension is written against the OpenCL API Specification Version V3.0.6.

This extension requires OpenCL 1.0.

39.2. New API Types

Structure returned by the device info query for `CL_DEVICE_PCI_BUS_INFO_KHR`:

```
typedef struct cl_device_pci_bus_info_khr {
    cl_uint    pci_domain;
    cl_uint    pci_bus;
    cl_uint    pci_device;
    cl_uint    pci_function;
} cl_device_pci_bus_info_khr;
```

39.3. New API Enums

Accepted value for the *param_name* parameter to **clGetDeviceInfo**:

```
#define CL_DEVICE_PCI_BUS_INFO_KHR  0x410F
```

39.4. Modifications to the OpenCL API Specification

39.4.1. Section 4.2 - Querying Devices:

Add to Table 5 - OpenCL Device Queries:

Table 5. OpenCL Device Queries

DeviceInfo	Return Type	Description
`CL_DEVICE_PCI_BUS_INFO_KHR`	`cl_device_pci_bus_info_khr`	Returns PCI bus information for the device. The PCI bus information is returned as a single structure that includes the PCI bus domain, the PCI bus identifier, the PCI device identifier, and the PCI device function identifier.

Chapter 40. Extended Bit Operations

This extension adds OpenCL C functions for performing extended bit operations. Specifically, the following functions are added:

- bitfield insert: insert bits from one source operand into another source operand.
- bitfield extract: extract bits from a source operand, with sign- or zero-extension.
- bit reverse: reverse the bits of a source operand.

40.1. General Information

40.1.1. Name Strings

`cl_khr_extended_bit_ops`

40.1.2. Version History

Date	Version	Description
2021-04-22	1.0.0	Initial version.

40.1.3. Dependencies

This extension is written against the OpenCL 3.0 C Language Specification and the OpenCL SPIR-V Environment Specification Version V3.0.6.

This extension requires OpenCL 1.0.

40.2. New OpenCL C Functions

```
gentype bitfield_insert( gentype base, gentype insert, uint offset, uint count )
igentype bitfield_extract_signed( gentype base, uint offset, uint count )
ugentype bitfield_extract_unsigned( gentype base, uint offset, uint count )
gentype bit_reverse( gentype base )
```

40.3. Modifications to the OpenCL C Specification

40.3.1. Modify Section 6.15.3. Integer Functions:

Add a new Section 6.15.3.X. Extended Bit Operations:

> The functions described in the following table can be used with built-in scalar or vector integer types to perform extended bit operations. The functions that operate on vector types operate component-wise. The description is per-component.
>
> In the table below, the generic type name `gentype` refers to the built-in integer types `char`, `charn`,

uchar, ucharn, short, shortn, ushort, ushortn, int, intn, uint, uintn, long, longn, ulong, and ulongn. The generic type name igentype refers to the built-in signed integer types char, charn, short, shortn, int, intn, long, and longn. The generic type name ugentype refers to the built-in unsigned integer types uchar, ucharn, ushort, ushortn, uint, uintn, ulong, and ulongn. *n* is 2, 3, 4, 8, or 16.

Table 53. Built-in Scalar and Vector Extended Bit Operations

Function	Description
`gentype bitfield_insert( gentype base, gentype insert, uint offset, uint count)`	Returns a copy of *base*, with a modified bitfield that comes from *insert*. Any bits of the result value numbered outside [*offset*, *offset* + *count* - 1] (inclusive) will come from the corresponding bits in *base*. Any bits of the result value numbered inside [*offset*, *offset* + *count* - 1] (inclusive) will come from the bits numbered [0, *count* - 1] (inclusive) of *insert*. *count* is the number of bits to be modified. If *count* equals 0, the return value will be equal to *base*. If *count* or *offset* or *offset* + *count* is greater than number of bits in gentype (for scalar types) or components of gentype (for vector types), the result is undefined.
`igentype bitfield_extract_signed( gentype base, uint offset, uint count)`	Returns an extracted bitfield from *base* with sign extension. The type of the return value is always a signed type. The bits of *base* numbered in [*offset*, *offset* + *count* - 1] (inclusive) are returned as the bits numbered in [0, *count* - 1] (inclusive) of the result. The remaining bits in the result will be sign extended by replicating the bit numbered *offset* + *count* - 1 of *base*. *count* is the number of bits to be extracted. If *count* equals 0, the result is 0. If the *count* or *offset* or *offset* + *count* is greater than number of bits in gentype (for scalar types) or components of gentype (for vector types), the result is undefined.

Function	Description
`ugentype bitfield_extract_unsigned( gentype base, uint offset, uint count)`	Returns an extracted bitfield from *base* with zero extension. The type of the return value is always an unsigned type. The bits of *base* numbered in [*offset*, *offset* + *count* - 1] (inclusive) are returned as the bits numbered in [0, *count* - 1] (inclusive) of the result. The remaining bits in the result will be zero. *count* is the number of bits to be extracted. If *count* equals 0, the result is 0. If the *count* or *offset* or *offset* + *count* is greater than number of bits in `gentype` (for scalar types) or components of `gentype` (for vector types), the result is undefined.
`gentype bit_reverse( gentype base)`	Returns the value of *base* with reversed bits. That is, the bit numbered *n* of the result value will be taken from the bit numbered *width* - *n* - 1 of *base* (for scalar types) or a component of *base* (for vector types), where *width* is number of bits of `gentype` (for scalar types) or components of `gentype` (for vector types).

Chapter 41. Suggested Local Work Size Query

This extension adds the ability to query a suggested local work-group size for a kernel running on a device for a specified global work size and global work offset. The suggested local work-group size will match the work-group size that would be chosen if the kernel were enqueued with the specified global work size and global work offset and a `NULL` local work size.

By using the suggested local work-group size query an application has greater insight into the local work-group size chosen by the OpenCL implementation, and the OpenCL implementation need not re-compute the local work-group size if the same kernel is enqueued multiple times with the same parameters.

41.1. General Information

41.1.1. Name Strings

`cl_khr_suggested_local_work_size`

41.1.2. Version History

Date	Version	Description
2021-04-22	1.0.0	Initial version.

41.1.3. Dependencies

This extension is written against the OpenCL API Specification Version V3.0.6.

This extension requires OpenCL 1.0.

41.2. New API Functions

```
cl_int  clGetKernelSuggestedLocalWorkSizeKHR(
    cl_command_queue command_queue,
    cl_kernel kernel,
    cl_uint work_dim,
    const size_t *global_work_offset,
    const size_t *global_work_size,
    size_t *suggested_local_work_size);
```

41.3. Modifications to the OpenCL API Specification

41.3.1. Section 5.9 - Kernel Objects:

41.3.1.1. New Section 5.9.4.X - Suggested Local Work Size Query

To query a suggested local work size for a kernel object, call the function

```
cl_int clGetKernelSuggestedLocalWorkSizeKHR(
    cl_command_queue command_queue,
    cl_kernel kernel,
    cl_uint work_dim,
    const size_t* global_work_offset,
    const size_t* global_work_size,
    size_t* suggested_local_work_size);
```

The returned suggested local work size is expected to match the local work size that would be chosen if the specified kernel object, with the same kernel arguments, were enqueued into the specified command-queue with the specified global work size, specified global work offset, and with a `NULL` local work size.

- *command_queue* specifies the command-queue and device for the query.
- *kernel* specifies the kernel object and kernel arguments for the query. The OpenCL context associated with *kernel* and *command_queue* must the same.
- *work_dim* specifies the number of work dimensions in the input global work offset and global work size, and the output suggested local work size.
- *global_work_offset* can be used to specify an array of at least *work_dim* global ID offset values for the query. This is optional and may be `NULL` to indicate there is no global ID offset.
- *global_work_size* is an array of at least *work_dim* values describing the global work size for the query.
- *suggested_local_work_size* is an output array of at least *work_dim* values that will contain the result of the query.

clGetKernelSuggestedLocalWorkSizeKHR returns `CL_SUCCESS` if the query executed successfully. Otherwise, it returns one of the following errors:

- `CL_INVALID_COMMAND_QUEUE` if *command_queue* is not a valid host command-queue.
- `CL_INVALID_KERNEL` if *kernel* is not a valid kernel object.
- `CL_INVALID_CONTEXT` if the context associated with *kernel* is not the same as the context associated with *command_queue*.
- `CL_INVALID_PROGRAM_EXECUTABLE` if there is no successfully built program executable available for *kernel* for the device associated with *command_queue*.
- `CL_INVALID_KERNEL_ARGS` if all argument values for *kernel* have not been set.
- `CL_MISALIGNED_SUB_BUFFER_OFFSET` if a sub-buffer object is set as an argument to *kernel* and the offset specified when the sub-buffer object was created is not aligned to `CL_DEVICE_MEM_BASE_ADDR_ALIGN` for the device associated with *command_queue*.

- `CL_INVALID_IMAGE_SIZE` if an image object is set as an argument to *kernel* and the image dimensions are not supported by device associated with *command_queue*.
- `CL_IMAGE_FORMAT_NOT_SUPPORTED` if an image object is set as an argument to *kernel* and the image format is not supported by the device associated with *command_queue*.
- `CL_INVALID_OPERATION` if an SVM pointer is set as an argument to *kernel* and the device associated with *command_queue* does not support SVM or the required SVM capabilities for the SVM pointer.
- `CL_INVALID_WORK_DIMENSION` if *work_dim* is not a valid value (i.e. a value between 1 and `CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS`).
- `CL_INVALID_GLOBAL_WORK_SIZE` if *global_work_size* is NULL or if any of the values specified in *global_work_size* are 0.
- `CL_INVALID_GLOBAL_WORK_SIZE` if any of the values specified in *global_work_size* exceed the maximum value representable by `size_t` on the device associated with *command_queue*.
- `CL_INVALID_GLOBAL_OFFSET` if the value specified in *global_work_size* plus the corresponding value in *global_work_offset* for dimension exceeds the maximum value representable by `size_t` on the device associated with *command_queue*.
- `CL_INVALID_VALUE` if *suggested_local_work_size* is NULL.
- `CL_OUT_OF_RESOURCES` if there is a failure to allocate resources required by the OpenCL implementation on the device.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

> These error conditions are consistent with error conditions for **clEnqueueNDRangeKernel**.

Chapter 42. Integer dot product

This extension adds support for SPIR-V instructions and OpenCL C built-in functions to compute the dot product of vectors of integers.

42.1. General Information

42.1.1. Name Strings

`cl_khr_integer_dot_product`

42.1.2. Version History

Date	Version	Description
2021-06-17	1.0.0	Initial version.
2021-06-23	2.0.0	All 8-bit support is mandatory, added 8-bit acceleration properties.

42.1.3. Dependencies

This extension is written against the OpenCL Specification Version 3.0.6, and OpenCL C Specification Version 3.0.6 and OpenCL Environment Specification Version 3.0.6.

This extension requires OpenCL 1.0.

42.1.4. Contributors

Kévin Petit, Arm Ltd.
Jeremy Kemp, Imagination Technologies
Ben Ashbaugh, Intel
Ruihao Zhang, Qualcomm
Stuart Brady, Arm Ltd
Balaji Calidas, Qualcomm
Ayal Zaks, Intel

42.2. New API Enums

Accepted value for the *param_name* parameter to **clGetDeviceInfo**:

```
CL_DEVICE_INTEGER_DOT_PRODUCT_INPUT_4x8BIT_PACKED_KHR      (1 << 0)
CL_DEVICE_INTEGER_DOT_PRODUCT_INPUT_4x8BIT_KHR             (1 << 1)

CL_DEVICE_INTEGER_DOT_PRODUCT_CAPABILITIES_KHR              0x1073

CL_DEVICE_INTEGER_DOT_PRODUCT_ACCELERATION_PROPERTIES_8BIT_KHR          0x1074
```

```
CL_DEVICE_INTEGER_DOT_PRODUCT_ACCELERATION_PROPERTIES_4x8BIT_PACKED_KHR  0x1075
```

42.3. New OpenCL C Functions

This extension defines a number of new functions that operate on vectors of integers. The exact function overloads available depend on the features supported by the target device.

```
uint dot(uchar4 a, uchar4 b);
int dot(char4 a, char4 b);
int dot(uchar4 a, char4 b);
int dot(char4 a, uchar4 b);

uint dot_acc_sat(uchar4 a, uchar4 b, uint acc);
int dot_acc_sat(char4 a, char4 b, int acc);
int dot_acc_sat(uchar4 a, char4 b, int acc);
int dot_acc_sat(char4 a, uchar4 b, int acc);

uint dot_4x8packed_uu_uint(uint a, uint b);
int dot_4x8packed_ss_int(uint a, uint b);
int dot_4x8packed_us_int(uint a, uint b);
int dot_4x8packed_su_int(uint a, uint b);

uint dot_acc_sat_4x8packed_uu_uint(uint a, uint b, uint acc);
int dot_acc_sat_4x8packed_ss_int(uint a, uint b, int acc);
int dot_acc_sat_4x8packed_us_int(uint a, uint b, int acc);
int dot_acc_sat_4x8packed_su_int(uint a, uint b, int acc);
```

42.4. Modifications to the OpenCL API Specification

(Modify Section 4.2, Querying Devices)

(Add the following to Table 4.3, *Device Queries*)

Device Info	Return Type	Description
CL_DEVICE_INTEGER_DOT_PRODUCT_CAPABILITIES_KHR	cl_device_integer_dot_product_capabilities_khr	Returns the integer dot product capabilities supported by the device. CL_DEVICE_INTEGER_DOT_PRODUCT_INPUT_4x8BIT_PACKED_KHR is always set indicating that all implementations that support cl_khr_integer_dot_product must support dot product built-in functions and, when SPIR-V is supported, SPIR-V instructions that take four-component vectors of 8-bit integers packed into 32-bit integers as input. CL_DEVICE_INTEGER_DOT_PRODUCT_INPUT_4x8BIT_KHR is set when dot product built-in functions and, when SPIR-V is supported, SPIR-V instructions that take four-component of 8-bit elements as input are supported. NOTE: CL_DEVICE_INTEGER_DOT_PRODUCT_INPUT_4x8BIT_KHR must be set in version 2.x of the extension.
CL_DEVICE_INTEGER_DOT_PRODUCT_ACCELERATION_PROPERTIES_8BIT_KHR	cl_device_integer_dot_product_acceleration_properties_khr	Returns a structure describing the exact 8-bit dot product combinations that are accelerated on the device. Each member is CL_TRUE if the combination it corresponds to is accelerated, CL_FALSE otherwise. NOTE: CL_DEVICE_INTEGER_DOT_PRODUCT_ACCELERATION_PROPERTIES_8BIT_KHR is missing before version 2.0 of the extension.
CL_DEVICE_INTEGER_DOT_PRODUCT_ACCELERATION_PROPERTIES_4x8BIT_PACKED_KHR	cl_device_integer_dot_product_acceleration_properties_khr	Returns a structure describing the exact 4x8-bit packed dot product combinations that are accelerated on the device. Each member is CL_TRUE if the combination it corresponds to is accelerated, CL_FALSE otherwise. NOTE: CL_DEVICE_INTEGER_DOT_PRODUCT_ACCELERATION_PROPERTIES_4x8BIT_PACKED_KHR is missing before version 2.0 of the extension.

OpenCL 3 devices must report the following feature macros via CL_DEVICE_OPENCL_C_FEATURES when the corresponding bit is set in the bitfield returned for CL_DEVICE_INTEGER_DOT_PRODUCT_CAPABILITIES_KHR:

Feature bit	Feature macro
CL_DEVICE_INTEGER_DOT_PRODUCT_INPUT_4x8BIT_PACKED_KHR	__opencl_c_integer_dot_product_input_4x8bit_packed

Feature bit	Feature macro
CL_DEVICE_INTEGER_DOT_PRODUCT_INPUT_4x8BIT_KHR	__opencl_c_integer_dot_product_input_4x8bit

The cl_device_integer_dot_product_acceleration_properties_khr structure describes the exact dot product operations that are accelerated on the device:

```
typedef struct cl_device_integer_dot_product_acceleration_properties_khr {
    cl_bool    signed_accelerated;
    cl_bool    unsigned_accelerated;
    cl_bool    mixed_signedness_accelerated;
    cl_bool    accumulating_saturating_signed_accelerated;
    cl_bool    accumulating_saturating_unsigned_accelerated;
    cl_bool    accumulating_saturating_mixed_signedness_accelerated;
} cl_device_integer_dot_product_acceleration_properties_khr;
```

- *signed_accelerated* is CL_TRUE when signed dot product operations are accelerated, CL_FALSE otherwise.
- *unsigned_accelerated* is CL_TRUE when unsigned dot product operations are accelerated, CL_FALSE otherwise.
- *mixed_signedness_accelerated* is CL_TRUE when mixed signedness dot product operations are accelerated, CL_FALSE otherwise.
- *accumulating_saturating_signed_accelerated* is CL_TRUE when accumulating saturating signed dot product operations are accelerated, CL_FALSE otherwise.
- *accumulating_saturating_unsigned_accelerated* is CL_TRUE when accumulating saturating unsigned dot product operations are accelerated, CL_FALSE otherwise.
- *accumulating_saturating_mixed_signedness_accelerated* is CL_TRUE when accumulating saturating mixed signedness dot product operations are accelerated, CL_FALSE otherwise.

A dot product operation is deemed accelerated if its implementation provides a performance advantage over application-provided code composed from elementary instructions and/or other dot product instructions, either because the implementation uses optimized machine code sequences whose generation from application-provided code cannot be guaranteed or because it uses hardware features that cannot otherwise be targeted from application-provided code.

42.5. Modifications to the OpenCL C Specification

(Modify section 6.13.3, Integer Functions)

The following built-in functions and preprocessor definitions are added:

```
#define cl_khr_integer_dot_product 1

if (CL_DEVICE_INTEGER_DOT_PRODUCT_INPUT_4x8BIT_PACKED_KHR) {
    #define __opencl_c_integer_dot_product_input_4x8bit_packed 1
```

```
    uint dot_4x8packed_uu_uint(uint a, uint b);
    int dot_4x8packed_ss_int(uint a, uint b);
    int dot_4x8packed_us_int(uint a, uint b);
    int dot_4x8packed_su_int(uint a, uint b);

    uint dot_acc_sat_4x8packed_uu_uint(uint a, uint b, uint acc);
    int dot_acc_sat_4x8packed_ss_int(uint a, uint b, int acc);
    int dot_acc_sat_4x8packed_us_int(uint a, uint b, int acc);
    int dot_acc_sat_4x8packed_su_int(uint a, uint b, int acc);
}

if (CL_DEVICE_INTEGER_DOT_PRODUCT_INPUT_4x8BIT_KHR) {
    #define __opencl_c_integer_dot_product_input_4x8bit 1

    uint dot(uchar4 a, uchar4 b);
    int dot(char4 a, char4 b);
    int dot(uchar4 a, char4 b);
    int dot(char4 a, uchar4 b);

    uint dot_acc_sat(uchar4 a, uchar4 b, uint acc);
    int dot_acc_sat(char4 a, char4 b, int acc);
    int dot_acc_sat(uchar4 a, char4 b, int acc);
    int dot_acc_sat(char4 a, uchar4 b, int acc);
}
```

- dot returns the dot product of the two input vectors a and b. The components of a and b are sign- or zero-extended to the width of the destination type and the vectors with extended components are multiplied component-wise. All the components of the resulting vectors are added together to form the final result.
- dot_acc_sat returns the saturating addition of the dot product of the two input vectors a and b and the accumulator acc:

```
product = dot(a,b);
result = add_sat(product, acc);
```

- dot_*_4x8packed_XY_R returns the dot product of the two vectors packed into a and b (lowest component in least significant byte). The components are unpacked, sign- or zero-extended to the width of the destination type before the multiplications and additions. X represents the signedness of the components of a, Y that of the components of b. R is the return type.

42.6. Modifications to the OpenCL SPIR-V Environment Specification

See OpenCL SPIR-V Environment Specification.

42.7. Interactions with Other Extensions

If `cl_khr_il_program` is supported then the SPIR-V environment specification modifications described above apply.

Chapter 43. Semaphores (Provisional)

OpenCL provides `cl_event` as a primary mechanism of synchronization between host and device as well as across devices. While events can be waited on or can be passed as dependencies across work-submissions, they suffer from following limitations:

- They are immutable.
- They are not reusable.

This extension introduces a new type of synchronization object to represent semaphores that can be reused, waited on, and signaled multiple times by OpenCL work-submissions.

In particular, this extension defines:

- A new type called `cl_semaphore_khr` to represent the semaphore objects.
- A new type called `cl_semaphore_properties_khr` to specify metadata associated with semaphores.
- Routines to create, retain, and release semaphores.
- Routines to wait on and signal semaphore objects.
- Routine to query the properties of semaphore objects.

43.1. General Information

43.1.1. Name Strings

`cl_khr_semaphore`

43.1.2. Version History

Date	Version	Description
2021-09-10	0.9.0	Initial version (provisional).
2023-08-01	0.9.1	Changed device handle list enum to the semaphore-specific `CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR` (provisional).

> This is a provisional OpenCL extension specification that has been Ratified under the Khronos Intellectual Property Framework. It is being made publicly available as a provisional extension to enable review and feedback from the community. While it is a provisional extension features may be added, removed, or changed in non-backward compatible ways.
>
> If you have feedback please create an issue on: https://github.com/KhronosGroup/OpenCL-Docs/

43.1.3. Dependencies

This extension is written against the OpenCL Specification Version 3.0.8.

This extension requires OpenCL 1.2.

43.1.4. Contributors

Ajit Hakke-Patil, NVIDIA
Amit Rao, NVIDIA
Balaji Calidas, QUALCOMM
Ben Ashbaugh, INTEL
Carsten Rohde, NVIDIA
Christoph Kubisch, NVIDIA
Debalina Bhattacharjee, NVIDIA
Faith Ekstrand, INTEL
James Jones, NVIDIA
Jeremy Kemp, IMAGINATION
Joshua Kelly, QUALCOMM
Karthik Raghavan Ravi, NVIDIA
Kedar Patil, NVIDIA
Kevin Petit, ARM
Nikhil Joshi, NVIDIA
Sharan Ashwathnarayan, NVIDIA
Vivek Kini, NVIDIA

43.2. New Types

```
typedef struct _cl_semaphore_khr* cl_semaphore_khr;

typedef cl_properties cl_semaphore_properties_khr;
typedef cl_uint cl_semaphore_info_khr;
typedef cl_uint cl_semaphore_type_khr;
typedef cl_ulong cl_semaphore_payload_khr;
```

43.3. New API Functions

```
cl_semaphore_khr clCreateSemaphoreWithPropertiesKHR(
    cl_context context,
    const cl_semaphore_properties_khr *sema_props,
    cl_int *errcode_ret);

cl_int clEnqueueWaitSemaphoresKHR(
    cl_command_queue command_queue,
    cl_uint num_sema_objects,
    const cl_semaphore_khr *sema_objects,
    const cl_semaphore_payload_khr *sema_payload_list,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list,
    cl_event *event);
```

```
cl_int clEnqueueSignalSemaphoresKHR(
    cl_command_queue command_queue,
    cl_uint num_sema_objects,
    const cl_semaphore_khr *sema_objects,
    const cl_semaphore_payload_khr *sema_payload_list,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list,
    cl_event *event);

cl_int clGetSemaphoreInfoKHR(
    cl_semaphore_khr sema_object,
    cl_semaphore_info_khr param_name,
    size_t param_value_size,
    void *param_value,
    size_t *param_value_size_ret);

cl_int clReleaseSemaphoreKHR(cl_semaphore_khr sema_object);

cl_int clRetainSemaphoreKHR(cl_semaphore_khr sema_object);
```

43.4. New API Enums

Accepted value for the *param_name* parameter to **clGetPlatformInfo** to query the semaphore types supported by an OpenCL platform:

```
CL_PLATFORM_SEMAPHORE_TYPES_KHR                         0x2036
```

Accepted value for the *param_name* parameter to **clGetDeviceInfo** to query the semaphore types supported by an OpenCL device:

```
CL_DEVICE_SEMAPHORE_TYPES_KHR                           0x204C
```

Semaphore types:

```
CL_SEMAPHORE_TYPE_BINARY_KHR                            1
```

New attributes that can be passed as part of `cl_semaphore_info_khr`:

```
CL_SEMAPHORE_CONTEXT_KHR                                0x2039
CL_SEMAPHORE_REFERENCE_COUNT_KHR                        0x203A
CL_SEMAPHORE_PROPERTIES_KHR                             0x203B
CL_SEMAPHORE_PAYLOAD_KHR                                0x203C
```

New attributes that can be passed as part of `cl_semaphore_info_khr` or `cl_semaphore_properties_khr`:

```
CL_SEMAPHORE_TYPE_KHR                                   0x203D
CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR                     0x2053
CL_SEMAPHORE_DEVICE_HANDLE_LIST_END_KHR                 0
```

New return values from **clGetEventInfo** when *param_name* is `CL_EVENT_COMMAND_TYPE`:

```
CL_COMMAND_SEMAPHORE_WAIT_KHR                           0x2042
CL_COMMAND_SEMAPHORE_SIGNAL_KHR                         0x2043
```

The following error codes can be returned by APIs introduced as part of this specification or the specifications that depend on this:

```
CL_INVALID_SEMAPHORE_KHR                                -1142
```

43.5. Modifications to existing APIs added by this spec

Following new enums are added to the list of supported *param_names* by **clGetPlatformInfo**:

Table 54. List of supported param_names by **clGetPlatformInfo**

Platform Info	**Return Type**	**Description**
`CL_PLATFORM_SEMAPHORE_TYPES_KHR`	`cl_semaphore_type_khr`[]	Returns the list of the semaphore types supported all devices in *platform*.

clGetPlatformInfo when called with *param_name* `CL_PLATFORM_SEMAPHORE_TYPES_KHR` must return common list of semaphore types supported by all devices in the platform.

Following new enums are added to the list of supported *param_names* by **clGetDeviceInfo**:

Table 55. List of supported param_names by **clGetDeviceInfo**

Device Info	**Return Type**	**Description**
`CL_DEVICE_SEMAPHORE_TYPES_KHR`	`cl_semaphore_type_khr`[]	Returns the list of the semaphore types supported by *device*.

clGetDeviceInfo when called with param_name `CL_DEVICE_SEMAPHORE_TYPES_KHR` must return a non-empty list of semaphore types for at least one of the devices in the platform. The results of this query should meet minimum requirements for `cl_semaphore_type_khr` as described by Description of new types added by this spec.

43.6. Description of new types added by this spec

Following new types are added:

- `cl_semaphore_type_khr` to represent the different types of semaphores.

◦ It is mandatory to support CL_SEMAPHORE_TYPE_BINARY_KHR.

- cl_semaphore_properties_khr to represent properties associated with semaphores.
  - CL_SEMAPHORE_TYPE_KHR must be supported.
- cl_semaphore_info_khr to represent queries to get additional information about semaphores.
  - All enums described in New API Enums for cl_semaphore_info_khr must be supported.
- cl_semaphore_payload_khr to represent payload values of semaphores.
- cl_semaphore_khr to represent semaphore objects.

Note that above types can be extended in future based on the need for additional types of semaphore and properties required by them. The specifics of the same can be added as a newer version of this specification or by a separate specification that depends on this for basic semaphore support.

43.7. Description of new APIs added by this spec

The following new APIs are added as part of this spec. The details of each are described below:

43.7.1. Creating semaphores

A **semaphore object** may be created using the function

```
cl_semaphore_khr clCreateSemaphoreWithPropertiesKHR(
    cl_context context,
    const cl_semaphore_properties_khr* sema_props,
    cl_int* errcode_ret);
```

context identifies a valid OpenCL context that the created cl_semaphore_khr will belong to.

sema_props specifies additional semaphore properties in the form list of <property_name, property_value> pairs terminated with 0. CL_SEMAPHORE_TYPE_KHR must be part of the list of properties specified by *sema_props*.

Following new properties are added to the list of possible supported properties by cl_semaphore_properties_khr that can be passed to **clCreateSemaphoreWithPropertiesKHR**:

Table 56. List of supported semaphore creation properties by **clCreateSemaphoreWithPropertiesKHR**

Semaphore Property	Property Value	Description
CL_SEMAPHORE_TYPE_KHR	cl_semaphore_type_khr	Specifies the type of semaphore to create. This property is always required.
CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR	cl_device_id[]	Specifies the list of OpenCL devices (terminated with CL_SEMAPHORE_DEVICE_HANDLE_LIST_END_KHR) to associate with the semaphore. Only a single device is permitted in the list.

If CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR is not specified as part of *sema_props*, the semaphore object created by **clCreateSemaphoreWithPropertiesKHR** is by default accessible to all devices in the *context*. For a multi-device context CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR must be specified in *sema_props*.

errcode_ret returns an appropriate error code. If *errcode_ret* is NULL, no error code is returned.

clCreateSemaphoreWithPropertiesKHR returns a valid semaphore object in an un-signaled state and and *errcode_ret* is set to CL_SUCCESS if the function is executed successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_CONTEXT if *context* is not a valid context.
- CL_INVALID_PROPERTY if a property name in *sema_props* is not a supported property name, if the value specified for a supported property name is not valid, or if the same property name is specified more than once. Additionally, if *context* is a multiple device context and *sema_props* does not specify CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR.
- CL_INVALID_DEVICE if CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR is specified as part of *sema_props*, but it does not identify exactly one valid device or if a device identified by CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR is not one of the devices within *context*.
- CL_INVALID_VALUE
  - if *sema_props* is NULL, or
  - if *sema_props* do not specify <property, value> pairs for minimum set of properties (i.e. CL_SEMAPHORE_TYPE_KHR) required for successful creation of a cl_semaphore_khr, or
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

43.7.2. Waiting on and signaling semaphores

To enqueue a command to wait on a set of semaphores, call the function

```
cl_int clEnqueueWaitSemaphoresKHR(
    cl_command_queue command_queue,
    cl_uint num_sema_objects,
    const cl_semaphore_khr* sema_objects,
    const cl_semaphore_payload_khr* sema_payload_list,
    cl_uint num_events_in_wait_list,
    const cl_event* event_wait_list,
    cl_event* event);
```

command_queue specifies a valid command-queue.

num_sema_objects specifies the number of semaphore objects to wait on.

sema_objects points to the list of semaphore objects to wait on. The length of the list must be at least *num_sema_objects*.

sema_payload_list points to the list of values of type `cl_semaphore_payload_khr` containing valid semaphore payload values to wait on. This can be set to `NULL` or will be ignored when all semaphores in the list of *sema_objects* are of type `CL_SEMAPHORE_TYPE_BINARY_KHR`.

num_events_in_wait_list specifies the number of events in *event_wait_list*.

event_wait_list specifies list of events that need to complete before **clEnqueueWaitSemaphoresKHR** can be executed. If *event_wait_list* is `NULL`, then **clEnqueueWaitSemaphoresKHR** does not wait on any event to complete. If *event_wait_list* is `NULL`, *num_events_in_wait_list* must be 0. If *event_wait_list* is not `NULL`, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points. The context associated with events in *event_wait_list* and that associated with *command_queue* must be the same.

event returns an event object that identifies this particular command and can be used to query or queue a wait for this particular command to complete. *event* can be `NULL` in which case it will not be possible for the application to query the status of this command or queue a wait for this command to complete.

The semaphore wait command waits for a list of events to complete and a list of semaphore objects to become signaled. The semaphore wait command returns an *event* which can be waited on to ensure that all events in the *event_wait_list* have completed and all semaphores in *sema_objects* have been signaled. **clEnqueueWaitSemaphoresKHR** will not return until the binary semaphores in *sema_objects* are in a state that makes them safe to re-signal. If necessary, implementations may block in **clEnqueueWaitSemaphoresKHR** to ensure the correct state of semaphores when returning. There are no implications from this behavior for the state of *event* or the events in *event_wait_list* when **clEnqueueWaitSemaphoresKHR** returns. Waiting on the same binary semaphore twice without an interleaving signal may lead to undefined behavior.

clEnqueueWaitSemaphoresKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns one of the following errors:

- `CL_INVALID_COMMAND_QUEUE`
  - if *command_queue* is not a valid command-queue, or
  - if the device associated with *command_queue* is not same as one of the devices specified by `CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR` at the time of creating one or more of *sema_objects*, or
  - if one or more of *sema_objects* belong to a context that does not contain a device associated with *command_queue*.
- `CL_INVALID_VALUE` if *num_sema_objects* is 0.
- `CL_INVALID_SEMAPHORE_KHR` if any of the semaphore objects specified by *sema_objects* is not valid.
- `CL_INVALID_CONTEXT` if the context associated with *command_queue* and any of the semaphore objects in *sema_objects* are not the same or if the context associated with *command_queue* and that associated with events in *event_wait_list* are not the same.
- `CL_INVALID_VALUE` if any of the semaphore objects specified by *sema_objects* requires a semaphore payload and *sema_payload_list* is `NULL`.

- CL_INVALID_EVENT_WAIT_LIST
  - if *event_wait_list* is NULL and *num_events_in_wait_list* is not 0, or
  - if *event_wait_list* is not NULL and *num_events_in_wait_list* is 0, or
  - if event objects in *event_wait_list* are not valid events.
- CL_EXEC_STATUS_ERROR_FOR_EVENTS_IN_WAIT_LIST if the execution status of any of the events in *event_wait_list* is a negative integer value.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

To enqueue a command to signal a set of semaphores, call the function

```
cl_int clEnqueueSignalSemaphoresKHR(
    cl_command_queue command_queue,
    cl_uint num_sema_objects,
    const cl_semaphore_khr* sema_objects,
    const cl_semaphore_payload_khr* sema_payload_list,
    cl_uint num_events_in_wait_list,
    const cl_event* event_wait_list,
    cl_event* event);
```

command_queue specifies a valid command-queue.

num_sema_objects specifies the number of semaphore objects to signal.

sema_objects points to the list of semaphore objects to signal. The length of the list must be at least *num_sema_objects*.

sema_payload_list points to the list of values of type cl_semaphore_payload_khr containing semaphore payload values to signal. This can be set to NULL or will be ignored when all semaphores in the list of *sema_objects* are of type CL_SEMAPHORE_TYPE_BINARY_KHR.

num_events_in_wait_list specifies the number of events in event_wait_list.

event_wait_list points to the list of events that need to complete before **clEnqueueSignalSemaphoresKHR** can be executed. If *event_wait_list* is NULL, then **clEnqueueSignalSemaphoresKHR** does not wait on any event to complete. If *event_wait_list* is NULL, *num_events_in_wait_list* must be 0. If *event_wait_list* is not NULL, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points. The context associated with events in *event_wait_list* and that associated with *command_queue* must be the same.

event returns an event object that identifies this particular command and can be used to query or queue a wait for this particular command to complete. *event* can be NULL in which case it will not be possible for the application to query the status of this command or queue a wait for this command to complete.

The semaphore signal command waits for a list of events to complete and then signals a list of semaphore objects. The semaphore signal command returns an *event* which can be waited on to ensure that all events in the *event_wait_list* have completed and all semaphores in *sema_objects* have been signaled. The successful completion of the event generated by **clEnqueueSignalSemaphoresKHR** called on one or more semaphore objects of type `CL_SEMAPHORE_TYPE_BINARY_KHR` changes the state of the corresponding semaphore objects to signaled. **clEnqueueSignalSemaphoresKHR** will not return until the binary semaphores in *sema_objects* are in a state that makes them safe to wait on again. If necessary, implementations may block in **clEnqueueSignalSemaphoresKHR** to ensure the correct state of semaphores when returning. There are no implications from this behavior for the state of *event* or the events in *event_wait_list* when **clEnqueueSignalSemaphoresKHR** returns. Signaling the same binary semaphore twice without an interleaving wait may lead to undefined behavior.

clEnqueueSignalSemaphoresKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns one of the following errors:

- `CL_INVALID_COMMAND_QUEUE`
  - if *command_queue* is not a valid command-queue, or
  - if the device associated with *command_queue* is not same as one of the devices specified by `CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR` at the time of creating one or more of *sema_objects*, or
  - if one or more of *sema_objects* belong to a context that does not contain a device associated with *command_queue*.
- `CL_INVALID_VALUE` if *num_sema_objects* is 0
- `CL_INVALID_SEMAPHORE_KHR` if any of the semaphore objects specified by *sema_objects* is not valid.
- `CL_INVALID_CONTEXT` if the context associated with *command_queue* and any of the semaphore objects in *sema_objects* are not the same or if the context associated with *command_queue* and that associated with events in *event_wait_list* are not the same.
- `CL_INVALID_VALUE` if any of the semaphore objects specified by *sema_objects* requires a semaphore payload and *sema_payload_list* is `NULL`.
- `CL_INVALID_EVENT_WAIT_LIST`
  - if *event_wait_list* is `NULL` and *num_events_in_wait_list* is not 0, or
  - if *event_wait_list* is not `NULL` and *num_events_in_wait_list* is 0, or
  - if event objects in *event_wait_list* are not valid events.
- `CL_EXEC_STATUS_ERROR_FOR_EVENTS_IN_WAIT_LIST` if the execution status of any of the events in *event_wait_list* is a negative integer value.
- `CL_OUT_OF_RESOURCES` if there is a failure to allocate resources required by the OpenCL implementation on the device.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

43.7.3. Semaphore Queries

To query information about a semaphore object, call the function

```
cl_int clGetSemaphoreInfoKHR(
    cl_semaphore_khr sema_object,
    cl_semaphore_info_khr param_name,
    size_t param_value_size,
    void* param_value,
    size_t* param_value_size_ret);
```

sema_object specifies the semaphore object being queried.

param_name is a constant that specifies the semaphore information to query, and must be one of the values shown in the Semaphore Queries table.

param_value is a pointer to memory where the result of the query is returned as described in the Semaphore Queries table. If *param_value* is NULL, it is ignored.

param_value_size specifies the size in bytes of memory pointed to *param_value*. This size must be greater than or equal to the size of the return type described in the Semaphore Queries table.

param_value_size_ret returns the actual size in bytes of data being queried by *param_value*. If *param_value_size_ret* is NULL, it is ignored.

Table 57. List of supported param_names by **clGetSemaphoreInfoKHR**

Semaphore Info	Return Type	Description
CL_SEMAPHORE_CONTEXT_KHR	cl_context	Returns the context specified when the semaphore is created.
CL_SEMAPHORE_REFERENCE_COUNT_KHR [1]	cl_uint	Returns the semaphore reference count.
CL_SEMAPHORE_PROPERTIES_KHR	cl_semaphore_properties_khr[]	Return the properties argument specified in **clCreateSemaphoreWithPropertiesKHR**. The implementation must return the values specified in the properties argument in the same order and without including additional properties.
CL_SEMAPHORE_TYPE_KHR	cl_semaphore_type_khr	Returns the semaphore type.
CL_SEMAPHORE_PAYLOAD_KHR	cl_semaphore_payload_khr	Returns the semaphore payload value. For semaphores of type CL_SEMAPHORE_TYPE_BINARY_KHR, the payload value returned will be 0 if the semaphore is in an un-signaled state and 1 if it is in a signaled state.
CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR	cl_device_id[]	Returns the list of OpenCL devices the semaphore is associated with.

clGetSemaphoreInfoKHR returns CL_SUCCESS if the information is queried successfully. Otherwise, it returns one of the following errors:

- CL_INVALID_SEMAPHORE_KHR
  - if *sema_object* is not a valid semaphore
- CL_INVALID_VALUE
  - if *param_name* is not one of the attribute defined in the Semaphore Queries table or
  - if *param_value_size* is less than the size of Return Type of the corresponding *param_name* attribute as defined in the Semaphore Queries table.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

43.7.4. Retaining and Releasing Semaphores

To release a semaphore object, call the function

```
cl_int clReleaseSemaphoreKHR(
    cl_semaphore_khr sema_object);
```

sema_object specifies the semaphore object to be released.

The *sema_object* reference count is decremented.

clReleaseSemaphoreKHR returns CL_SUCCESS if the function is executed successfully. Otherwise, it returns one of the following errors:

- CL_INVALID_SEMAPHORE_KHR if *sema_object* is not a valid semaphore object.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

After the reference count becomes zero and commands queued for execution on a command-queue(s) that use *sema_object* have finished, the semaphore object is deleted. Using this function to release a reference that was not obtained by creating the object via **clCreateSemaphoreWithPropertiesKHR** or by calling **clRetainSemaphoreKHR** causes undefined behavior.

To retain a semaphore object, call the function

```
cl_int clRetainSemaphoreKHR(
    cl_semaphore_khr sema_object);
```

sema_object specifies the semaphore object to be retained.

increments the reference count of *sema_object*.

clRetainSemaphoreKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns one of the following errors:

- `CL_INVALID_SEMAPHORE_KHR` if *sema_object* is not a valid semaphore object.
- `CL_OUT_OF_RESOURCES` if there is a failure to allocate resources required by the OpenCL implementation on the device.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

43.8. Sample Code

1. Example for semaphore creation in a single device context

```
// Get cl_devices of the platform.
clGetDeviceIDs(..., &devices, &deviceCount);

// Create cl_context with just first device
context = clCreateContext(..., 1, devices, ...);

// Create clSema of type cl_semaphore_khr usable on single device in the context

cl_semaphore_properties_khr sema_props[] =
        {(cl_semaphore_properties_khr)CL_SEMAPHORE_TYPE_KHR,
         (cl_semaphore_properties_khr)CL_SEMAPHORE_TYPE_BINARY_KHR,
          0};

int errcode_ret = 0;

cl_semaphore_khr clSema = clCreateSemaphoreWithPropertiesKHR(context,
                                                              sema_props,
                                                              &errcode_ret);
```

2. Example for semaphore creation for a single device in a multi-device context

```
// Get cl_devices of the platform.
clGetDeviceIDs(..., &devices, &deviceCount);

// Create cl_context with first two devices
clCreateContext(..., 2, devices, ...);

// Create clSema of type cl_semaphore_khr usable only on device 0
cl_semaphore_properties_khr sema_props[] = {
    (cl_semaphore_properties_khr)CL_SEMAPHORE_TYPE_KHR,
    (cl_semaphore_properties_khr)CL_SEMAPHORE_TYPE_BINARY_KHR,
    (cl_semaphore_properties_khr)CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR,
    (cl_semaphore_properties_khr)devices[0],
    CL_SEMAPHORE_DEVICE_HANDLE_LIST_END_KHR,
    0
```

```
};

int errcode_ret = 0;

cl_semaphore_khr clSema = clCreateSemaphoreWithPropertiesKHR(context,
                                                             sema_props,
                                                             &errcode_ret);
```

3. Example for synchronization using Wait and Signal

```
// clSema is created using clCreateSemaphoreWithPropertiesKHR
// using one of the examples for semaphore creation.

cl_semaphore_khr clSema = clCreateSemaphoreWithPropertiesKHR(context,
                                                             sema_props,
                                                             &errcode_ret);

// Start the main loop

while (true) {
    // (not shown) Signal the semaphore from other work

    // Wait for the semaphore in OpenCL
    // by calling clEnqueueWaitSemaphoresKHR on 'clSema'
    clEnqueueWaitSemaphoresKHR(/*command_queue*/           command_queue,
                               /*num_sema_objects*/        1,
                               /*sema_objects*/            &clSema,
                               /*sema_payload_list*/       NULL,
                               /*num_events_in_wait_list*/ 0,
                               /*event_wait_list*/         NULL,
                               /*event*/                   NULL);

    // Launch kernel that accesses extMem
    clEnqueueNDRangeKernel(command_queue, ...);

    // Signal the semaphore in OpenCL
    clEnqueueSignalSemaphoresKHR(/*command_queue*/           command_queue,
                                 /*num_sema_objects*/        1,
                                 /*sema_objects*/            &clSema,
                                 /*sema_payload_list*/       NULL,
                                 /*num_events_in_wait_list*/ 0,
                                 /*event_wait_list*/         NULL,
                                 /*event*/                   NULL);

    // (not shown) Launch other work that waits on 'clSema'
}
```

4. Example for **clGetSemaphoreInfoKHR**

```
// clSema is created using clCreateSemaphoreWithPropertiesKHR
// using one of the examples for semaphore creation.

cl_semaphore_khr clSema = clCreateSemaphoreWithPropertiesKHR(context,
                                                              sema_props,
                                                              &errcode_ret);

// Start the main rendering loop

while (true) {
    // (not shown) Signal the semaphore from other work

    // Wait for the semaphore in OpenCL, by calling clEnqueueWaitSemaphoresKHR on
'clSema'
    clEnqueueWaitSemaphoresKHR(/*command_queue*/              command_queue,
                               /*num_sema_objects*/           1,
                               /*sema_objects*/               &clSema,
                               /*sema_payload_list*/          NULL,
                               /*num_events_in_wait_list*/    0,
                               /*event_wait_list*/            NULL,
                               /*event*/                      NULL);

    // Launch kernel in OpenCL
    clEnqueueNDRangeKernel(command_queue, ...);

    // Signal the semaphore in OpenCL
    clEnqueueSignalSemaphoresKHR(/*command_queue*/              command_queue,
                                 /*num_sema_objects*/           1,
                                 /*sema_objects*/               &clSema,
                                 /*sema_payload_list*/          NULL,
                                 /*num_events_in_wait_list*/    0,
                                 /*event_wait_list*/            NULL,
                                 /*event*/                      NULL);

    // Query type of clSema
    clGetSemaphoreInfoKHR(/*sema_object*/          clSema,
                          /*param_name*/           CL_SEMAPHORE_TYPE_KHR,
                          /*param_value_size*/     sizeof(cl_semaphore_type_khr),
                          /*param_value*/          &clSemaType,
                          /*param_value_ret_size*/ &clSemaTypeSize);

    if (clSemaType == CL_SEMAPHORE_TYPE_BINARY_KHR) {
        // Do something
    }
    else {
        // Do something else
    }
    // (not shown) Launch other work that waits on 'clSema'
}
```

[1] The reference count returned should be considered immediately stale. It is unsuitable for general use in applications. This feature is provided for identifying memory leaks.

Chapter 44. External Semaphores (Provisional)

cl_khr_semaphore introduced semaphores as a new type along with a set of APIs for create, release, retain, wait and signal operations on it. This extension defines APIs and mechanisms to share semaphores created in an external API by importing into and exporting from OpenCL.

This extension defines:

- New attributes that can be passed as part of cl_semaphore_properties_khr for specifying properties of external semaphores to be imported or exported.
- New attributes that can be passed as part of cl_semaphore_info_khr for specifying properties of external semaphores to be exported.
- An extension to **clCreateSemaphoreWithPropertiesKHR** to accept external semaphore properties allowing to import or export an external semaphore into or from OpenCL.
- Semaphore handle types required for importing and exporting semaphores.
- Modifications to Wait and Signal API behavior when dealing with external semaphores created from different handle types.
- API query exportable semaphores handles using specified handle type.

Other related extensions define specific external semaphores that may be imported into or exported from OpenCL.

44.1. General Information

44.1.1. Name Strings

cl_khr_external_semaphore
cl_khr_external_semaphore_dx_fence
cl_khr_external_semaphore_opaque_fd
cl_khr_external_semaphore_sync_fd
cl_khr_external_semaphore_win32

44.1.2. Version History

Date	Version	Description
2021-09-10	0.9.0	Initial version (provisional).
2023-11-16	0.9.1	Added CL_SEMAPHORE_EXPORTABLE_KHR.
2023-11-21	0.9.2	Added re-import function call to cl_khr_external_semaphore_sync_fd

This is a provisional OpenCL extension specification that has been Ratified under the Khronos Intellectual Property Framework. It is being made publicly available

> as a provisional extension to enable review and feedback from the community. While it is a provisional extension features may be added, removed, or changed in non-backward compatible ways.
>
> If you have feedback please create an issue on: https://github.com/KhronosGroup/OpenCL-Docs/

44.1.3. Dependencies

This extension is written against the OpenCL Specification Version 3.0.8.

This extension requires OpenCL 1.2.

The `cl_khr_semaphore` extension is required as it defines semaphore objects as well as for wait and signal operations on semaphores.

For OpenCL to be able to import external semaphores from other APIs using this extension, the other API is required to provide below mechanisms:

- Ability to export semaphore handles
- Ability to query semaphore handle in the form of one of the handle type supported by OpenCL.

The other APIs that want to use semaphore exported by OpenCL using this extension are required to provide below mechanism:

- Ability to import semaphore handles using handle types exported by OpenCL.

44.1.4. Contributors

Ajit Hakke-Patil, NVIDIA
Amit Rao, NVIDIA
Balaji Calidas, QUALCOMM
Ben Ashbaugh, INTEL
Carsten Rohde, NVIDIA
Christoph Kubisch, NVIDIA
Debalina Bhattacharjee, NVIDIA
Faith Ekstrand, INTEL
James Jones, NVIDIA
Jeremy Kemp, IMAGINATION
Joshua Kelly, QUALCOMM
Karthik Raghavan Ravi, NVIDIA
Kedar Patil, NVIDIA
Kevin Petit, ARM
Nikhil Joshi, NVIDIA
Sharan Ashwathnarayan, NVIDIA
Vivek Kini, NVIDIA

44.2. New Types

```
typedef cl_uint cl_external_semaphore_handle_type_khr;
```

The `cl_khr_external_semaphore_sync_fd` extension adds:

```
typedef cl_properties cl_semaphore_reimport_properties_khr;
```

44.3. New API Functions

```
cl_int clGetSemaphoreHandleForTypeKHR(
    cl_semaphore_khr sema_object,
    cl_device_id device,
    cl_external_semaphore_handle_type_khr handle_type,
    size_t handle_size,
    void *handle_ptr,
    size_t *handle_size_ret);
```

The `cl_khr_external_semaphore_sync_fd` extension adds:

```
cl_int clReImportSemaphoreSyncFdKHR(
    cl_semaphore_khr sema_object,
    cl_semaphore_reimport_properties_khr *reimport_props,
    int fd);
```

44.4. New API Enums

Accepted value for the *param_name* parameter to **clGetPlatformInfo** to query external semaphore handle types that may be imported or exported by all devices in an OpenCL platform:

```
CL_PLATFORM_SEMAPHORE_IMPORT_HANDLE_TYPES_KHR                   0x2037
CL_PLATFORM_SEMAPHORE_EXPORT_HANDLE_TYPES_KHR                   0x2038
```

Accepted value for the *param_name* parameter to **clGetDeviceInfo** to query external semaphore handle types that may be imported or exported by an OpenCL device:

```
CL_DEVICE_SEMAPHORE_IMPORT_HANDLE_TYPES_KHR                     0x204D
CL_DEVICE_SEMAPHORE_EXPORT_HANDLE_TYPES_KHR                     0x204E
```

Following new attributes can be passed as part of `cl_semaphore_properties_khr` and `cl_semaphore_info_khr`:

```
CL_SEMAPHORE_EXPORT_HANDLE_TYPES_KHR                        0x203F
CL_SEMAPHORE_EXPORT_HANDLE_TYPES_LIST_END_KHR               0
```

The following new attribute that can be passed as part of `cl_semaphore_info_khr`:

```
CL_SEMAPHORE_EXPORTABLE_KHR                                 0x2054
```

External semaphore handle type added by `cl_khr_external_semaphore_dx_fence`:

```
CL_SEMAPHORE_HANDLE_D3D12_FENCE_KHR               0x2059
```

External semaphore handle type added by `cl_khr_external_semaphore_opaque_fd`:

```
CL_SEMAPHORE_HANDLE_OPAQUE_FD_KHR                 0x2055
```

External semaphore handle type added by `cl_khr_external_semaphore_sync_fd`:

```
CL_SEMAPHORE_HANDLE_SYNC_FD_KHR                   0x2058
```

External semaphore handle types added by `cl_khr_external_semaphore_win32`:

```
CL_SEMAPHORE_HANDLE_OPAQUE_WIN32_KHR              0x2056
CL_SEMAPHORE_HANDLE_OPAQUE_WIN32_KMT_KHR          0x2057
```

44.5. Modifications to existing APIs added by this spec

Following new enums are added to the list of supported *param_names* by **clGetPlatformInfo**:

Table 58. List of supported param_names by **clGetPlatformInfo**

Platform Info	Return Type	Description
`CL_PLATFORM_SEMAPHORE_IMPORT_HANDLE_TYPES_KHR`	`cl_external_semaphore_handle_type_khr`[]	Returns the list of importable external semaphore handle types supported by all devices in *platform*. This size of this query may be 0 if no importable external semaphore handle types are supported by all devices in *platform*.

Platform Info	Return Type	Description
`CL_PLATFORM_SEMAPHORE_EXPORT_HANDLE_TYPES_KHR`	`cl_external_semaphore_handle_type_khr`[]	Returns the list of exportable external semaphore handle types supported by all devices in the platform. This size of this query may be 0 if no exportable external semaphore handle types are supported by all devices in *platform*.

clGetPlatformInfo when called with *param_name* `CL_PLATFORM_SEMAPHORE_IMPORT_HANDLE_TYPES_KHR` returns a common list of external semaphore handle types supported for importing by all devices in the platform.

clGetPlatformInfo when called with *param_name* `CL_PLATFORM_SEMAPHORE_EXPORT_HANDLE_TYPES_KHR` returns a common list of external semaphore handle types supported for exporting by all devices in the platform.

Following new enums are added to the list of supported *param_names* by **clGetDeviceInfo**:

Table 59. List of supported param_names by **clGetDeviceInfo**

Device Info	Return Type	Description
`CL_DEVICE_SEMAPHORE_IMPORT_HANDLE_TYPES_KHR`	`cl_external_semaphore_handle_type_khr`[]	Returns the list of importable external semaphore handle types supported by *device*. This size of this query may be 0 indicating that the device does not support importing semaphores.
`CL_DEVICE_SEMAPHORE_EXPORT_HANDLE_TYPES_KHR`	`cl_external_semaphore_handle_type_khr`[]	Returns the list of exportable external semaphore handle types supported by *device*. This size of this query may be 0 indicating that the device does not support exporting semaphores.

clGetDeviceInfo when called with *param_name* `CL_DEVICE_SEMAPHORE_IMPORT_HANDLE_TYPES_KHR` returns a list of external semaphore handle types supported for importing.

clGetDeviceInfo when called with *param_name* `CL_DEVICE_SEMAPHORE_EXPORT_HANDLE_TYPES_KHR` returns a list of external semaphore handle types supported for exporting.

One of the above two queries `CL_DEVICE_SEMAPHORE_IMPORT_HANDLE_TYPES_KHR` and `CL_DEVICE_SEMAPHORE_EXPORT_HANDLE_TYPES_KHR` must return a non-empty list indicating support for at least one of the valid semaphore handles types either for import or for export or both.

Following new properties are added to the list of possible supported properties by **clCreateSemaphoreWithPropertiesKHR**:

Table 60. List of supported semaphore creation properties by **clCreateSemaphoreWithPropertiesKHR**

Semaphore Property	Property Value	Description
CL_SEMAPHORE_EXPORT_HANDLE_TYPES_KHR	cl_external_semaphore_handle_type_khr[]	Specifies the list of semaphore handle type properties terminated with CL_SEMAPHORE_EXPORT_HANDLE_TYPES_LIST_END_KHR that can be used to export the semaphore being created.

Add to the list of error conditions for **clCreateSemaphoreWithPropertiesKHR**:

- CL_INVALID_DEVICE if one or more devices identified by properties CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR can not import the requested external semaphore handle type.
- CL_INVALID_VALUE if more than one semaphore handle type is specified in the CL_SEMAPHORE_EXPORT_HANDLE_TYPES_KHR list.
- CL_INVALID_OPERATION If *props_list* specifies a cl_external_semaphore_handle_type_khr followed by a handle as well as CL_SEMAPHORE_EXPORT_HANDLE_TYPES_KHR. Exporting a semaphore handle from a semaphore that was created by importing an external semaphore handle is not permitted.

Add to the list of supported *param_names* by **clGetSemaphoreInfoKHR**:

Table 61. List of supported param_names by **clGetSemaphoreInfoKHR**

Semaphore Info	Return Type	Description
CL_SEMAPHORE_EXPORT_HANDLE_TYPES_KHR	cl_external_semaphore_handle_type_khr[]	Returns the list of external semaphore handle types that may be used for exporting. The size of this query may be 0 indicating that this semaphore does not support any handle types for exporting.
CL_SEMAPHORE_EXPORTABLE_KHR	cl_bool	Returns CL_TRUE if the semaphore is exportable and CL_FALSE otherwise.

44.6. Exporting semaphore external handles

To export an external handle from a semaphore, call the function

```
cl_int clGetSemaphoreHandleForTypeKHR(
    cl_semaphore_khr sema_object,
    cl_device_id device,
    cl_external_semaphore_handle_type_khr handle_type,
    size_t handle_size,
    void* handle_ptr,
    size_t* handle_size_ret);
```

sema_object specifies a valid semaphore object with exportable properties.

device specifies a valid device for which a semaphore handle is being requested.

handle_type specifies the type of semaphore handle that should be returned for this exportable *sema_object* and must be one of the values specified when *sema_object* was created.

handle_size specifies the size of memory pointed by *handle_ptr*.

handle_ptr is a pointer to memory where the exported external handle is returned. If *param_value* is `NULL`, it is ignored.

handle_size_ret returns the actual size in bytes for the external handle. If *handle_size_ret* is `NULL`, it is ignored.

clGetSemaphoreHandleForTypeKHR returns `CL_SUCCESS` if the semaphore handle is queried successfully. Otherwise, it returns one of the following errors:

- `CL_INVALID_SEMAPHORE_KHR`
  - if *sema_object* is not a valid semaphore
  - if *sema_object* is not exportable
- `CL_INVALID_DEVICE`
  - if *device* is not a valid device, or
  - if *sema_object* belongs to a context that is not associated with *device*, or
  - if *sema_object* can not be shared with *device*.
- `CL_INVALID_VALUE` if the requested external semaphore handle type was not specified when *sema_object* was created.
- `CL_INVALID_VALUE` if *handle_size* is less than the size needed to store the returned handle.
- `CL_OUT_OF_RESOURCES` if there is a failure to allocate resources required by the OpenCL implementation on the device.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

44.7. Importing semaphore external handles

Applications can import a semaphore payload into an existing semaphore using an external semaphore handle. The effects of the import operation will be either temporary or permanent, as specified by the application. If the import is temporary, the implementation must restore the semaphore to its prior permanent state after submitting the next semaphore wait operation. Performing a subsequent temporary import on a semaphore before performing a semaphore wait has no effect on this requirement; the next wait submitted on the semaphore must still restore its last permanent state. A permanent payload import behaves as if the target semaphore was destroyed, and a new semaphore was created with the same handle but the imported payload. Because importing a semaphore payload temporarily or permanently detaches the existing payload from a semaphore, similar usage restrictions to those applied to **clReleaseSemaphoreKHR** are applied to any command that imports a semaphore payload. Which of these import types is used is referred to as the import operation's permanence. Each handle type supports either one or both types of permanence.

The implementation must perform the import operation by either referencing or copying the payload referred to by the specified external semaphore handle, depending on the handle's type. The import method used is referred to as the handle type's transference. When using handle types with reference transference, importing a payload to a semaphore adds the semaphore to the set of all semaphores sharing that payload. This set includes the semaphore from which the payload was exported. Semaphore signaling and waiting operations performed on any semaphore in the set must behave as if the set were a single semaphore. Importing a payload using handle types with copy transference creates a duplicate copy of the payload at the time of import, but makes no further reference to it. Semaphore signaling and waiting operations performed on the target of copy imports must not affect any other semaphore or payload.

Export operations have the same transference as the specified handle type's import operations. Additionally, exporting a semaphore payload to a handle with copy transference has the same side effects on the source semaphore's payload as executing a semaphore wait operation. If the semaphore was using a temporarily imported payload, the semaphore's prior permanent payload will be restored.

Please refer to handle specific specifications for more details on transference and permanence requirements specific to handle type.

44.8. Descriptions of External Semaphore Handle Types

This section describes the external semaphore handle types that are added by related extensions.

Applications can import the same semaphore payload into multiple OpenCL contexts, into the same context from which it was exported, and multiple times into a given OpenCL context. In all cases, each import operation must create a distinct semaphore object.

44.8.1. File Descriptor Handle Types

The `cl_khr_external_semaphore_opaque_fd` extension extends `cl_external_semaphore_handle_type_khr` to support the following new types of handles, and adds as a property that may be specified when creating a semaphore from an external handle:

- `CL_SEMAPHORE_HANDLE_OPAQUE_FD_KHR` specifies a POSIX file descriptor handle that has only limited valid usage outside of OpenCL and other compatible APIs. It must be compatible with the POSIX system calls dup, dup2, close, and the non-standard system call dup3. Additionally, it must be transportable over a socket using an SCM_RIGHTS control message. It owns a reference to the underlying synchronization primitive represented by its semaphore object.

Transference and permanence properties for handle types added by `cl_khr_external_semaphore_opaque_fd`:

Table 62. Transference and Permanence Properties for `cl_khr_external_semaphore_opaque_fd` handles

Handle Type	Transference	Permanence
`CL_SEMAPHORE_HANDLE_OPAQUE_FD_KHR`	Reference	Temporary, Permanent

The cl_khr_external_semaphore_sync_fd extension extends cl_external_semaphore_handle_type_khr to support the following new types of handles, and adds as a property that may be specified when creating a semaphore from an external handle:

- CL_SEMAPHORE_HANDLE_SYNC_FD_KHR specifies a POSIX file descriptor handle to a Linux Sync File or Android Fence object. It can be used with any native API accepting a valid sync file or fence as input. It owns a reference to the underlying synchronization primitive associated with the file descriptor. Implementations which support importing this handle type must accept any type of sync or fence FD supported by the native system they are running on.

The special value -1 for fd is treated like a valid sync file descriptor referring to an object that has already signaled. The import operation will succeed and the semaphore will have a temporarily imported payload as if a valid file descriptor had been provided.

Note: This special behavior for importing an invalid sync file descriptor allows easier interoperability with other system APIs which use the convention that an invalid sync file descriptor represents work that has already completed and does not need to be waited for. It is consistent with the option for implementations to return a -1 file descriptor when exporting a CL_SEMAPHORE_HANDLE_SYNC_FD_KHR from a cl_semaphore_khr which is signaled.

Transference and permanence properties for handle types added by cl_khr_external_semaphore_sync_fd:

Table 63. Transference and Permanence Properties for cl_khr_external_semaphore_sync_fd *handles*

Handle Type	Transference	Permanence
CL_SEMAPHORE_HANDLE_SYNC_FD_KHR	Copy	Temporary

For these extensions, importing a semaphore payload from a file descriptor transfers ownership of the file descriptor from the application to the OpenCL implementation. The application must not perform any operations on the file descriptor after a successful import.

A handle of type CL_SEMAPHORE_HANDLE_SYNC_FD_KHR may be re-imported into an existing semaphore using **clReImportSemaphoreSyncFdKHR**:

```
cl_int clReImportSemaphoreSyncFdKHR(
    cl_semaphore_khr sema_object,
    cl_semaphore_reimport_properties_khr* reimport_props,
    int fd);
```

sema_object specifies a valid semaphore object with importable properties.

reimport_props Must be NULL. Reserved for future use.

fd external file descriptor handle to import

Calling **clReImportSemaphoreSyncFdKHR** is equivalent to destroying *sema_object* and re-creating it with the original *sema_props* from **clCreateSemaphoreWithPropertiesKHR**, except a handle specified by *fd* will be imported. The semaphore *sema_object* must have originally imported an external handle of type CL_SEMAPHORE_HANDLE_SYNC_FD_KHR.

- CL_INVALID_SEMAPHORE_KHR
  - if *sema_object* is not a valid semaphore
- CL_INVALID_SEMAPHORE_KHR if a CL_SEMAPHORE_HANDLE_SYNC_FD_KHR handle was not imported when *sema_object* was created.
- CL_INVALID_VALUE if *fd* is invalid.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.

44.8.2. NT Handle Types

The cl_khr_external_semaphore_dx_fence extension extends cl_external_semaphore_handle_type_khr to support the following new types of handles, and adds as a property that may be specified when creating a semaphore from an external handle:

- CL_SEMAPHORE_HANDLE_D3D12_FENCE_KHR specifies an NT handle returned by ID3D12Device::CreateSharedHandle referring to a Direct3D 12 fence, or ID3D11Device5::CreateFence referring to a Direct3D 11 fence. It owns a reference to the underlying synchronization primitive associated with the Direct3D fence.

When waiting on semaphores using **clEnqueueWaitSemaphoresKHR** or signaling semaphores using **clEnqueueSignalSemaphoresKHR**, the semaphore payload must be provided for semaphores created from CL_SEMAPHORE_HANDLE_D3D12_FENCE_KHR.

- If *sema_objects* list has a mix of semaphores obtained from CL_SEMAPHORE_HANDLE_D3D12_FENCE_KHR and other handle types, then the *sema_payload_list* should point to a list of *num_sema_objects* payload values for each semaphore in *sema_objects*. However, the payload values corresponding to semaphores with type CL_SEMAPHORE_TYPE_BINARY_KHR can be set to 0 or will be ignored.

clEnqueueWaitSemaphoresKHR and **clEnqueueSignalSemaphoresKHR** may return CL_INVALID_VALUE if *sema_objects* list has one or more semaphores obtained from CL_SEMAPHORE_HANDLE_D3D12_FENCE_KHR and *sema_payload_list* is NULL.

Transference and permanence properties for handle types added by cl_khr_external_semaphore_dx_fence:

Table 64. Transference and Permanence Properties for cl_khr_external_semaphore_dx_fence *handles*

Handle Type	Transference	Permanence
CL_SEMAPHORE_HANDLE_D3D12_FENCE_KHR	Reference	Temporary, Permanent

The cl_khr_external_semaphore_win32 extension extends cl_external_semaphore_handle_type_khr to support the following new types of handles, and adds as a property that may be specified when creating a semaphore from an external handle:

- `CL_SEMAPHORE_HANDLE_OPAQUE_WIN32_KHR` specifies an NT handle that has only limited valid usage outside of OpenCL and other compatible APIs. It must be compatible with the functions DuplicateHandle, CloseHandle, CompareObjectHandles, GetHandleInformation, and SetHandleInformation. It owns a reference to the underlying synchronization primitive represented by its semaphore object.
- `CL_SEMAPHORE_HANDLE_OPAQUE_WIN32_KMT_KHR` specifies a global share handle that has only limited valid usage outside of OpenCL and other compatible APIs. It is not compatible with any native APIs. It does not own a reference to the underlying synchronization primitive represented by its semaphore object, and will therefore become invalid when all semaphore objects associated with it are destroyed.

Transference and permanence properties for handle types added by `cl_khr_external_semaphore_win32`:

Table 65. Transference and Permanence Properties for `cl_khr_external_semaphore_win32` *handles*

Handle Type	**Transference**	**Permanence**
`CL_SEMAPHORE_HANDLE_OPAQUE_WIN32_KHR`	Reference	Temporary, Permanent
`CL_SEMAPHORE_HANDLE_OPAQUE_WIN32_KMT_KHR`	Reference	Temporary, Permanent

For these extensions, importing a semaphore payload from Windows handles does not transfer ownership of the handle to the OpenCL implementation. For handle types defined as NT handles, the application must release ownership using the CloseHandle system call when the handle is no longer needed.

44.9. Sample Code

1. Example for importing a semaphore created by another API in OpenCL in a single-device context.

```
// Get cl_devices of the platform.
clGetDeviceIDs(..., &devices, &deviceCount);

// Create cl_context with just first device
clCreateContext(..., 1, devices, ...);

// Obtain fd/win32 or similar handle for external semaphore to be imported
// from the other API.
int fd = getFdForExternalSemaphore();

// Create clSema of type cl_semaphore_khr usable on the only available device
// assuming the semaphore was imported from the same device.

cl_semaphore_properties_khr sema_props[] =
```

```
        {(cl_semaphore_properties_khr)CL_SEMAPHORE_TYPE_KHR,
         (cl_semaphore_properties_khr)CL_SEMAPHORE_TYPE_BINARY_KHR,
         (cl_semaphore_properties_khr)CL_SEMAPHORE_HANDLE_OPAQUE_FD_KHR,
         (cl_semaphore_properties_khr)fd,
          0};


int errcode_ret = 0;
cl_semaphore_khr clSema = clCreateSemaphoreWithPropertiesKHR(context,
                                                              sema_props,
                                                              &errcode_ret);
```

2. Example for importing a semaphore created by another API in OpenCL in a multi-device context for single device usage.

```
// Get cl_devices of the platform.
clGetDeviceIDs(..., &devices, &deviceCount);

// Create cl_context with first two devices
clCreateContext(..., 2, devices, ...);

// Obtain fd/win32 or similar handle for external semaphore to be imported
// from the other API.
int fd = getFdForExternalSemaphore();

// Create clSema of type cl_semaphore_khr usable only on device 1
// assuming the semaphore was imported from the same device.
cl_semaphore_properties_khr sema_props[] = {
    (cl_semaphore_properties_khr)CL_SEMAPHORE_TYPE_KHR,
    (cl_semaphore_properties_khr)CL_SEMAPHORE_TYPE_BINARY_KHR,
    (cl_semaphore_properties_khr)CL_SEMAPHORE_HANDLE_OPAQUE_FD_KHR,
    (cl_semaphore_properties_khr)fd,
    (cl_semaphore_properties_khr)CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR,
    (cl_semaphore_properties_khr)devices[1],
    CL_SEMAPHORE_DEVICE_HANDLE_LIST_END_KHR,
    0
};

int errcode_ret = 0;
cl_semaphore_khr clSema = clCreateSemaphoreWithPropertiesKHR(context,
                                                              sema_props,
                                                              &errcode_ret);
```

3. Example for synchronization using a semaphore created by another API and imported in OpenCL

```
// Create clSema using one of the above examples of external semaphore creation.
```

```
int errcode_ret = 0;
cl_semaphore_khr clSema = clCreateSemaphoreWithPropertiesKHR(context,
                                                              sema_props,
                                                              &errcode_ret);

// Start the main loop

while (true) {
    // (not shown) Signal the semaphore from the other API

    // Wait for the semaphore in OpenCL
    clEnqueueWaitSemaphoresKHR(/*command_queue*/          command_queue,
                               /*num_sema_objects*/       1,
                               /*sema_objects*/           &clSema,
                               /*num_events_in_wait_list*/ 0,
                               /*event_wait_list*/        NULL,
                               /*event*/                  NULL);

    // Launch kernel
    clEnqueueNDRangeKernel(command_queue, ...);

    // Signal the semaphore in OpenCL
    clEnqueueSignalSemaphoresKHR(/*command_queue*/          command_queue,
                                 /*num_sema_objects*/       1,
                                 /*sema_objects*/           &clSema,
                                 /*num_events_in_wait_list*/ 0,
                                 /*event_wait_list*/        NULL,
                                 /*event*/                  NULL);

    // (not shown) Launch work in the other API that waits on 'clSema'

}
```

4. Example for synchronization using semaphore exported by OpenCL

```
// Get cl_devices of the platform.
clGetDeviceIDs(..., &devices, &deviceCount);

// Create cl_context with first two devices
clCreateContext(..., 2, devices, ...);

// Create clSema of type cl_semaphore_khr usable only on device 1
cl_semaphore_properties_khr sema_props[] = {
    (cl_semaphore_properties_khr)CL_SEMAPHORE_TYPE_KHR,
    (cl_semaphore_properties_khr)CL_SEMAPHORE_TYPE_BINARY_KHR,
    (cl_semaphore_properties_khr)CL_SEMAPHORE_EXPORT_HANDLE_TYPES_KHR,
    (cl_semaphore_properties_khr)CL_SEMAPHORE_HANDLE_OPAQUE_FD_KHR,
    CL_SEMAPHORE_EXPORT_HANDLE_TYPES_LIST_END_KHR,
    (cl_semaphore_properties_khr)CL_SEMAPHORE_DEVICE_HANDLE_LIST_KHR,
    (cl_semaphore_properties_khr)devices[1],
```

```
    CL_SEMAPHORE_DEVICE_HANDLE_LIST_END_KHR,
    0
};

int errcode_ret = 0;
cl_semaphore_khr clSema = clCreateSemaphoreWithPropertiesKHR(context,
                                                              sema_props,
                                                              &errcode_ret);

// Application queries handle-type and the exportable handle associated with the
semaphore.
clGetSemaphoreInfoKHR(clSema,
                      CL_SEMAPHORE_EXPORT_HANDLE_TYPES_KHR,
                      sizeof(cl_external_semaphore_handle_type_khr),
                      &handle_type,
                      &handle_type_size);

// The other API or process can use the exported semaphore handle
// to import
int fd = -1;
if (handle_type == CL_SEMAPHORE_HANDLE_OPAQUE_FD_KHR) {
    clGetSemaphoreHandleForTypeKHR(clSema,
                                   device,
                                   CL_SEMAPHORE_HANDLE_OPAQUE_FD_KHR,
                                   sizeof(int),
                                   &fd,
                                   NULL);
}

// Start the main rendering loop

while (true) {
    // (not shown) Signal the semaphore from the other API

    // Wait for the semaphore in OpenCL
    clEnqueueWaitSemaphoresKHR(/*command_queue*/           command_queue,
                               /*num_sema_objects*/        1,
                               /*sema_objects*/            &clSema,
                               /*num_events_in_wait_list*/ 0,
                               /*event_wait_list*/         NULL,
                               /*event*/                   NULL);

    // Launch kernel
    clEnqueueNDRangeKernel(command_queue, ...);

    // Signal the semaphore in OpenCL
    clEnqueueSignalSemaphoresKHR(/*command_queue*/           command_queue,
                                 /*num_sema_objects*/        1,
                                 /*sema_objects*/            &clSema,
                                 /*num_events_in_wait_list*/ 0,
                                 /*event_wait_list*/         NULL,
```

```
                                          /*event*/                      NULL);

    // (not shown) Launch work in the other API that waits on 'clSema'
}
```

Chapter 45. External Memory (Provisional)

This extension defines a generic mechanism to share buffer and image objects between OpenCL and many other APIs.

In particular, the `cl_khr_external_memory` extension defines:

- Optional properties to import external memory exported by other APIs into OpenCL for a set of devices.
- Routines to explicitly hand off memory ownership between OpenCL and other APIs.

Other related extensions define specific external memory types that may be imported into OpenCL.

45.1. General Information

45.1.1. Name Strings

`cl_khr_external_memory`
`cl_khr_external_memory_dma_buf`
`cl_khr_external_memory_dx`
`cl_khr_external_memory_opaque_fd`
`cl_khr_external_memory_win32`

45.1.2. Version History

Date	Version	Description
2021-09-10	0.9.0	Initial version (provisional).
2023-05-04	0.9.1	Clarified device handle list enum cannot be specified without an external memory handle (provisional).
2023-08-01	0.9.2	Changed device handle list enum to the memory-specific `CL_MEM_DEVICE_HANDLE_LIST_KHR` (provisional).
2023-08-29	0.9.3	Added query for {CL_DEVICE_EXTERNAL_MEMORY_IMPORT_ASSUME_LINEAR_HANDLE_TYPES_KHR} (provisional).

This is a provisional OpenCL extension specification that has been Ratified under the Khronos Intellectual Property Framework. It is being made publicly available as a provisional extension to enable review and feedback from the community. While it is a provisional extension features may be added, removed, or changed in non-backward compatible ways.

If you have feedback please create an issue on: https://github.com/KhronosGroup/OpenCL-Docs/

45.1.3. Dependencies

This extension is written against the OpenCL Specification Version 3.0.8.

Because this extension adds new properties for **clCreateBufferWithProperties** and **clCreateImageWithProperties** this extension requires OpenCL 3.0.

45.1.4. Contributors

Ajit Hakke-Patil, NVIDIA
Amit Rao, NVIDIA
Balaji Calidas, QUALCOMM
Ben Ashbaugh, INTEL
Carsten Rohde, NVIDIA
Christoph Kubisch, NVIDIA
Debalina Bhattacharjee, NVIDIA
Faith Ekstrand, INTEL
James Jones, NVIDIA
Jeremy Kemp, IMAGINATION
Joshua Kelly, QUALCOMM
Karthik Raghavan Ravi, NVIDIA
Kedar Patil, NVIDIA
Kevin Petit, ARM
Nikhil Joshi, NVIDIA
Sharan Ashwathnarayan, NVIDIA
Vivek Kini, NVIDIA

45.2. New Types

```
typedef cl_uint cl_external_memory_handle_type_khr;
```

45.3. New API Functions

```
cl_int clEnqueueAcquireExternalMemObjectsKHR(
    cl_command_queue command_queue,
    cl_uint num_mem_objects,
    const cl_mem *mem_objects,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list,
    cl_event *event);

cl_int clEnqueueReleaseExternalMemObjectsKHR(
    cl_command_queue command_queue,
    cl_uint num_mem_objects,
    const cl_mem *mem_objects,
    cl_uint num_events_in_wait_list,
```

```
    const cl_event *event_wait_list,
    cl_event *event);
```

45.4. New API Enums

Accepted value for the *param_name* parameter to **clGetPlatformInfo** to query external memory handle types that may be imported by all devices in an OpenCL platform:

```
CL_PLATFORM_EXTERNAL_MEMORY_IMPORT_HANDLE_TYPES_KHR                 0x2044
```

Accepted value for the *param_name* parameter to **clGetDeviceInfo** to query external memory handle types that may be imported by an OpenCL device:

```
CL_DEVICE_EXTERNAL_MEMORY_IMPORT_HANDLE_TYPES_KHR                           0x204F
CL_DEVICE_EXTERNAL_MEMORY_IMPORT_ASSUME_LINEAR_IMAGES_HANDLE_TYPES_KHR      0x2052
```

New properties accepted as *properties* to **clCreateBufferWithProperties** and **clCreateImageWithProperties**:

```
CL_MEM_DEVICE_HANDLE_LIST_KHR                                       0x2051
CL_MEM_DEVICE_HANDLE_LIST_END_KHR                                   0
```

New return values from **clGetEventInfo** when *param_name* is `CL_EVENT_COMMAND_TYPE`:

```
CL_COMMAND_ACQUIRE_EXTERNAL_MEM_OBJECTS_KHR                         0x2047
CL_COMMAND_RELEASE_EXTERNAL_MEM_OBJECTS_KHR                         0x2048
```

External memory handle type added by `cl_khr_external_memory_dma_buf`:

```
CL_EXTERNAL_MEMORY_HANDLE_DMA_BUF_KHR              0x2067
```

External memory handle types added by `cl_khr_external_memory_dx`:

```
CL_EXTERNAL_MEMORY_HANDLE_D3D11_TEXTURE_KHR        0x2063
CL_EXTERNAL_MEMORY_HANDLE_D3D11_TEXTURE_KMT_KHR    0x2064
CL_EXTERNAL_MEMORY_HANDLE_D3D12_HEAP_KHR           0x2065
CL_EXTERNAL_MEMORY_HANDLE_D3D12_RESOURCE_KHR       0x2066
```

External memory handle type added by `cl_khr_external_memory_opaque_fd`:

```
CL_EXTERNAL_MEMORY_HANDLE_OPAQUE_FD_KHR            0x2060
```

External memory handle types added by `cl_khr_external_memory_win32`:

```
CL_EXTERNAL_MEMORY_HANDLE_OPAQUE_WIN32_KHR          0x2061
CL_EXTERNAL_MEMORY_HANDLE_OPAQUE_WIN32_KMT_KHR      0x2062
```

45.5. Modifications to existing APIs added by this spec

Following new enums are added to the list of supported *param_names* by **clGetPlatformInfo**:

Table 66. List of supported param_names by clGetPlatformInfo

Platform Info	Return Type	Description
`CL_PLATFORM_EXTERNAL_MEMORY_IMPORT_HANDLE_TYPES_KHR`	`cl_external_memory_handle_type_khr`[]	Returns the list of importable external memory handle types supported by all devices in *platform*.

clGetPlatformInfo when called with *param_name* `CL_PLATFORM_EXTERNAL_MEMORY_IMPORT_HANDLE_TYPES_KHR` must return a common list of external memory handle types supported by all devices in the platform.

Following new enums are added to the list of supported *param_names* by **clGetDeviceInfo**:

Table 67. List of supported param_names by clGetDeviceInfo

Device Info	Return Type	Description
`CL_DEVICE_EXTERNAL_MEMORY_IMPORT_HANDLE_TYPES_KHR`	`cl_external_memory_handle_type_khr`[]	Returns the list of importable external memory handle types supported by *device*.
`CL_DEVICE_EXTERNAL_MEMORY_IMPORT_ASSUME_LINEAR_IMAGES_HANDLE_TYPES_KHR`	`cl_external_memory_handle_type_khr`[]	Returns the list of importable external memory handle types supported by *device*, that are assumed to apply linear layout to imported images when no other tiling information is provided.

clGetDeviceInfo when called with param_name `CL_DEVICE_EXTERNAL_MEMORY_IMPORT_HANDLE_TYPES_KHR` must return a non-empty list of external memory handle types for at least one of the devices in the platform.

clGetDeviceInfo when called with param_name `CL_DEVICE_EXTERNAL_MEMORY_IMPORT_ASSUME_LINEAR_IMAGES_HANDLE_TYPES_KHR` returns a list of external memory handle types that are assumed to have a linear memory layout when no other tiling information is provided. This list contains a subset of `CL_DEVICE_EXTERNAL_MEMORY_IMPORT_HANDLE_TYPES_KHR`. The returned list may be empty.

External memory handle types not in `CL_DEVICE_EXTERNAL_MEMORY_IMPORT_ASSUME_LINEAR_IMAGES_HANDLE_TYPES_KHR` may have any memory layout. The layout interpretation of images imported with these handle types is implementation defined.

Following new properties are added to the list of supported properties by

clCreateBufferWithProperties and **clCreateImageWithProperties**.

Table 68. List of supported buffer and image creation properties

Property	Property Value	Description
CL_MEM_DEVICE_HANDLE_LIST_KHR	cl_device_id[]	Specifies the list of OpenCL devices (terminated with CL_MEM_DEVICE_HANDLE_LIST_END_KHR) to associate with the external memory handle.

If CL_MEM_DEVICE_HANDLE_LIST_KHR is not specified as part of *properties*, the memory object created by **clCreateBufferWithProperties** or **clCreateImageWithProperties** is by default accessible to all devices in the *context*.

The properties used to create a buffer or image from an external memory handle are described by related extensions. When a buffer or image is created from an external memory handle, the *flags* used to specify usage information for the buffer or image must not include CL_MEM_USE_HOST_PTR, CL_MEM_ALLOC_HOST_PTR, or CL_MEM_COPY_HOST_PTR, and the *host_ptr* argument must be NULL. When images are created from an external memory handle, implementations may acquire information about image attributes such as format and layout at the time of creation. When such information is acquired at image creation time, it is used for the lifetime of the image object.

Add to the list of error conditions for **clCreateBufferWithProperties** and **clCreateImageWithProperties**:

- CL_INVALID_DEVICE
  - if a device identified by the property CL_MEM_DEVICE_HANDLE_LIST_KHR is not a valid device or is not associated with *context*, or
  - if a device identified by property CL_MEM_DEVICE_HANDLE_LIST_KHR cannot import the requested external memory object type, or
  - if CL_MEM_DEVICE_HANDLE_LIST_KHR is not specified as part of *properties* and one or more devices in *context* cannot import the requested external memory object type.
- CL_INVALID_VALUE
  - if *properties* includes a supported external memory handle and *flags* includes CL_MEM_USE_HOST_PTR, CL_MEM_ALLOC_HOST_PTR, or CL_MEM_COPY_HOST_PTR.
- CL_INVALID_HOST_PTR
  - if *properties* includes a supported external memory handle and *host_ptr* is not NULL.
- CL_INVALID_PROPERTY
  - if *properties* does not include a supported external memory handle and CL_MEM_DEVICE_HANDLE_LIST_KHR is specified as part of *properties*.

Add images created from an external memory handle to the description of image_row_pitch and image_slice_pitch for cl_image_desc:

- image_row_pitch is the scan-line pitch in bytes. The image_row_pitch must be zero if *host_ptr* is NULL, the image is not a 2D image created from a buffer, and the image is not an image created from an external memory handle. If image_row_pitch is zero and *host_ptr* is not NULL then the image row pitch is calculated as `image_width` × the size of an image element in bytes. If `image_row_pitch` is zero and the image is created from an external memory handle then the image row pitch is implementation-defined. The image row pitch must be ≥ `image_width` × the size of an image element in bytes and must be a multiple of the size of an image element in bytes. For a 2D image created from a buffer the image row pitch must also be a multiple of the maximum of the `CL_DEVICE_IMAGE_PITCH_ALIGNMENT` value for all devices in the context that support images.

- `image_slice_pitch` is the size in bytes of each 2D slice in a 3D image or the size in bytes of each image in a 1D or 2D image array. The `image_slice_pitch` must be zero if *host_ptr* is `NULL` and the image is not created from an external memory handle. If `image_slice_pitch` is zero and *host_ptr* is not `NULL` then the image slice pitch is calculated as the image row pitch × `image_height` for a 2D image array or a 3D image, and as the image row pitch for a 1D image array. If `image_slice_pitch` is zero and the image is created from an external memory handle then the image slice pitch is implementation-defined. The image slice pitch must be ≥ the image image row pitch × `image_height` for a 2D image array or a 3D image, must be ≥ the image row pitch for a 1D image array, and must be a multiple of the image row pitch.

45.6. Description of new types added by this spec

The following new APIs are added as part of this spec. The details of each are described below:

45.6.1. Acquiring and Releasing External Memory Objects

To enqueue a command to acquire OpenCL memory objects created from external memory handles, call the function

```
cl_int clEnqueueAcquireExternalMemObjectsKHR(
    cl_command_queue command_queue,
    cl_uint num_mem_objects,
    const cl_mem* mem_objects,
    cl_uint num_events_in_wait_list,
    const cl_event* event_wait_list,
    cl_event* event);
```

command_queue specifies a valid command-queue.

num_mem_objects specifies the number of memory objects to acquire.

mem_objects points to a list of valid memory objects.

num_events_in_wait_list specifies the number of events in *event_wait_list*.

event_wait_list points to the list of events that need to complete before **clEnqueueAcquireExternalMemObjectsKHR** can be executed. If *event_wait_list* is `NULL`, then **clEnqueueAcquireExternalMemObjectsKHR** does not explicitly wait on any event to complete. If *event_wait_list* is `NULL`, *num_events_in_wait_list* must be 0. If *event_wait_list* is not `NULL`, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points. The context associated with events in *event_wait_list* and that of *command_queue* must be the same.

event returns an event object that identifies this particular command and can be used to query or queue a wait for this particular command to complete. *event* can be `NULL` in which case it will not be possible for the application to query the status of this command or queue a wait for this command to complete.

Applications must acquire the memory objects that are created using external handles before they can be used by any OpenCL commands queued to a command-queue. Behavior is undefined if a memory object created from an external memory handle is used by an OpenCL command queued to a command-queue without being acquired. This is to guarantee that the state of the memory objects is up-to-date and they are accessible to OpenCL. See "Example with Acquire / Release" provided in Sample Code for more details on how to use this API.

If *num_mem_objects* is 0 and *mem_objects* is `NULL`, the command will trivially succeed after its event dependencies are satisfied and will update its completion event.

clEnqueueAcquireExternalMemObjectsKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns one of the following errors:

- `CL_INVALID_VALUE` if *num_mem_objects* is zero and *mem_objects* is not a `NULL` value or if *num_mem_objects* is greater than 0 and *mem_objects* is `NULL`.
- `CL_INVALID_MEM_OBJECT` if any of the memory objects in *mem_objects* is not a valid OpenCL memory object created using an external memory handle.
- `CL_INVALID_COMMAND_QUEUE`
  - if *command_queue* is not a valid command-queue, or
  - if device associated with *command_queue* is not one of the devices specified by `CL_MEM_DEVICE_HANDLE_LIST_KHR` at the time of creating one or more of *mem_objects*, or
  - if one or more of *mem_objects* belong to a context that does not contain a device associated with *command_queue*.
- `CL_INVALID_EVENT_WAIT_LIST`
  - if *event_wait_list* is `NULL` and *num_events_in_wait_list* is not 0, or
  - if *event_wait_list* is not `NULL` and *num_events_in_wait_list* is 0, or
  - if event objects in *event_wait_list* are not valid events.
- `CL_EXEC_STATUS_ERROR_FOR_EVENTS_IN_WAIT_LIST` if the execution status of any of the events in *event_wait_list* is a negative integer value.
- `CL_OUT_OF_RESOURCES` if there is a failure to allocate resources required by the OpenCL implementation on the device.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

To enqueue a command to release OpenCL memory objects created from external memory handles, call the function

```
cl_int clEnqueueReleaseExternalMemObjectsKHR(
    cl_command_queue command_queue,
    cl_uint num_mem_objects,
    const cl_mem* mem_objects,
    cl_uint num_events_in_wait_list,
    const cl_event* event_wait_list,
    cl_event* event);
```

command_queue specifies a valid command-queue.

num_mem_objects specifies the number of memory objects to release.

mem_objects points to a list of valid memory objects.

num_events_in_wait_list specifies the number of events in *event_wait_list*.

event_wait_list points to the list of events that need to complete before **clEnqueueReleaseExternalMemObjectsKHR** can be executed. If *event_wait_list* is `NULL`, then **clEnqueueReleaseExternalMemObjectsKHR** does not wait on any event to complete. If *event_wait_list* is `NULL`, *num_events_in_wait_list* must be 0. If *event_wait_list* is not `NULL`, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points. The context associated with events in *event_wait_list* and that of *command_queue* must be the same.

event returns an event object that identifies this particular command and can be used to query or queue a wait for this particular command to complete. *event* can be `NULL` in which case it will not be possible for the application to query the status of this command or queue a wait for this command to complete.

Applications must release the memory objects that are acquired using **clEnqueueReleaseExternalMemObjectsKHR** before using them through any commands in the other API. This is to guarantee that the state of memory objects is up-to-date and they are accessible to the other API. See "Example with Acquire / Release" provided in Sample Code for more details on how to use this API.

If *num_mem_objects* is 0 and *mem_objects* is `NULL`, the command will trivially succeed after its event dependencies are satisfied and will update its completion event.

clEnqueueReleaseExternalMemObjectsKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns one of the following errors:

- `CL_INVALID_VALUE` if *num_mem_objects* is zero and *mem_objects* is not a `NULL` value or if *num_mem_objects* is greater than 0 and *mem_objects* is `NULL`.
- `CL_INVALID_MEM_OBJECT` if any of the memory objects in *mem_objects* is not a valid OpenCL memory object created using an external memory handle.
- `CL_INVALID_COMMAND_QUEUE`
    - if *command_queue* is not a valid command-queue, or
    - if device associated with *command_queue* is not one of the devices specified by `CL_MEM_`

DEVICE_HANDLE_LIST_KHR at the time of creating one or more of *mem_objects*, or

  - if one or more of *mem_objects* belong to a context that does not contain a device associated with *command_queue*.

- CL_INVALID_EVENT_WAIT_LIST
  - if *event_wait_list* is NULL and *num_events_in_wait_list* is not 0, or
  - if *event_wait_list* is not NULL and *num_events_in_wait_list* is 0, or
  - if event objects in *event_wait_list* are not valid events.
- CL_EXEC_STATUS_ERROR_FOR_EVENTS_IN_WAIT_LIST if the execution status of any of the events in *event_wait_list* is a negative integer value.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

45.7. Descriptions of External Memory Handle Types

This section describes the external memory handle types that are added by related extensions.

Applications can import the same payload into multiple OpenCL contexts and multiple times into a given OpenCL context. In all cases, each import operation must create a distinct memory object.

45.7.1. File Descriptor Handle Types

The cl_khr_external_memory_opaque_fd extension extends cl_external_memory_handle_type_khr to support the following new types of handles, and adds as a property that may be specified when creating a buffer or an image memory object from an external handle:

- CL_EXTERNAL_MEMORY_HANDLE_OPAQUE_FD_KHR specifies a POSIX file descriptor handle that has only limited valid usage outside of OpenCL and other compatible APIs. It must be compatible with the POSIX system calls dup, dup2, close, and the non-standard system call dup3. Additionally, it must be transportable over a socket using an SCM_RIGHTS control message. It owns a reference to the underlying memory resource represented by its memory object.

The cl_khr_external_memory_dma_buf extension extends cl_external_memory_handle_type_khr to support the following types of handles, and adds as a property that may be specified when creating a buffer or an image memory object from an external handle:

- CL_EXTERNAL_MEMORY_HANDLE_DMA_BUF_KHR is a file descriptor for a Linux dma_buf. It owns a reference to the underlying memory resource represented by its memory object.

For these extensions, importing memory from a file descriptor transfers ownership of the file descriptor from the application to the OpenCL implementation. The application must not perform any operations on the file descriptor after a successful import. The imported memory object holds a reference to its payload.

45.7.2. NT Handle Types

The `cl_khr_external_memory_dx` extension extends `cl_external_memory_handle_type_khr` to support the following new types of handles, and adds as a property that may be specified when creating a buffer or an image memory object from an external handle:

- `CL_EXTERNAL_MEMORY_HANDLE_D3D11_TEXTURE_KHR` specifies an NT handle returned by IDXGIResource1::CreateSharedHandle referring to a Direct3D 10 or 11 texture resource. It owns a reference to the memory used by the Direct3D resource.
- `CL_EXTERNAL_MEMORY_HANDLE_D3D11_TEXTURE_KMT_KHR` specifies a global share handle returned by IDXGIResource::GetSharedHandle referring to a Direct3D 10 or 11 texture resource. It does not own a reference to the underlying Direct3D resource, and will therefore become invalid when all memory objects and Direct3D resources associated with it are destroyed.
- `CL_EXTERNAL_MEMORY_HANDLE_D3D12_HEAP_KHR` specifies an NT handle returned by ID3D12Device::CreateSharedHandle referring to a Direct3D 12 heap resource. It owns a reference to the resources used by the Direct3D heap.
- `CL_EXTERNAL_MEMORY_HANDLE_D3D12_RESOURCE_KHR` specifies an NT handle returned by ID3D12Device::CreateSharedHandle referring to a Direct3D 12 committed resource. It owns a reference to the memory used by the Direct3D resource.

The `cl_khr_external_memory_win32` extension extends `cl_external_memory_handle_type_khr` to support the following new types of handles, and adds as a property that may be specified when creating a buffer or an image memory object from an external handle:

- `CL_EXTERNAL_MEMORY_HANDLE_OPAQUE_WIN32_KHR` specifies an NT handle that has only limited valid usage outside of OpenCL and other compatible APIs. It must be compatible with the functions DuplicateHandle, CloseHandle, CompareObjectHandles, GetHandleInformation, and SetHandleInformation. It owns a reference to the underlying memory resource represented by its memory object.
- `CL_EXTERNAL_MEMORY_HANDLE_OPAQUE_WIN32_KMT_KHR` specifies a global share handle that has only limited valid usage outside of OpenCL and other compatible APIs. It is not compatible with any native APIs. It does not own a reference to the underlying memory resource represented by its memory object, and will therefore become invalid when all memory objects associated with it are destroyed.

For these extensions, importing memory object payloads from Windows handles does not transfer ownership of the handle to the OpenCL implementation. For handle types defined as NT handles, the application must release handle ownership using the CloseHandle system call when the handle is no longer needed. For handle types defined as NT handles, the imported memory object holds a reference to its payload.

Note: Non-NT handle import operations do not add a reference to their associated payload. If the original object owning the payload is destroyed, all resources and handles sharing that payload will become invalid.

45.8. Sample Code

1. Example for creating a CL buffer from an exported external buffer in a single device context.

```
// Get cl_devices of the platform.
clGetDeviceIDs(..., &devices, &deviceCount);

// Create cl_context with just first device
clCreateContext(..., 1, devices, ...);

// Obtain fd/win32 or similar handle for external memory to be imported
// from other API.
int fd = getFdForExternalMemory();

// Create extMemBuffer of type cl_mem from fd.
cl_mem_properties_khr extMemProperties[] =
{
    (cl_mem_properties_khr)CL_EXTERNAL_MEMORY_HANDLE_OPAQUE_FD_KHR,
    (cl_mem_properties_khr)fd,
    0
};

cl_mem extMemBuffer = clCreateBufferWithProperties(/*context*/          clContext,
                                                   /*properties*/
extMemProperties,
                                                   /*flags*/            0,
                                                   /*size*/             size,
                                                   /*host_ptr*/         NULL,
                                                   /*errcode_ret*/
&errcode_ret);
```

2. Example for creating a CL Image from an exported external Image for single device usage in a multi-device context

```
// Get cl_devices of the platform.
clGetDeviceIDs(..., &devices, &deviceCount);

// Create cl_context with first two devices
clCreateContext(..., 2, devices, ...);

// Create img of type cl_mem usable only on devices[0]

// Create img of type cl_mem.
// Obtain fd/win32 or similar handle for external memory to be imported
// from other API.
int fd = getFdForExternalMemory();

// Set cl_image_format based on external image info
cl_image_format clImgFormat = { };
```

```
clImageFormat.image_channel_order = CL_RGBA;
clImageFormat.image_channel_data_type = CL_UNORM_INT8;

// Set cl_image_desc based on external image info
size_t clImageFormatSize;
cl_image_desc image_desc = { };
image_desc.image_type = CL_MEM_OBJECT_IMAGE2D_ARRAY;
image_desc.image_width = width;
image_desc.image_height = height;
image_desc.image_depth = depth;
image_desc.image_array_size = num_slices;
image_desc.image_row_pitch = width * 8 * 4; // May need alignment
image_desc.image_slice_pitch = image_desc.image_row_pitch * height;
image_desc.num_mip_levels = 1;
image_desc.num_samples = 0;
image_desc.buffer = NULL;

cl_mem_properties_khr extMemProperties[] = {
    (cl_mem_properties_khr)CL_EXTERNAL_MEMORY_HANDLE_OPAQUE_FD_KHR,
    (cl_mem_properties_khr)fd,
    (cl_mem_properties_khr)CL_MEM_DEVICE_HANDLE_LIST_KHR,
    (cl_mem_properties_khr)devices[0],
    CL_MEM_DEVICE_HANDLE_LIST_END_KHR,
    0
};

cl_mem img = clCreateImageWithProperties(/*context*/        clContext,
                                         /*properties*/     extMemProperties,
                                         /*flags*/          0,
                                         /*image_format*/   &clImgFormat,
                                         /*image_desc*/     &image_desc,
                                         /*errcode_ret*/    &errcode_ret);

// Use clGetImageInfo to get cl_image_format details.
size_t clImageFormatSize;
clGetImageInfo(img,
               CL_IMAGE_FORMAT,
               sizeof(cl_image_format),
               &clImageFormat,
               &clImageFormatSize);
```

3. Example for synchronization using Wait and Signal

```
// Start the main rendering loop

// Create extSem of type cl_semaphore_khr using clCreateSemaphoreWithPropertiesKHR

// Create extMem of type cl_mem using clCreateBufferWithProperties or
clCreateImageWithProperties
```

```
while (true) {
    // (not shown) Signal the semaphore from the other API

    // Wait for the semaphore in OpenCL, by calling clEnqueueWaitSemaphoresKHR on
'extSem'
    clEnqueueWaitSemaphoresKHR(/*command_queue*/           command_queue,
                               /*num_sema_objects*/        1,
                               /*sema_objects*/            &extSem,
                               /*sema_payload_list*/       NULL,
                               /*num_events_in_wait_list*/ 0,
                               /*event_wait_list*/         NULL,
                               /*event*/                   NULL);

    // Launch kernel that accesses extMem
    clEnqueueNDRangeKernel(command_queue, ...);

    // Signal the semaphore in OpenCL
    clEnqueueSignalSemaphoresKHR(/*command_queue*/           command_queue,
                                 /*num_sema_objects*/        1,
                                 /*sema_objects*/            &extSem,
                                 /*sema_payload_list*/       NULL,
                                 /*num_events_in_wait_list*/ 0,
                                 /*event_wait_list*/         NULL,
                                 /*event*/                   NULL);

    // (not shown) Launch work in other API that waits on 'extSem'
}
```

4. Example with memory sharing using acquire/release

```
// Create extSem of type cl_semaphore_khr using
// clCreateSemaphoreWithPropertiesKHR with CL_SEMAPHORE_HANDLE_*_KHR.

// Create extMem1 and extMem2 of type cl_mem using clCreateBufferWithProperties
// or clCreateImageWithProperties

while (true) {
    // (not shown) Signal the semaphore from the other API. Wait for the
    // semaphore in OpenCL, by calling clEnqueueWaitForSemaphore on extSem
    clEnqueueWaitSemaphoresKHR(/*command_queue*/           cq1,
                               /*num_sema_objects*/        1,
                               /*sema_objects*/            &extSem,
                               /*sema_payload_list*/       NULL,
                               /*num_events_in_wait_list*/ 0,
                               /*event_wait_list*/         NULL,
                               /*event*/                   NULL);

    // Get explicit ownership of extMem1
    clEnqueueAcquireExternalMemObjectsKHR(/*command_queue*/           cq1,
                                          /*num_mem_objects*/         1,
```

```
                                                   /*mem_objects*/                 extMem1,
                                                   /*num_events_in_wait_list*/    0,
                                                   /*event_wait_list*/            NULL,
                                                   /*event*/                      NULL);

    // Launch kernel that accesses extMem1 on cq1 on cl_device1
    clEnqueueNDRangeKernel(cq1,  ..., &event1);

    // Launch kernel that accesses both extMem1 and extMem2 on cq2 on cl_device2
    // Migration of extMem1 and extMem2 handles through regular CL memory
    // migration.
    clEnqueueNDRangeKernel(cq2, ..., &event1, &event2);

    // Give up ownership of extMem1 before you signal the semaphore. Handle
    // memory migration here.
    clEnqueueReleaseExternalMemObjectsKHR(/*command_queue*/           cq2
                                          /*num_mem_objects*/         1,
                                          /*mem_objects*/             &extMem1,
                                          /*num_events_in_wait_list*/ 0,
                                          /*event_wait_list*/         NULL,
                                          /*event*/                   NULL);

    // Signal the semaphore from OpenCL
    clEnqueueSignalSemaphoresKHR(/*command_queue*/           cq2,
                                 /*num_sema_objects*/        1,
                                 /*sema_objects*/            &extSem,
                                 /*sema_payload_list*/       NULL,
                                 /*num_events_in_wait_list*/ 0,
                                 /*event_wait_list*/         NULL,
                                 /*event*/                   NULL);

    // (not shown) Launch work in other API that waits on 'extSem'
    // Other API accesses ext1, but not ext2 on device-1
}
```

45.9. Issues

1. How should the import of images that are created in external APIs with non-linear tiling be robustly handled?

   UNRESOLVED

Chapter 46. Command Buffers (Provisional)

This extension adds the ability to record and replay buffers of OpenCL commands.

46.1. General Information

46.1.1. Name Strings

`cl_khr_command_buffer`

46.1.2. Version History

Date	Version	Description
2021-11-10	0.9.0	First assigned version (provisional).
2022-08-24	0.9.1	Specify an error if a command-buffer is finalized multiple times (provisional).
2023-03-31	0.9.2	Introduce context query `CL_COMMAND_BUFFER_CONTEXT_KHR` (provisional).
2023-04-04	0.9.3	Remove Invalid command-buffer state (provisional).
2023-05-11	0.9.4	Add clCommandSVMMemcpyKHR and clCommandSVMMemFillKHR command entries (provisional).

This is a provisional OpenCL extension specification that has been Ratified under the Khronos Intellectual Property Framework. It is being made publicly available as a provisional extension to enable review and feedback from the community. While it is a provisional extension features may be added, removed, or changed in non-backward compatible ways.

If you have feedback please create an issue on: https://github.com/KhronosGroup/OpenCL-Docs/

46.1.3. Dependencies

This extension is written against the OpenCL Specification version 3.0.6.

This extension requires OpenCL 1.2 or later. Buffering of SVM commands requires OpenCL 2.0 or later.

46.1.4. Contributors

Ewan Crawford, Codeplay Software Ltd.
Gordon Brown, Codeplay Software Ltd.
Kenneth Benzie, Codeplay Software Ltd.
Alastair Murray, Codeplay Software Ltd.

Jack Frankland, Codeplay Software Ltd.
Balaji Calidas, Qualcomm Technologies Inc.
Joshua Kelly, Qualcomm Technologies, Inc.
Kevin Petit, Arm Ltd.
Aharon Abramson, Intel.
Ben Ashbaugh, Intel.
Boaz Ouriel, Intel.
Chris Gearing, Intel.
Pekka Jääskeläinen, Tampere University and Intel.
Jan Solanti, Tampere University
Nikhil Joshi, NVIDIA
James Price, Google
Brice Videau, Argonne National Laboratory

46.2. Overview

Command-buffers enable a reduction in overhead when enqueuing the same workload multiple times. By separating the command-queue setup from dispatch, the ability to replay a set of previously created commands is introduced.

Device-side *cl_sync_point_khr* synchronization-points can be used within command-buffers to define command dependencies. This allows the commands of a command-buffer to execute out-of-order on a single compatible command-queue. The command-buffer itself has no inherent in-order/out-of-order property, this ordering is inferred from the command-queue used on command recording. Out-of-order enqueues without event dependencies of both regular commands, such as **clEnqueueFillBuffer**, and command-buffers are allowed to execute concurrently, and it is up to the user to express any dependencies using events.

The command-queues a command-buffer will be executed on can be set on replay via parameters to **clEnqueueCommandBufferKHR**, provided they are compatible with the command-queues used on command-buffer recording.

46.2.1. Background

On embedded devices where building a command stream accounts for a significant expenditure of resources and where workloads are often required to be pipelined, a solution that minimizes driver overhead can significantly improve the utilization of accelerators by removing a bottleneck in repeated command stream generation.

An additional motivator is lowering task execution latency, as devices can be kept occupied with work by repeated submissions, without having to wait on the host to construct commands again for a similar workload.

46.2.2. Rationale

The command-buffer abstraction over the generation of command streams is a proven approach which facilitates a significant reduction in driver overhead in existing real-world applications with repetitive pipelined workloads which are built on top of Vulkan, DirectX 12, and Metal.

A primary goal is for a command-buffer to avoid any interaction with application code after being enqueued until all recorded commands have completed. As such, any command which maps or migrates memory objects; reads or writes memory objects; or enqueues a native kernel, is not available for command-buffer recording. Finally commands recorded into a command buffer do not wait for or return event objects, these are instead replaced with device-side synchronization-point identifiers which enable out-of-order execution when enqueued on compatible command-queues.

Adding new entry-points for individual commands, rather than recording existing command-queue APIs with begin/end markers was a design decision made for the following reasons:

- Individually specified entry points makes it clearer to the user what's supported, as opposed to adding a large number of error conditions throughout the specification with all the restrictions.
- Prevents code forking in existing entry points for the implementer, as otherwise separate paths in each entry point need to be maintained for both the recording and normal cases.
- Allows the definition of a new device-side synchronization primitive rather than overloading `cl_event`. As use of `cl_event` in individual commands allows host interaction from callback and user-events, as well as introducing complexities when a command-buffer is enqueued multiple times regarding profiling and execution status.
- New entry points facilitate returning handles to individual commands, allowing those commands to be modified between enqueues of the command buffer. Not all command handles are used in this extension, but providing them facilitates other extensions layered on top to take advantage of them to provide additional mutable functionality.

46.2.3. Simultaneous Use

The optional simultaneous use capability was added to the extension so that vendors can support pipelined workflows, where command-buffers are repeatedly enqueued without blocking in user code. However, simultaneous use may result in command-buffers being more expensive to enqueue than in a sequential model, so the capability is optional to enable optimizations on command-buffer recording.

46.3. Interactions with Other Extensions

The introduction of the command-buffer abstraction enables functionality beyond what the `cl_khr_command_buffer` extension currently provides, i.e. the recording of immutable commands to a single queue which can then be executed without commands synchronizing outside the command-buffer. It is intended that extra functionality expanding on this will be provided as layered extensions on top of `cl_khr_command_buffer`.

Having `cl_khr_command_buffer` as a minimal base specification means that the API defines mechanisms for functionality that is not enabled by this extension, these are described in the following sub-sections. `cl_khr_command_buffer` will retain its provisional extension status until other layered extensions are released, as these may reveal modifications needed to the base specification to support their intended use cases.

46.3.1. ND-range Kernel Command Properties

The **clCommandNDRangeKernelKHR** entry-point defines a `properties` parameter of new type `cl_ndrange_kernel_command_properties_khr`. No properties are defined in `cl_khr_command_buffer`, but the parameter is intended to enable future functionality that would change the characteristics of the kernel command.

46.3.2. Command Handles

All command recording entry-points define a `cl_mutable_command_khr` output parameter which provides a handle to the specific command being recorded. Use of these output handles is not enabled by the `cl_khr_command_buffer` extension, but the handles will allow individual commands in a command-buffer to be referenced by the user. In particular, the capability for an application to use these handles to modify commands between enqueues of a command-buffer is envisaged.

46.3.3. List of Queues

Only a single command-queue can be associated with a command-buffer in the `cl_khr_command_buffer` extension, but the API is designed with the intention that a future extension will allow commands to be recorded across multiple queues in the same command-buffer, providing replay of heterogeneous task graphs.

Using multiple queue functionality will result in an error without any layered extensions to relax usage of the following API features:

- When a command-buffer is created the API enables passing a list of queues that the command-buffer will record commands to. Only a single queue is permitted in `cl_khr_command_buffer`.
- Individual command recording entry-points define a `cl_command_queue` parameter for which of the queues set on command-buffer creation that command should be record to. This must be passed as NULL in `cl_khr_command_buffer`.
- **clEnqueueCommandBufferKHR** takes a list of queues for command-buffer execution, correspond to those set on creation. Only a single queue is permitted in `cl_khr_command_buffer`.

46.4. New Types

46.4.1. Command Buffer Types

Bitfield for querying command-buffer capabilities of an OpenCL device with **clGetDeviceInfo**, see device queries table:

```
typedef cl_bitfield cl_device_command_buffer_capabilities_khr
```

Types describing command-buffers:

```
// Returned by clCreateCommandBufferKHR()
typedef struct _cl_command_buffer_khr* cl_command_buffer_khr;
```

```
// Unique ID to a device-side synchronization-point used to describe the
// ordering of commands when recording a command-buffer. Valid for use
// only within the same command-buffer during recording.
typedef cl_uint cl_sync_point_khr;

// Handle returned on command recording
typedef struct _cl_mutable_command_khr* cl_mutable_command_khr;

// Properties of a clCommandNDRangeKernelKHR command
typedef cl_properties cl_ndrange_kernel_command_properties_khr;

// Properties for command-buffer creation
typedef cl_properties cl_command_buffer_properties_khr;

// Bitfield representing flags for command-buffers
typedef cl_bitfield cl_command_buffer_flags_khr;

// Enumerated type for use in clGetCommandBufferInfoKHR()
typedef cl_uint cl_command_buffer_info_khr;

// Return type for CL_COMMAND_BUFFER_STATE_KHR in clGetCommandBufferInfoKHR()
typedef cl_uint cl_command_buffer_state_khr;
```

46.5. New API Functions

Command-buffer entry points from Section 5.X:

```
cl_command_buffer_khr clCreateCommandBufferKHR(
    cl_uint num_queues,
    const cl_command_queue* queues,
    const cl_command_buffer_properties_khr* properties,
    cl_int* errcode_ret);

cl_int clRetainCommandBufferKHR(cl_command_buffer_khr command_buffer);

cl_int clReleaseCommandBufferKHR(cl_command_buffer_khr command_buffer);

cl_int clFinalizeCommandBufferKHR(cl_command_buffer_khr command_buffer);

cl_int clEnqueueCommandBufferKHR(
    cl_uint num_queues,
    cl_command_queue* queues,
    cl_command_buffer_khr command_buffer,
    cl_uint num_events_in_wait_list,
    const cl_event* event_wait_list,
    cl_event* event);

cl_int clCommandBarrierWithWaitListKHR(
```

```
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);

cl_int clCommandCopyBufferKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_mem src_buffer,
    cl_mem dst_buffer,
    size_t src_offset,
    size_t dst_offset,
    size_t size,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);

cl_int clCommandCopyBufferRectKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_mem src_buffer,
    cl_mem dst_buffer,
    const size_t* src_origin,
    const size_t* dst_origin,
    const size_t* region,
    size_t src_row_pitch,
    size_t src_slice_pitch,
    size_t dst_row_pitch,
    size_t dst_slice_pitch,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);

cl_int clCommandCopyBufferToImageKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_mem src_buffer,
    cl_mem dst_image,
    size_t src_offset,
    const size_t* dst_origin,
    const size_t* region,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);

cl_int clCommandCopyImageKHR(
```

```
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_mem src_image,
    cl_mem dst_image,
    const size_t* src_origin,
    const size_t* dst_origin,
    const size_t* region,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);

cl_int clCommandCopyImageToBufferKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_mem src_image,
    cl_mem dst_buffer,
    const size_t* src_origin,
    const size_t* region,
    size_t dst_offset,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);

cl_int clCommandFillBufferKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_mem buffer,
    const void* pattern,
    size_t pattern_size,
    size_t offset,
    size_t size,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);

cl_int clCommandFillImageKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_mem image,
    const void* fill_color,
    const size_t* origin,
    const size_t* region,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);

cl_int clCommandNDRangeKernelKHR(
```

```
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    const cl_ndrange_kernel_command_properties_khr* properties,
    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_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);

cl_int clGetCommandBufferInfoKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_buffer_info_khr param_name,
    size_t param_value_size,
    void* param_value,
    size_t* param_value_size_ret);
```

The following SVM entry points are supported only with at least OpenCL 2.0 and starting from 0.9.4 of this extension:

```
cl_int clCommandSVMMemcpyKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    void* dst_ptr,
    const void* src_ptr,
    size_t size,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);

cl_int clCommandSVMMemFillKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    void* svm_ptr,
    const void* pattern,
    size_t pattern_size,
    size_t size,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);
```

46.6. New API Enums

Enums for querying device command-buffer capabilities with **clGetDeviceInfo**, see device queries table:

```
// Accepted values for the param_name parameter to clGetDeviceInfo
CL_DEVICE_COMMAND_BUFFER_CAPABILITIES_KHR              0x12A9
CL_DEVICE_COMMAND_BUFFER_REQUIRED_QUEUE_PROPERTIES_KHR 0x12AA

// Bits for cl_device_command_buffer_capabilities_khr bitfield
CL_COMMAND_BUFFER_CAPABILITY_KERNEL_PRINTF_KHR        (0x1 << 0)
CL_COMMAND_BUFFER_CAPABILITY_DEVICE_SIDE_ENQUEUE_KHR  (0x1 << 1)
CL_COMMAND_BUFFER_CAPABILITY_SIMULTANEOUS_USE_KHR     (0x1 << 2)
CL_COMMAND_BUFFER_CAPABILITY_OUT_OF_ORDER_KHR         (0x1 << 3)

// Values for cl_command_buffer_state_khr
CL_COMMAND_BUFFER_STATE_RECORDING_KHR             0x0
CL_COMMAND_BUFFER_STATE_EXECUTABLE_KHR            0x1
CL_COMMAND_BUFFER_STATE_PENDING_KHR               0x2
```

Enums for base command-buffers functionality:

```
// Error codes
CL_INVALID_COMMAND_BUFFER_KHR                     -1138
CL_INVALID_SYNC_POINT_WAIT_LIST_KHR               -1139
CL_INCOMPATIBLE_COMMAND_QUEUE_KHR                 -1140

// Bitfield to clCreateCommandBufferKHR
CL_COMMAND_BUFFER_FLAGS_KHR                       0x1293

// Bits for cl_command_buffer_flags_khr bitfield
CL_COMMAND_BUFFER_SIMULTANEOUS_USE_KHR            (0x1 << 0)

// cl_command_buffer_info_khr queries to clGetCommandBufferInfoKHR
CL_COMMAND_BUFFER_QUEUES_KHR                      0x1294
CL_COMMAND_BUFFER_NUM_QUEUES_KHR                  0x1295
CL_COMMAND_BUFFER_REFERENCE_COUNT_KHR             0x1296
CL_COMMAND_BUFFER_STATE_KHR                       0x1297
CL_COMMAND_BUFFER_PROPERTIES_ARRAY_KHR            0x1298
CL_COMMAND_BUFFER_CONTEXT_KHR                     0x1299

// cl_event command-buffer enqueue command type
CL_COMMAND_COMMAND_BUFFER_KHR                     0x12A8
```

46.7. Modifications to section 4.2 of the OpenCL API Specification

Add to **Table 5**, *Device Queries*, of section 4.2:

Table 5. List of supported param_names by **clGetDeviceInfo**

Device Info	Return Type	Description
`CL_DEVICE_COMMAND_BUFFER_CAPABILITIES_KHR`	`cl_device_command_buffer_capabilities_khr`	Describes device command-buffer capabilities, encoded as bits in a bitfield. Supported capabilities are: `CL_COMMAND_BUFFER_CAPABILITY_KERNEL_PRINTF_KHR` Device supports the ability to record commands that execute kernels which contain printf calls. `CL_COMMAND_BUFFER_CAPABILITY_DEVICE_SIDE_ENQUEUE_KHR` Device supports the ability to record commands that execute kernels which contain device-side kernel-enqueue calls. `CL_COMMAND_BUFFER_CAPABILITY_SIMULTANEOUS_USE_KHR` Device supports the command-buffers having a Pending Count that exceeds 1. `CL_COMMAND_BUFFER_CAPABILITY_OUT_OF_ORDER_KHR` Device supports the ability to record command-buffers to out-of-order command-queues.
`CL_DEVICE_COMMAND_BUFFER_REQUIRED_QUEUE_PROPERTIES_KHR`	`cl_command_queue_properties`	Bitmask of the minimum properties with which a command-queue must be created to allow a command-buffer to be executed on it. It is valid for a command-queue to be created with extra properties in addition to this base requirement and still be compatible with command-buffer execution.

46.8. Add new section "Section 5.X - Command Buffers" to OpenCL API Specification

A *command-buffer* object represents a series of operations to be enqueued on one or more command-queues without any application code interaction. Grouping the operations together allows efficient enqueuing of repetitive operations, as well as enabling driver optimizations.

Command-buffers are *sequential use* by default, but may also be set to *simultaneous use* on creation if the device optionally supports this capability. A sequential use command-buffer must have a Pending Count of 0 or 1. The simultaneous use capability removes this restriction and allows command-buffers to have a Pending Count greater than 1.

Command-buffers are created using an ordered list of command-queues that commands are recorded to and execute on by default. These command-queues can be replaced on command-buffer enqueue with different command-queues, provided for each element in the replacement list the substitute command-queue is compatible with the command-queue used on command-buffer creation. Where a *compatible* command-queue is defined as a command-queue with identical properties targeting the same device and in the same OpenCL context.

While constructing a command-buffer it is valid for the user to interleave calls to the same queue which create commands, such as **clCommandNDRangeKernelKHR**, with queue submission calls, such as **clEnqueueNDRangeKernel** or **clEnqueueCommandBufferKHR**. That is, there is no effect on queue state from recording commands. The purpose of the queue parameter is to define the device and properties of the command, which are constant queries on the queue object.

A command-buffer object should increment the reference count of attached OpenCL objects such as queues, buffers, images, and kernels referenced in commands recorded to the command-buffer. This enables correct behavior of the command-buffer when its attached objects have been released. On destruction of the command-buffer it should decrement these reference counts, allowing the attached objects to be freed if appropriate.

> A command-buffer object does not update the reference count of objects set as arguments on kernels recorded into the command-buffer. This is consistent with the reference counting behavior of **clSetKernelArg**.
>
> Applications should ensure that objects passed as arguments to kernels recorded to a command-buffer are not deleted until the command-buffer has been released. Undefined behavior may result from the failure to follow this usage requirement for all the command-buffers an object is used as a kernel argument in.
>
> If using layered extension `cl_khr_command_buffer_mutable_dispatch`, see related note on safe usage.

46.8.1. Add new section "Section 5.X.1 - Command Buffer Lifecycle"

A command-buffer is always in one of the following states:

Recording

Initial state of a command-buffer on creation, where commands can be recorded to the command-buffer.

Executable

State after command recording has finished with **clFinalizeCommandBufferKHR** and the command-buffer may be enqueued.

Pending

Once a command-buffer has been enqueued to a command-queue it enters the Pending state until completion, at which point it moves back to the Executable state.

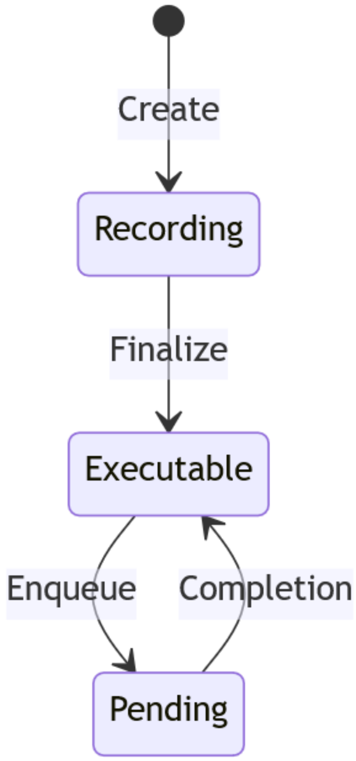

Figure 1. Lifecycle of a command-buffer.

The Pending Count is the number of copies of the command buffer in the Pending state. By default a command-buffer's Pending Count must be 0 or 1. If the command-buffer was created with `CL_COMMAND_BUFFER_SIMULTANEOUS_USE_KHR` then the command-buffer may have a Pending Count greater than 1.

46.8.2. Add new section "Section 5.X.2 - Creating Command Buffer Objects"

The function

```
cl_command_buffer_khr clCreateCommandBufferKHR(
    cl_uint num_queues,
    const cl_command_queue* queues,
    const cl_command_buffer_properties_khr* properties,
    cl_int* errcode_ret);
```

Is used to create a command-buffer that can record commands to the specified queues.

> Upon creation the command-buffer is defined as being in the Recording state, in order for the command-buffer to be enqueued it must first be finalized using **clFinalizeCommandBufferKHR** after which no further commands can be recorded. A command-buffer is submitted for execution on command-queues with a call to **clEnqueueCommandBufferKHR**.

num_queues The number of command-queues listed in *queues*. This extension only supports a single command-queue, so this **must** be one.

queues Is a pointer to a command-queue that the command-buffer commands will be recorded to.

queues must be a non-NULL value.

properties Specifies a list of properties for the command-buffer and their corresponding values. Each property name is immediately followed by the corresponding desired value. The list is terminated with 0. The list of supported properties is described in the table below. If a supported property and its value is not specified in properties, its default value will be used. *properties* can be NULL in which case the default values for supported command-buffer properties will be used.

Table 69. **clCreateCommandBufferKHR** *properties*

Recording Properties	Property Value	Description
CL_COMMAND_BUFFER_FLAGS_KHR	cl_command_buffer_flags_khr	This is a bitfield and can be set to a combination of the following values: CL_COMMAND_BUFFER_SIMULTANEOUS_USE_KHR - Allow multiple instances of the command-buffer to be submitted to the device for execution. If set, devices must support CL_COMMAND_BUFFER_CAPABILITY_SIMULTANEOUS_USE_KHR. The default value of this property is 0.

errcode_ret Returns an appropriate error code. If *errcode_ret* is NULL, no error code is returned.

clCreateCommandBufferKHR returns a valid non-zero command-buffer and *errcode_ret* is set to CL_SUCCESS if the command-buffer is created successfully. Otherwise, it returns a NULL value with one of the following error values returned in *errcode_ret*:

- CL_INVALID_COMMAND_QUEUE if any command-queue in *queues* is not a valid command-queue.
- CL_INCOMPATIBLE_COMMAND_QUEUE_KHR if any command-queue in *queues* is an out-of-order command-queue and the device associated with the command-queue does not support the CL_COMMAND_BUFFER_CAPABILITY_OUT_OF_ORDER_KHR capability.
- CL_INCOMPATIBLE_COMMAND_QUEUE_KHR if the properties of any command-queue in *queues* does not contain the minimum properties specified by CL_DEVICE_COMMAND_BUFFER_REQUIRED_QUEUE_PROPERTIES_KHR.
- CL_INVALID_CONTEXT if all the command-queues in *queues* do not have the same OpenCL context.
- CL_INVALID_VALUE if *num_queues* is not one.
- CL_INVALID_VALUE if *queues* is NULL.
- CL_INVALID_VALUE if values specified in *properties* are not valid, or if the same property name is specified more than once.

- CL_INVALID_PROPERTY if values specified in *properties* are valid but are not supported by all the devices associated with command-queues in *queues*.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

```
cl_int clRetainCommandBufferKHR(
    cl_command_buffer_khr command_buffer);
```

Increments the *command_buffer* reference count.

command_buffer Specifies the command-buffer to retain.

clRetainCommandBufferKHR returns CL_SUCCESS if the function is executed successfully. Otherwise, it returns one of the following errors:

- CL_INVALID_COMMAND_BUFFER_KHR if *command_buffer* is not a valid command-buffer.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

```
cl_int clReleaseCommandBufferKHR(
    cl_command_buffer_khr command_buffer);
```

Decrements the *command_buffer* reference count.

> After the *command_buffer* reference count becomes zero and has finished execution, the command-buffer is deleted.

command_buffer Specifies the command-buffer to release.

clReleaseCommandBufferKHR returns CL_SUCCESS if the function is executed successfully. Otherwise, it returns one of the following errors:

- CL_INVALID_COMMAND_BUFFER_KHR if *command_buffer* is not a valid command-buffer.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

46.8.3. Add new section "Section 5.X.2 - Enqueuing a Command Buffer"

The function

```
cl_int clFinalizeCommandBufferKHR(
    cl_command_buffer_khr command_buffer);
```

Finalizes command recording ready for enqueuing the command-buffer on a command-queue.

> **clFinalizeCommandBufferKHR** places the command-buffer in the Executable state where commands can no longer be recorded, at this point the command-buffer is ready to be enqueued.

command_buffer Refers to a valid command-buffer object.

clFinalizeCommandBufferKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns one of the following errors:

- `CL_INVALID_COMMAND_BUFFER_KHR` if *command_buffer* is not a valid command-buffer.
- `CL_INVALID_OPERATION` if *command_buffer* is not in the Recording state.
- `CL_OUT_OF_RESOURCES` if there is a failure to allocate resources required by the OpenCL implementation on the device.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

```
cl_int clEnqueueCommandBufferKHR(
    cl_uint num_queues,
    cl_command_queue* queues,
    cl_command_buffer_khr command_buffer,
    cl_uint num_events_in_wait_list,
    const cl_event* event_wait_list,
    cl_event* event);
```

Enqueues a command-buffer to execute on command-queues specified by *queues*, or on default command-queues used during recording if *queues* is empty.

> To enqueue a command-buffer it must be in a Executable state, see **clFinalizeCommandBufferKHR**.

num_queues The number of command-queues listed in *queues*.

queues A pointer to an ordered list of command-queues compatible with the command-queues used on recording. *queues* can be `NULL` in which case the default command-queues used on command-buffer creation are used and *num_queues* must be 0.

command_buffer Refers to a valid command-buffer object.

event_wait_list, *num_events_in_wait_list* Specify events that need to complete before this particular command can be executed. If *event_wait_list* is `NULL`, then this particular command does not wait on any event to complete. If *event_wait_list* is `NULL`, *num_events_in_wait_list* must be 0. If event_wait_list is not `NULL`, the list of events pointed to by *event_wait_list* must be valid and *num_events_in_wait_list* must be greater than 0. The events specified in *event_wait_list* act as synchronization points. The context associated with events in *event_wait_list* and command_queue must be the same. The memory associated with *event_wait_list* can be reused or freed after the function returns.

event Returns an event object that identifies this command and can be used to query for profiling information or queue a wait for this particular command to complete. *event* can be `NULL` in which case it will not be possible for the application to wait on this command or query it for profiling information.

clEnqueueCommandBufferKHR returns `CL_SUCCESS` if the command-buffer execution was successfully queued, or one of the errors below:

- `CL_INVALID_COMMAND_BUFFER_KHR` if *command_buffer* is not a valid command-buffer.
- `CL_INVALID_OPERATION` if *command_buffer* has not been finalized.
- `CL_INVALID_OPERATION` if *command_buffer* was not created with the `CL_COMMAND_BUFFER_SIMULTANEOUS_USE_KHR` flag and is in the Pending state.
- `CL_INVALID_VALUE` if *queues* is `NULL` and *num_queues* is > 0, or *queues* is not `NULL` and *num_queues* is 0.
- `CL_INVALID_VALUE` if *num_queues* is > 0 and not the same value as *num_queues* set on *command_buffer* creation.
- `CL_INVALID_COMMAND_QUEUE` if any element of *queues* is not a valid command-queue.
- `CL_INCOMPATIBLE_COMMAND_QUEUE_KHR` if any element of *queues* is not compatible with the command-queue set on *command_buffer* creation at the same list index.
- `CL_INVALID_CONTEXT` if any element of *queues* does not have the same context as the command-queue set on *command_buffer* creation at the same list index.
- `CL_INVALID_CONTEXT` if context associated with *command_buffer* and events in *event_wait_list* are not the same.
- `CL_OUT_OF_RESOURCES` if there is a failure to queue the execution instance of *command_buffer* on the command-queues because of insufficient resources needed to execute *command_buffer*.
- `CL_INVALID_EVENT_WAIT_LIST` if *event_wait_list* is `NULL` and *num_events_in_wait_list* > 0, or *event_wait_list* is not `NULL` and *num_events_in_wait_list* is 0, or if event objects in *event_wait_list* are not valid events.
- `CL_OUT_OF_RESOURCES` if there is a failure to allocate resources required by the OpenCL implementation on the device.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

46.8.4. Add new section "Section 5.X.3 - Recording Commands to a Command Buffer"

The function

```
cl_int clCommandBarrierWithWaitListKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);
```

Records a barrier operation used as a synchronization point.

> **clCommandBarrierWithWaitListKHR** Waits for either a list of synchronization-points to complete, or if the list is empty it waits for all commands previously recorded in *command_buffer* to complete before it completes. This command blocks command execution, that is, any following commands recorded after it do not execute until it completes.

command_buffer Refers to a valid command-buffer object.

command_queue Specifies the command-queue the command will be recorded to. Parameter is unused by this extension as only a single command-queue is supported and **must** be `NULL`.

sync_point_wait_list, *num_sync_points_in_wait_list* Specify synchronization-points that need to complete before this particular command can be executed.

If *sync_point_wait_list* is `NULL`, *num_sync_points_in_wait_list* must be 0. If *sync_point_wait_list* is not `NULL`, the list of synchronization-points pointed to by *sync_point_wait_list* must be valid and *num_sync_points_in_wait_list* must be greater than 0. The synchronization-points specified in *sync_point_wait_list* are **device-side** synchronization-points. The command-buffer associated with synchronization-points in *sync_point_wait_list* must be the same as *command_buffer*. The memory associated with *sync_point_wait_list* can be reused or freed after the function returns.

If *sync_point_wait_list* is `NULL`, then this particular command waits until all previous recorded commands to *command_queue* have completed.

sync_point Returns a synchronization-point ID that identifies this particular command. Synchronization-point objects are unique and can be used to identify this barrier command later on. *sync_point* can be `NULL` in which case it will not be possible for the application to record a wait for this command to complete. If the *sync_point_wait_list* and the *sync_point* arguments are not `NULL`, the *sync_point* argument should not refer to an element of the *sync_point_wait_list* array.

mutable_handle Returns a handle to the command. Handle is unused by this extension and must be passed as `NULL`.

clCommandBarrierWithWaitListKHR returns `CL_SUCCESS` if the function is executed successfully.

Otherwise, it returns one of the following errors:

- CL_INVALID_COMMAND_QUEUE if *command_queue* is not NULL.
- CL_INVALID_COMMAND_BUFFER_KHR if *command_buffer* is not a valid command-buffer.
- CL_INVALID_CONTEXT if the context associated with *command_queue* and *command_buffer* is not the same.
- CL_INVALID_OPERATION if *command_buffer* has been finalized.
- CL_INVALID_VALUE if *mutable_handle* is not NULL.
- CL_INVALID_SYNC_POINT_WAIT_LIST_KHR if *sync_point_wait_list* is NULL and *num_sync_points_in_wait_list* is > 0, or *sync_point_wait_list* is not NULL and *num_sync_points_in_wait_list* is 0, or if synchronization-point objects in *sync_point_wait_list* are not valid synchronization-points.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

The function

```
cl_int clCommandCopyBufferKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_mem src_buffer,
    cl_mem dst_buffer,
    size_t src_offset,
    size_t dst_offset,
    size_t size,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);
```

Records a command to copy from one buffer object to another.

command_buffer Refers to a valid command-buffer object.

command_queue Specifies the command-queue the command will be recorded to. Parameter is unused by this extension as only a single command-queue is supported and **must** be NULL.

src_buffer, *dst_buffer*, *src_offset*, *dst_offset*, *size* Refer to **clEnqueueCopyBuffer**.

sync_point_wait_list, *num_sync_points_in_wait_list* Specify synchronization-points that need to complete before this particular command can be executed.

If *sync_point_wait_list* is NULL, *num_sync_points_in_wait_list* must be 0. If *sync_point_wait_list* is not NULL, the list of synchronization-points pointed to by *sync_point_wait_list* must be valid and *num_sync_points_in_wait_list* must be greater than 0. The synchronization-points specified in *sync_point_wait_list* are **device-side** synchronization-points. The command-buffer associated with synchronization-points in *sync_point_wait_list* must be the same as *command_buffer*. The memory associated with *sync_point_wait_list* can be reused or freed after the function returns.

sync_point Returns a synchronization-point ID that identifies this particular command. Synchronization-point objects are unique and can be used to identify this command later on. *sync_point* can be `NULL` in which case it will not be possible for the application to record a wait for this command to complete. If the *sync_point_wait_list* and the *sync_point* arguments are not `NULL`, the *sync_point* argument should not refer to an element of the *sync_point_wait_list* array.

mutable_handle Returns a handle to the command. Handle is unused by this extension and must be passed as `NULL`.

clCommandCopyBufferKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns the errors defined by **clEnqueueCopyBuffer** except:

`CL_INVALID_COMMAND_QUEUE` is replaced with:

- `CL_INVALID_COMMAND_QUEUE` if *command_queue* is not `NULL`.

`CL_INVALID_CONTEXT` is replaced with:

- `CL_INVALID_CONTEXT` if the context associated with *command_queue*, *command_buffer*, *src_buffer*, and *dst_buffer* are not the same.

`CL_INVALID_EVENT_WAIT_LIST` is replaced with:

- `CL_INVALID_SYNC_POINT_WAIT_LIST_KHR` if *sync_point_wait_list* is `NULL` and *num_sync_points_in_wait_list* is > 0, or *sync_point_wait_list* is not `NULL` and *num_sync_points_in_wait_list* is 0, or if synchronization-point objects in *sync_point_wait_list* are not valid synchronization-points.

New errors:

- `CL_INVALID_COMMAND_BUFFER_KHR` if *command_buffer* is not a valid command-buffer.
- `CL_INVALID_OPERATION` if *command_buffer* has been finalized.
- `CL_INVALID_VALUE` if *mutable_handle* is not `NULL`.

The function

```
cl_int clCommandCopyBufferRectKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_mem src_buffer,
    cl_mem dst_buffer,
    const size_t* src_origin,
    const size_t* dst_origin,
    const size_t* region,
    size_t src_row_pitch,
    size_t src_slice_pitch,
    size_t dst_row_pitch,
    size_t dst_slice_pitch,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);
```

Records a command to copy a rectangular region from a buffer object to another buffer object.

> **clCommandCopyBufferRectKHR** records a command to copy a 2D or 3D rectangular region from the buffer object identified by *src_buffer* to a 2D or 3D region in the buffer object identified by *dst_buffer*. Copying begins at the source offset and destination offset which are computed as described in the description for *src_origin* and *dst_origin*.
>
> Each byte of the region's width is copied from the source offset to the destination offset. After copying each width, the source and destination offsets are incremented by their respective source and destination row pitches. After copying each 2D rectangle, the source and destination offsets are incremented by their respective source and destination slice pitches.

command_buffer Refers to a valid command-buffer object.

command_queue Specifies the command-queue the command will be recorded to. Parameter is unused by this extension as only a single command-queue is supported and **must** be `NULL`.

src_origin, *dst_origin*, *region*, *src_row_pitch*, *src_slice_pitch*, *dst_row_pitch*, *dst_slice_pitch* Refer to **clEnqueueCopyBufferRect**.

sync_point_wait_list, *num_sync_points_in_wait_list* Specify synchronization-points that need to complete before this particular command can be executed.

If *sync_point_wait_list* is `NULL`, *num_sync_points_in_wait_list* must be 0. If *sync_point_wait_list* is not `NULL`, the list of synchronization-points pointed to by *sync_point_wait_list* must be valid and *num_sync_points_in_wait_list* must be greater than 0. The synchronization-points specified in *sync_point_wait_list* are **device-side** synchronization-points. The command-buffer associated with synchronization-points in *sync_point_wait_list* must be the same as *command_buffer*. The memory associated with *sync_point_wait_list* can be reused or freed after the function returns.

sync_point Returns a synchronization-point ID that identifies this particular command. Synchronization-point objects are unique and can be used to identify this command later on. *sync_point* can be `NULL` in which case it will not be possible for the application to record a wait for this command to complete. If the *sync_point_wait_list* and the *sync_point* arguments are not `NULL`, the *sync_point* argument should not refer to an element of the *sync_point_wait_list* array.

mutable_handle Returns a handle to the command. Handle is unused by this extension and must be passed as `NULL`.

clCommandCopyBufferRectKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns the errors defined by **clEnqueueCopyBufferRect** except:

`CL_INVALID_COMMAND_QUEUE` is replaced with:

- `CL_INVALID_COMMAND_QUEUE` if *command_queue* is not `NULL`.

`CL_INVALID_CONTEXT` is replaced with:

- `CL_INVALID_CONTEXT` if the context associated with *command_queue*, *command_buffer*, *src_buffer*, and *dst_buffer* are not the same.

`CL_INVALID_EVENT_WAIT_LIST` is replaced with:

- `CL_INVALID_SYNC_POINT_WAIT_LIST_KHR` if *sync_point_wait_list* is `NULL` and *num_sync_points_in_wait_list* is > 0, or *sync_point_wait_list* is not `NULL` and *num_sync_points_in_wait_list* is 0, or if synchronization-point objects in *sync_point_wait_list* are not valid synchronization-points.

New errors:

- `CL_INVALID_COMMAND_BUFFER_KHR` if *command_buffer* is not a valid command-buffer.
- `CL_INVALID_OPERATION` if *command_buffer* has been finalized.
- `CL_INVALID_VALUE` if *mutable_handle* is not `NULL`.

The function

```
cl_int clCommandCopyBufferToImageKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_mem src_buffer,
    cl_mem dst_image,
    size_t src_offset,
    const size_t* dst_origin,
    const size_t* region,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);
```

Records a command to copy a buffer object to an image object.

command_buffer Refers to a valid command-buffer object.

command_queue Specifies the command-queue the command will be recorded to. Parameter is unused by this extension as only a single command-queue is supported and **must** be `NULL`.

src_buffer, *dst_image*, *src_offset*, *dst_origin*, *region* Refer to **clEnqueueCopyBufferToImage**

sync_point_wait_list, *num_sync_points_in_wait_list* Specify synchronization-points that need to complete before this particular command can be executed.

If *sync_point_wait_list* is `NULL`, *num_sync_points_in_wait_list* must be 0. If *sync_point_wait_list* is not `NULL`, the list of synchronization-points pointed to by *sync_point_wait_list* must be valid and *num_sync_points_in_wait_list* must be greater than 0. The synchronization-points specified in *sync_point_wait_list* are **device-side** synchronization-points. The command-buffer associated with synchronization-points in *sync_point_wait_list* must be the same as *command_buffer*. The memory associated with *sync_point_wait_list* can be reused or freed after the function returns.

sync_point Returns a synchronization-point ID that identifies this particular command. Synchronization-point objects are unique and can be used to identify this command later on. *sync_point* can be `NULL` in which case it will not be possible for the application to record a wait for this command to complete. If the *sync_point_wait_list* and the *sync_point* arguments are not `NULL`, the *sync_point* argument should not refer to an element of the *sync_point_wait_list* array.

mutable_handle Returns a handle to the command. Handle is unused by this extension and must be passed as `NULL`.

clCommandCopyBufferToImageKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns the errors defined by **clEnqueueCopyBufferToImage** except:

`CL_INVALID_COMMAND_QUEUE` is replaced with:

- `CL_INVALID_COMMAND_QUEUE` if *command_queue* is not `NULL`.

`CL_INVALID_CONTEXT` is replaced with:

- `CL_INVALID_CONTEXT` if the context associated with *command_queue*, *command_buffer*, *src_buffer*, and *dst_image* are not the same.

`CL_INVALID_EVENT_WAIT_LIST` is replaced with:

- `CL_INVALID_SYNC_POINT_WAIT_LIST_KHR` if *sync_point_wait_list* is `NULL` and *num_sync_points_in_wait_list* is > 0, or *sync_point_wait_list* is not `NULL` and *num_sync_points_in_wait_list* is 0, or if synchronization-point objects in *sync_point_wait_list* are not valid synchronization-points.

New errors:

- `CL_INVALID_COMMAND_BUFFER_KHR` if *command_buffer* is not a valid command-buffer.
- `CL_INVALID_OPERATION` if *command_buffer* has been finalized.

* CL_INVALID_VALUE if *mutable_handle* is not NULL.

The function

```
cl_int clCommandCopyImageKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_mem src_image,
    cl_mem dst_image,
    const size_t* src_origin,
    const size_t* dst_origin,
    const size_t* region,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);
```

Records a command to copy image objects.

> It is currently a requirement that the *src_image* and *dst_image* image memory objects for **clCommandCopyImageKHR** must have the exact same image format, i.e. the cl_image_format descriptor specified when *src_image* and *dst_image* are created must match.

command_buffer Refers to a valid command-buffer object.

command_queue Specifies the command-queue the command will be recorded to. Parameter is unused by this extension as only a single command-queue is supported and **must** be NULL.

src_image, *dst_image*, *src_origin*, *dst_origin*, *region* Refer to **clEnqueueCopyImage**.

sync_point_wait_list, *num_sync_points_in_wait_list* Specify synchronization-points that need to complete before this particular command can be executed.

If *sync_point_wait_list* is NULL, *num_sync_points_in_wait_list* must be 0. If *sync_point_wait_list* is not NULL, the list of synchronization-points pointed to by *sync_point_wait_list* must be valid and *num_sync_points_in_wait_list* must be greater than 0. The synchronization-points specified in *sync_point_wait_list* are **device-side** synchronization-points. The command-buffer associated with synchronization-points in *sync_point_wait_list* must be the same as *command_buffer*. The memory associated with *sync_point_wait_list* can be reused or freed after the function returns.

sync_point Returns a synchronization-point ID that identifies this particular command. Synchronization-point objects are unique and can be used to identify this command later on. *sync_point* can be NULL in which case it will not be possible for the application to record a wait for this command to complete. If the *sync_point_wait_list* and the *sync_point* arguments are not NULL, the *sync_point* argument should not refer to an element of the *sync_point_wait_list* array.

mutable_handle Returns a handle to the command. Handle is unused by this extension and must be passed as NULL.

clCommandCopyImageKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns the errors defined by **clEnqueueCopyImage** except:

`CL_INVALID_COMMAND_QUEUE` is replaced with:

- `CL_INVALID_COMMAND_QUEUE` if *command_queue* is not `NULL`.

`CL_INVALID_CONTEXT` is replaced with:

- `CL_INVALID_CONTEXT` if the context associated with *command_queue*, *command_buffer*, *src_image*, and *dst_image* are not the same.

`CL_INVALID_EVENT_WAIT_LIST` is replaced with:

- `CL_INVALID_SYNC_POINT_WAIT_LIST_KHR` if *sync_point_wait_list* is `NULL` and *num_sync_points_in_wait_list* is > 0, or *sync_point_wait_list* is not `NULL` and *num_sync_points_in_wait_list* is 0, or if synchronization-point objects in *sync_point_wait_list* are not valid synchronization-points.

New errors:

- `CL_INVALID_COMMAND_BUFFER_KHR` if *command_buffer* is not a valid command-buffer.
- `CL_INVALID_OPERATION` if *command_buffer* has been finalized.
- `CL_INVALID_VALUE` if *mutable_handle* is not `NULL`.

The function

```
cl_int clCommandCopyImageToBufferKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_mem src_image,
    cl_mem dst_buffer,
    const size_t* src_origin,
    const size_t* region,
    size_t dst_offset,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);
```

Records a command to copy an image object to a buffer object.

command_buffer Refers to a valid command-buffer object.

command_queue Specifies the command-queue the command will be recorded to. Parameter is unused by this extension as only a single command-queue is supported and **must** be `NULL`.

src_image, *dst_buffer*, *src_origin*, *region*, *dst_offset* Refer to **clEnqueueCopyImageToBuffer**.

sync_point_wait_list, *num_sync_points_in_wait_list* Specify synchronization-points that need to complete before this particular command can be executed.

If *sync_point_wait_list* is NULL, *num_sync_points_in_wait_list* must be 0. If *sync_point_wait_list* is not NULL, the list of synchronization-points pointed to by *sync_point_wait_list* must be valid and *num_sync_points_in_wait_list* must be greater than 0. The synchronization-points specified in *sync_point_wait_list* are **device-side** synchronization-points. The command-buffer associated with synchronization-points in *sync_point_wait_list* must be the same as *command_buffer*. The memory associated with *sync_point_wait_list* can be reused or freed after the function returns.

sync_point Returns a synchronization-point ID that identifies this particular command. Synchronization-point objects are unique and can be used to identify this command later on. *sync_point* can be NULL in which case it will not be possible for the application to record a wait for this command to complete. If the *sync_point_wait_list* and the *sync_point* arguments are not NULL, the *sync_point* argument should not refer to an element of the *sync_point_wait_list* array.

mutable_handle Returns a handle to the command. Handle is unused by this extension and must be passed as NULL.

clCommandCopyImageToBufferKHR returns CL_SUCCESS if the function is executed successfully. Otherwise, it returns the errors defined by **clEnqueueCopyImageToBuffer** except:

CL_INVALID_COMMAND_QUEUE is replaced with:

- CL_INVALID_COMMAND_QUEUE if *command_queue* is not NULL.

CL_INVALID_CONTEXT is replaced with:

- CL_INVALID_CONTEXT if the context associated with *command_queue*, *command_buffer*, *src_image*, and *dst_buffer* are not the same.

CL_INVALID_EVENT_WAIT_LIST is replaced with:

- CL_INVALID_SYNC_POINT_WAIT_LIST_KHR if *sync_point_wait_list* is NULL and *num_sync_points_in_wait_list* is > 0, or *sync_point_wait_list* is not NULL and *num_sync_points_in_wait_list* is 0, or if synchronization-point objects in *sync_point_wait_list* are not valid synchronization-points.

New errors:

- CL_INVALID_COMMAND_BUFFER_KHR if *command_buffer* is not a valid command-buffer.
- CL_INVALID_OPERATION if *command_buffer* has been finalized.
- CL_INVALID_VALUE if *mutable_handle* is not NULL.

The function

```
cl_int clCommandFillBufferKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_mem buffer,
    const void* pattern,
    size_t pattern_size,
    size_t offset,
    size_t size,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);
```

Records a command to fill a buffer object with a pattern of a given pattern size.

> The usage information which indicates whether the memory object can be read or written by a kernel and/or the host and is given by the `cl_mem_flags` argument value specified when *buffer* is created is ignored by **clCommandFillBufferKHR**.

command_buffer Refers to a valid command-buffer object.

command_queue Specifies the command-queue the command will be recorded to. Parameter is unused by this extension as only a single command-queue is supported and **must** be `NULL`.

buffer, *pattern*, *pattern_size*, *offset*, *size* Refer to **clEnqueueFillBuffer**.

sync_point_wait_list, *num_sync_points_in_wait_list* Specify synchronization-points that need to complete before this particular command can be executed.

If *sync_point_wait_list* is `NULL`, *num_sync_points_in_wait_list* must be 0. If *sync_point_wait_list* is not `NULL`, the list of synchronization-points pointed to by *sync_point_wait_list* must be valid and *num_sync_points_in_wait_list* must be greater than 0. The synchronization-points specified in *sync_point_wait_list* are **device-side** synchronization-points. The command-buffer associated with synchronization-points in *sync_point_wait_list* must be the same as *command_buffer*. The memory associated with *sync_point_wait_list* can be reused or freed after the function returns.

sync_point Returns a synchronization-point ID that identifies this particular command. Synchronization-point objects are unique and can be used to identify this command later on. *sync_point* can be `NULL` in which case it will not be possible for the application to record a wait for this command to complete. If the *sync_point_wait_list* and the *sync_point* arguments are not `NULL`, the *sync_point* argument should not refer to an element of the *sync_point_wait_list* array.

mutable_handle Returns a handle to the command. Handle is unused by this extension and must be passed as `NULL`.

clCommandFillBufferKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns the errors defined by **clEnqueueFillBuffer** except:

`CL_INVALID_COMMAND_QUEUE` is replaced with:

- CL_INVALID_COMMAND_QUEUE if *command_queue* is not NULL.

CL_INVALID_CONTEXT is replaced with:

- CL_INVALID_CONTEXT if the context associated with *command_queue*, *command_buffer*, and *buffer* are not the same.

CL_INVALID_EVENT_WAIT_LIST is replaced with:

- CL_INVALID_SYNC_POINT_WAIT_LIST_KHR if *sync_point_wait_list* is NULL and *num_sync_points_in_wait_list* is > 0, or *sync_point_wait_list* is not NULL and *num_sync_points_in_wait_list* is 0, or if synchronization-point objects in *sync_point_wait_list* are not valid synchronization-points.

New errors:

- CL_INVALID_COMMAND_BUFFER_KHR if *command_buffer* is not a valid command-buffer.
- CL_INVALID_OPERATION if *command_buffer* has been finalized.
- CL_INVALID_VALUE if *mutable_handle* is not NULL.

The function

```
cl_int clCommandFillImageKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    cl_mem image,
    const void* fill_color,
    const size_t* origin,
    const size_t* region,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);
```

Records a command to fill an image object with a specified color.

> The usage information which indicates whether the memory object can be read or written by a kernel and/or the host and is given by the cl_mem_flags argument value specified when image is created is ignored by **clCommandFillImageKHR**.

command_buffer Refers to a valid command-buffer object.

command_queue Specifies the command-queue the command will be recorded to. Parameter is unused by this extension as only a single command-queue is supported and **must** be NULL.

image, *fill_color*, *origin*, *region* Refer to **clEnqueueFillImage**.

sync_point_wait_list, *num_sync_points_in_wait_list* Specify synchronization-points that need to complete before this particular command can be executed.

If *sync_point_wait_list* is `NULL`, *num_sync_points_in_wait_list* must be 0. If *sync_point_wait_list* is not `NULL`, the list of synchronization-points pointed to by *sync_point_wait_list* must be valid and *num_sync_points_in_wait_list* must be greater than 0. The synchronization-points specified in *sync_point_wait_list* are **device-side** synchronization-points. The command-buffer associated with synchronization-points in *sync_point_wait_list* must be the same as *command_buffer*. The memory associated with *sync_point_wait_list* can be reused or freed after the function returns.

sync_point Returns a synchronization-point ID that identifies this particular command. Synchronization-point objects are unique and can be used to identify this command later on. *sync_point* can be `NULL` in which case it will not be possible for the application to record a wait for this command to complete. If the *sync_point_wait_list* and the *sync_point* arguments are not `NULL`, the *sync_point* argument should not refer to an element of the *sync_point_wait_list* array.

mutable_handle Returns a handle to the command. Handle is unused by this extension and must be passed as `NULL`.

clCommandFillImageKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns the errors defined by **clEnqueueFillImage** except:

`CL_INVALID_COMMAND_QUEUE` is replaced with:

- `CL_INVALID_COMMAND_QUEUE` if *command_queue* is not `NULL`.

`CL_INVALID_CONTEXT` is replaced with:

- `CL_INVALID_CONTEXT` if the context associated with *command_queue*, *command_buffer*, and *image* are not the same.

`CL_INVALID_EVENT_WAIT_LIST` is replaced with:

- `CL_INVALID_SYNC_POINT_WAIT_LIST_KHR` if *sync_point_wait_list* is `NULL` and *num_sync_points_in_wait_list* is > 0, or *sync_point_wait_list* is not `NULL` and *num_sync_points_in_wait_list* is 0, or if synchronization-point objects in *sync_point_wait_list* are not valid synchronization-points.

New errors:

- `CL_INVALID_COMMAND_BUFFER_KHR` if *command_buffer* is not a valid command-buffer.
- `CL_INVALID_OPERATION` if *command_buffer* has been finalized.
- `CL_INVALID_VALUE` if *mutable_handle* is not `NULL`.

The function

```
cl_int clCommandNDRangeKernelKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    const cl_ndrange_kernel_command_properties_khr* properties,
    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_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);
```

Records a command to execute a kernel on a device.

> The work-group size to be used for *kernel* can also be specified in the program source using the `__attribute__((reqd_work_group_size(X, Y, Z)))` qualifier. In this case the size of work-group specified by *local_work_size* must match the value specified by the `reqd_work_group_size __attribute__` qualifier.
>
> These work-group instances are executed in parallel across multiple compute units or concurrently on the same compute unit.
>
> Each work-item is uniquely identified by a global identifier. The global ID, which can be read inside the kernel, is computed using the value given by *global_work_size* and *global_work_offset*. In addition, a work-item is also identified within a work-group by a unique local ID. The local ID, which can also be read by the kernel, is computed using the value given by *local_work_size*. The starting local ID is always (0, 0, … 0).

command_buffer Refers to a valid command-buffer object.

command_queue Specifies the command-queue the command will be recorded to. Parameter is unused by this extension as only a single command-queue is supported and **must** be `NULL`.

properties Specifies a list of properties for the kernel command and their corresponding values. Each property name is immediately followed by the corresponding desired value. The list is terminated with 0. If no properties are required, *properties* may be `NULL`. This extension does not define any properties.

kernel A valid kernel object which **must** have its arguments set. Any changes to *kernel* after calling **clCommandNDRangeKernelKHR**, such as with **clSetKernelArg** or **clSetKernelExecInfo**, have no effect on the recorded command. If *kernel* is recorded to a following **clCommandNDRangeKernelKHR** command however, then that command will capture the updated state of *kernel*.

work_dim, *global_work_offset*, *global_work_size*, *local_work_size* Refer to **clEnqueueNDRangeKernel**.

sync_point_wait_list, *num_sync_points_in_wait_list* Specify synchronization-points that need to complete before this particular command can be executed.

If *sync_point_wait_list* is NULL, *num_sync_points_in_wait_list* must be 0. If *sync_point_wait_list* is not NULL, the list of synchronization-points pointed to by *sync_point_wait_list* must be valid and *num_sync_points_in_wait_list* must be greater than 0. The synchronization-points specified in *sync_point_wait_list* are **device-side** synchronization-points. The command-buffer associated with synchronization-points in *sync_point_wait_list* must be the same as *command_buffer*. The memory associated with *sync_point_wait_list* can be reused or freed after the function returns.

sync_point Returns a synchronization-point ID that identifies this particular command. Synchronization-point objects are unique and can be used to identify this command later on. *sync_point* can be NULL in which case it will not be possible for the application to record a wait for this command to complete. If the *sync_point_wait_list* and the *sync_point* arguments are not NULL, the *sync_point* argument should not refer to an element of the *sync_point_wait_list* array.

mutable_handle Returns a handle to the command. Handle is unused by this extension and must be passed as NULL.

clCommandNDRangeKernelKHR returns CL_SUCCESS if the function is executed successfully. Otherwise, it returns the errors defined by **clEnqueueNDRangeKernel** except:

CL_INVALID_COMMAND_QUEUE is replaced with:

- CL_INVALID_COMMAND_QUEUE if *command_queue* is not NULL.

CL_INVALID_CONTEXT is replaced with:

- CL_INVALID_CONTEXT if the context associated with *command_queue*, *command_buffer*, and *kernel* are not the same.

CL_INVALID_EVENT_WAIT_LIST is replaced with:

- CL_INVALID_SYNC_POINT_WAIT_LIST_KHR if *sync_point_wait_list* is NULL and *num_sync_points_in_wait_list* is > 0, or *sync_point_wait_list* is not NULL and *num_sync_points_in_wait_list* is 0, or if synchronization-point objects in *sync_point_wait_list* are not valid synchronization-points.

New errors:

- CL_INVALID_COMMAND_BUFFER_KHR if *command_buffer* is not a valid command-buffer.
- CL_INVALID_VALUE if values specified in *properties* are not valid
- CL_INVALID_OPERATION if *command_buffer* has been finalized.
- CL_INVALID_VALUE if *mutable_handle* is not NULL.
- CL_INVALID_OPERATION if the device associated with *command_queue* does not support CL_COMMAND_BUFFER_CAPABILITY_KERNEL_PRINTF_KHR and *kernel* contains a printf call.
- CL_INVALID_OPERATION if the device associated with *command_queue* does not support CL_COMMAND_BUFFER_CAPABILITY_DEVICE_SIDE_ENQUEUE_KHR and *kernel* contains a kernel-enqueue call.

The function

```
cl_int clCommandSVMMemcpyKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    void* dst_ptr,
    const void* src_ptr,
    size_t size,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);
```

Records a command to do an SVM memcpy operation.

command_buffer Refers to a valid command-buffer object.

command_queue Specifies the command-queue the command will be recorded to. Parameter is unused by this extension as only a single command-queue is supported and **must** be `NULL`.

dst_ptr is the pointer to a host (if the device supports system SVM) or SVM memory allocation where data is copied to.

src_ptr is the pointer to a host (if the device supports system SVM) or SVM memory allocation where data is copied from.

size is the size in bytes of data being copied.

sync_point_wait_list, *num_sync_points_in_wait_list* Specify synchronization-points that need to complete before this particular command can be executed.

If *sync_point_wait_list* is `NULL`, *num_sync_points_in_wait_list* must be 0. If *sync_point_wait_list* is not `NULL`, the list of synchronization-points pointed to by *sync_point_wait_list* must be valid and *num_sync_points_in_wait_list* must be greater than 0. The synchronization-points specified in *sync_point_wait_list* are **device-side** synchronization-points. The command-buffer associated with synchronization-points in *sync_point_wait_list* must be the same as *command_buffer*. The memory associated with *sync_point_wait_list* can be reused or freed after the function returns.

sync_point Returns a synchronization-point ID that identifies this particular command. Synchronization-point objects are unique and can be used to identify this command later on. *sync_point* can be `NULL` in which case it will not be possible for the application to record a wait for this command to complete. If the *sync_point_wait_list* and the *sync_point* arguments are not `NULL`, the *sync_point* argument should not refer to an element of the *sync_point_wait_list* array.

mutable_handle Returns a handle to the command. Handle is unused by this extension and must be passed as `NULL`.

clCommandSVMMemcpyKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns the errors defined by **clEnqueueSVMMemcpy** except:

CL_INVALID_COMMAND_QUEUE is replaced with:

- CL_INVALID_COMMAND_QUEUE if *command_queue* is not NULL.

CL_INVALID_CONTEXT is replaced with:

- CL_INVALID_CONTEXT if the context associated with *command_queue*, *command_buffer*, and *kernel* are not the same.

CL_INVALID_EVENT_WAIT_LIST is replaced with:

- CL_INVALID_SYNC_POINT_WAIT_LIST_KHR if *sync_point_wait_list* is NULL and *num_sync_points_in_wait_list* is > 0, or *sync_point_wait_list* is not NULL and *num_sync_points_in_wait_list* is 0, or if synchronization-point objects in *sync_point_wait_list* are not valid synchronization-points.

New errors:

- CL_INVALID_COMMAND_BUFFER_KHR if *command_buffer* is not a valid command-buffer.
- CL_INVALID_OPERATION if *command_buffer* has been finalized.
- CL_INVALID_VALUE if *mutable_handle* is not NULL.

The function

```
cl_int clCommandSVMMemFillKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_queue command_queue,
    void* svm_ptr,
    const void* pattern,
    size_t pattern_size,
    size_t size,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle);
```

Records a command to fill a region in SVM with a pattern of a given pattern size.

command_buffer Refers to a valid command-buffer object.

command_queue Specifies the command-queue the command will be recorded to. Parameter is unused by this extension as only a single command-queue is supported and **must** be NULL.

svm_ptr is a pointer to a (if the device supports system SVM) or SVM memory region that will be filled with *pattern*. It must be aligned to *pattern_size* bytes. If *svm_ptr* is allocated using **clSVMAlloc** then it must be allocated from the same context from which *command_queue* was created. Otherwise the behavior is undefined.

pattern is a pointer to the data pattern of size *pattern_size* in bytes. *pattern* will be used to fill a region in *buffer* starting at *svm_ptr* and is *size* bytes in size. The data pattern must be a scalar or vector integer or floating-point data type supported by OpenCL. For example, if region pointed to by *svm_ptr* is to be filled with a pattern of float4 values, then *pattern* will be a pointer to a cl_float4 value and *pattern_size* will be `sizeof(cl_float4)`. The maximum value of *pattern_size* is the size of the largest integer or floating-point vector data type supported by the OpenCL device. The memory associated with *pattern* can be reused or freed after the function returns.

size is the size in bytes of region being filled starting with *svm_ptr* and must be a multiple of *pattern_size*.

sync_point_wait_list, *num_sync_points_in_wait_list* Specify synchronization-points that need to complete before this particular command can be executed.

If *sync_point_wait_list* is `NULL`, *num_sync_points_in_wait_list* must be 0. If *sync_point_wait_list* is not `NULL`, the list of synchronization-points pointed to by *sync_point_wait_list* must be valid and *num_sync_points_in_wait_list* must be greater than 0. The synchronization-points specified in *sync_point_wait_list* are **device-side** synchronization-points. The command-buffer associated with synchronization-points in *sync_point_wait_list* must be the same as *command_buffer*. The memory associated with *sync_point_wait_list* can be reused or freed after the function returns.

sync_point Returns a synchronization-point ID that identifies this particular command. Synchronization-point objects are unique and can be used to identify this command later on. *sync_point* can be `NULL` in which case it will not be possible for the application to record a wait for this command to complete. If the *sync_point_wait_list* and the *sync_point* arguments are not `NULL`, the *sync_point* argument should not refer to an element of the *sync_point_wait_list* array.

mutable_handle Returns a handle to the command. Handle is unused by this extension and must be passed as `NULL`.

clCommandSVMMemFillKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns the errors defined by **clEnqueueSVMMemFill** except:

`CL_INVALID_COMMAND_QUEUE` is replaced with:

- `CL_INVALID_COMMAND_QUEUE` if *command_queue* is not `NULL`.

`CL_INVALID_CONTEXT` is replaced with:

- `CL_INVALID_CONTEXT` if the context associated with *command_queue*, *command_buffer*, and *kernel* are not the same.

`CL_INVALID_EVENT_WAIT_LIST` is replaced with:

- `CL_INVALID_SYNC_POINT_WAIT_LIST_KHR` if *sync_point_wait_list* is `NULL` and *num_sync_points_in_wait_list* is > 0, or *sync_point_wait_list* is not `NULL` and *num_sync_points_in_wait_list* is 0, or if synchronization-point objects in *sync_point_wait_list* are not valid synchronization-points.

New errors:

- `CL_INVALID_COMMAND_BUFFER_KHR` if *command_buffer* is not a valid command-buffer.

- CL_INVALID_OPERATION if *command_buffer* has been finalized.
- CL_INVALID_VALUE if *mutable_handle* is not NULL.

46.8.5. Add new section "Section 5.X.4 - Command Buffer Queries"

The function

```
cl_int clGetCommandBufferInfoKHR(
    cl_command_buffer_khr command_buffer,
    cl_command_buffer_info_khr param_name,
    size_t param_value_size,
    void* param_value,
    size_t* param_value_size_ret);
```

Queries information about a command-buffer.

command_buffer Specifies the command-buffer being queried.

param_name Specifies the information to query.

param_value_size Specifies the size in bytes of memory pointed to by *param_value*. This size must be ≥ size of return type as described in the table below. If *param_value* is NULL, it is ignored.

param_value A pointer to memory where the appropriate result being queried is returned. If *param_value* is NULL, it is ignored.

param_value_size_ret Returns the actual size in bytes of data being queried by *param_value*. If *param_value_size_ret* is NULL, it is ignored.

The list of supported *param_name* values and the information returned in *param_value* by **clGetCommandBufferInfoKHR** is described in the table below.

Table 70. **clGetCommandBufferInfoKHR** *values*

Command Buffer Info	Return Type	Description
CL_COMMAND_BUFFER_NUM_QUEUES_KHR	cl_uint	The number of command-queues specified when *command_buffer* was created.
CL_COMMAND_BUFFER_QUEUES_KHR	cl_command_queue[]	Return the list of command-queues specified when the *command_buffer* was created.
CL_COMMAND_BUFFER_REFERENCE_COUNT_KHR [1]	cl_uint	Return the *command_buffer* reference count.

Command Buffer Info	Return Type	Description
CL_COMMAND_BUFFER_STATE_KHR	cl_command_buffer_state_khr	Return the state of *command_buffer*. CL_COMMAND_BUFFER_STATE_RECORDING_KHR is returned when *command_buffer* has not been finalized. CL_COMMAND_BUFFER_STATE_EXECUTABLE_KHR is returned when *command_buffer* has been finalized and there is not a Pending instance of *command_buffer* awaiting completion on a command_queue. CL_COMMAND_BUFFER_STATE_PENDING_KHR is returned when an instance of *command_buffer* has been enqueued for execution but not yet completed.

Command Buffer Info	Return Type	Description
CL_COMMAND_BUFFER_PROPERTIES_ARRAY_KHR	cl_command_buffer_properties_khr[]	Return the *properties* argument specified in **clCreateCommandBufferKHR**. If the *properties* argument specified in **clCreateCommandBufferKHR** used to create *command_buffer* was not NULL, the implementation must return the values specified in the properties argument. If the *properties* argument specified in **clCreateCommandBufferKHR** used to create *command_buffer* was NULL, the implementation may return either a *param_value_size_ret* of 0 (i.e. there is are no properties to be returned), or the implementation may return a property value of 0 (where 0 is used to terminate the properties list).
CL_COMMAND_BUFFER_CONTEXT_KHR	cl_context	Return the context associated with *command_buffer*.

clGetCommandBufferInfoKHR returns CL_SUCCESS if the function is executed successfully. Otherwise, it returns one of the following errors:

- CL_INVALID_COMMAND_BUFFER_KHR if *command_buffer* is not a valid command-buffer.
- CL_INVALID_VALUE if *param_name* is not one of the supported values or if size in bytes specified by *param_value_size* is less than size of return type and *param_value* is not a NULL value.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

46.9. Modifications to section 5.11 of the OpenCL API Specification

In the opening paragraph add **clEnqueueCommandBufferKHR** to list of commands that can return an event object.

Add to Table 37, *Event Command Types*:

Events Created By	Event Command Type
clEnqueueCommandBufferKHR	CL_COMMAND_COMMAND_BUFFER_KHR

46.10. Sample Code

```
#define CL_CHECK(ERROR)                             \
  if (ERROR) {                                      \
    std::cerr << "OpenCL error: " << ERROR << "\n"; \
    return ERROR;                                   \
  }

int main() {
  cl_platform_id platform;
  CL_CHECK(clGetPlatformIDs(1, &platform, nullptr));
  cl_device_id device;
  CL_CHECK(clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, 1, &device, nullptr));

  cl_int error;
  cl_context context =
      clCreateContext(nullptr, 1, &device, nullptr, nullptr, &error);
  CL_CHECK(error);

  const char* code = R"OpenCLC(
kernel void vector_addition(global int* tile1, global int* tile2,
                            global int* res) {
  size_t index = get_global_id(0);
  res[index] = tile1[index] + tile2[index];
}
)OpenCLC";
  const size_t length = std::strlen(code);

  cl_program program =
      clCreateProgramWithSource(context, 1, &code, &length, &error);
  CL_CHECK(error);

  CL_CHECK(clBuildProgram(program, 1, &device, nullptr, nullptr, nullptr));

  cl_kernel kernel = clCreateKernel(program, "vector_addition", &error);
  CL_CHECK(error);

  constexpr size_t frame_count = 60;
  constexpr size_t frame_elements = 1024;
  constexpr size_t frame_size = frame_elements * sizeof(cl_int);

  constexpr size_t tile_count = 16;
```

```
constexpr size_t tile_elements = frame_elements / tile_count;
constexpr size_t tile_size = tile_elements * sizeof(cl_int);

cl_mem buffer_tile1 =
    clCreateBuffer(context, CL_MEM_READ_ONLY, tile_size, nullptr, &error);
CL_CHECK(error);
cl_mem buffer_tile2 =
    clCreateBuffer(context, CL_MEM_READ_ONLY, tile_size, nullptr, &error);
CL_CHECK(error);
cl_mem buffer_res =
    clCreateBuffer(context, CL_MEM_WRITE_ONLY, tile_size, nullptr, &error);
CL_CHECK(error);

CL_CHECK(clSetKernelArg(kernel, 0, sizeof(buffer_tile1), &buffer_tile1));
CL_CHECK(clSetKernelArg(kernel, 1, sizeof(buffer_tile2), &buffer_tile2));
CL_CHECK(clSetKernelArg(kernel, 2, sizeof(buffer_res), &buffer_res));

cl_command_queue command_queue =
  clCreateCommandQueue(context, device,
                       CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE, &error);
CL_CHECK(error);

cl_command_buffer_khr command_buffer =
    clCreateCommandBufferKHR(1, &command_queue, nullptr, &error);
CL_CHECK(error);

cl_mem buffer_src1 =
    clCreateBuffer(context, CL_MEM_READ_ONLY, frame_size, nullptr, &error);
CL_CHECK(error);
cl_mem buffer_src2 =
    clCreateBuffer(context, CL_MEM_READ_ONLY, frame_size, nullptr, &error);
CL_CHECK(error);
cl_mem buffer_dst =
    clCreateBuffer(context, CL_MEM_WRITE_ONLY, frame_size, nullptr, &error);
CL_CHECK(error);

cl_sync_point_khr tile_sync_point = 0;
for (size_t tile_index = 0; tile_index < tile_count; tile_index++) {
  std::array<cl_sync_point_khr, 2> copy_sync_points;
  CL_CHECK(clCommandCopyBufferKHR(command_buffer,
      command_queue, buffer_src1, buffer_tile1, tile_index * tile_size, 0,
      tile_size, tile_sync_point ? 1 : 0,
      tile_sync_point ? &tile_sync_point : nullptr, &copy_sync_points[0]),
      nullptr);
  CL_CHECK(clCommandCopyBufferKHR(command_buffer,
      command_queue, buffer_src2, buffer_tile2, tile_index * tile_size, 0,
      tile_size, tile_sync_point ? 1 : 0,
      tile_sync_point ? &tile_sync_point : nullptr, &copy_sync_points[1]),
      nullptr);

  cl_sync_point_khr nd_sync_point;
```

```
    CL_CHECK(clCommandNDRangeKernelKHR(command_buffer,
        command_queue, nullptr, kernel, 1, nullptr, &tile_elements, nullptr,
        copy_sync_points.size(), copy_sync_points.data(), &nd_sync_point,
        nullptr));

    CL_CHECK(clCommandCopyBufferKHR(command_buffer,
        command_queue, buffer_res, buffer_dst, 0, tile_index * tile_size,
        tile_size, 1, &nd_sync_point, &tile_sync_point, nullptr));
  }

  CL_CHECK(clFinalizeCommandBufferKHR(command_buffer));

  std::random_device random_device;
  std::mt19937 random_engine{random_device()};
  std::uniform_int_distribution<cl_int> random_distribution{
     0, std::numeric_limits<cl_int>::max() / 2};
  auto random_generator = [&]() { return random_distribution(random_engine); };

  for (size_t frame_index = 0; frame_index < frame_count; frame_index++) {
    std::array<cl_event, 2> write_src_events;
    std::vector<cl_int> src1(frame_elements);
    std::generate(src1.begin(), src1.end(), random_generator);
    CL_CHECK(clEnqueueWriteBuffer(command_queue, buffer_src1, CL_FALSE, 0,
                                  frame_size, src1.data(), 0, nullptr,
                                  &write_src_events[0]));
    std::vector<cl_int> src2(frame_elements);
    std::generate(src2.begin(), src2.end(), random_generator);
    CL_CHECK(clEnqueueWriteBuffer(command_queue, buffer_src2, CL_FALSE, 0,
                                  frame_size, src2.data(), 0, nullptr,
                                  &write_src_events[1]));

    CL_CHECK(clEnqueueCommandBufferKHR(0, NULL, command_buffer, 2,
                                       write_src_events.data(), nullptr));

    CL_CHECK(clFinish(command_queue));

    CL_CHECK(clReleaseEvent(write_src_event[0]));
    CL_CHECK(clReleaseEvent(write_src_event[1]));
  }

  CL_CHECK(clReleaseCommandBufferKHR(command_buffer));
  CL_CHECK(clReleaseCommandQueue(command_queue));

  CL_CHECK(clReleaseMemObject(buffer_src1));
  CL_CHECK(clReleaseMemObject(buffer_src2));
  CL_CHECK(clReleaseMemObject(buffer_dst));

  CL_CHECK(clReleaseMemObject(buffer_tile1));
  CL_CHECK(clReleaseMemObject(buffer_tile2));
  CL_CHECK(clReleaseMemObject(buffer_res));
```

```
    CL_CHECK(clReleaseKernel(kernel));
    CL_CHECK(clReleaseProgram(program));
    CL_CHECK(clReleaseContext(context));

    return 0;
  }
```

46.11. Issues

1. Introduce a `clCloneCommandBufferKHR` entry-point for cloning a command-buffer.

   UNRESOLVED

2. Enable detached command-buffer execution, where command-buffers are executed on their own internal queue to prevent locking user created queues for the duration of their execution.

   UNRESOLVED

[1] The reference count returned should be considered immediately stale. It is unsuitable for general use in applications. This feature is provided for identifying memory leaks.

Chapter 47. Kernel Optimization Hints

This extension adds mechanisms to provide information to the compiler that may improve the performance of some kernels. Specifically, this extension adds the ability to:

- Tell the compiler the *expected* value of a variable.
- Allow the compiler to *assume* a condition is true.

These functions are not required for functional correctness.

The initial version of this extension extends the OpenCL SPIR-V environment to support new instructions for offline compilation tool chains. Similar functionality may be provided by some OpenCL C online compilation tool chains, but formal support in OpenCL C is not required by the initial version of the extension.

47.1. General Information

47.1.1. Name Strings

`cl_khr_expect_assume`

47.1.2. Version History

Date	Version	Description
2021-11-10	1.0.0	First assigned version.

47.1.3. Dependencies

This extension is written against the OpenCL Specifications Version V3.0.8.

The initial version of this extension extends the OpenCL SPIR-V environment to support new instructions. Please refer to the OpenCL SPIR-V Environment Specification that describes how this extension modifies the OpenCL SPIR-V environment.

47.2. Sample Code

Although this extension does not formally extend OpenCL C, the ability to provide *expect* and *assume* information is supported by many OpenCL C compiler tool chains. The sample code below describes how to test for and provide *expect* and *assume* information to compilers based on Clang:

```
// __has_builtin is an optional compiler feature that is supported by Clang.
// If this feature is not supported, we will assume the builtin is not present.
#ifndef __has_builtin
#define __has_builtin(x)    0
#endif

kernel void test(global int* dst, global int* src)
```

```
{
    int value = src[get_global_id(0)];

    // Tell the compiler that the most likely source value is zero.
#if __has_builtin(__builtin_expect)
    value = __builtin_expect(value, 0);
#endif

    // Tell the compiler that the source value is non-negative.
    // Behavior is undefined if the source value is actually negative.
#if __has_builtin(__builtin_assume)
    __builtin_assume(value >= 0);
#endif

    dst[get_global_id(0)] = value % 4;
}
```

Chapter 48. Sub-group Rotation

This extension adds support for a new sub-group data exchange operation that makes it possible to rotate values through the work items in a sub-group.

48.1. General Information

48.1.1. Name Strings

`cl_khr_subgroup_rotate`

48.1.2. Version History

Date	Version	Description
2022-04-22	1.0.0	Initial version.

48.1.3. Dependencies

This extension is written against the OpenCL Specification Version 3.0.10, and OpenCL C Specification Version 3.0.10 and OpenCL Environment Specification Version 3.0.10.

This extension requires OpenCL 2.0.

48.1.4. Contributors

Kévin Petit, Arm Ltd.
Ben Ashbaugh, Intel
Ruihao Zhang, Qualcomm
Sven van Haastregt, Arm Ltd.
Anastasia Stulova, Arm Ltd.
Stuart Brady, Arm Ltd.

48.2. New OpenCL C Functions

This extension adds the following built-in function:

```
gentype sub_group_rotate(gentype value, int delta)
gentype sub_group_clustered_rotate(gentype value, int delta, uint clustersize)
```

48.3. Modifications to the OpenCL C Specification

(Add a new section 6.15.x, Sub-group Rotation)

The following preprocessor definitions are added:

```
#define cl_khr_subgroup_rotate 1
```

The table below describes a specialized OpenCL C programming language built-in function that allow work items in a sub-group to exchange data. This function need not be encountered by all work items in a sub-group executing the kernel. For the functions below, the generic type name `gentype` may be one of the supported built-in scalar data types `char`, `uchar`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `float`, `double` (if double precision is supported), or `half` (if half precision is supported).

Function	Description
`gentype sub_group_rotate( gentype value, int delta)`	Returns *value* for the work item with sub-group local ID equal to the remainder of the division of the sum of this work item's sub-group local ID and *delta* by the maximum sub-group size. The value of *delta* is required to be dynamically-uniform for all work items in the sub-group, otherwise the behavior is undefined. The return value is undefined if the work item with sub-group local ID equal to the calculated index is inactive.
`gentype sub_group_clustered_rotate( gentype value, int delta, uint clustersize)`	Returns *value* for the work item with sub-group local ID equal to the sum of, the remainder of the division of the sum of this work item's ID within the cluster and *delta* by *clustersize*, and the sub-group local ID of the first work-item of the cluster to which the work-item executing the function belongs. The value of *delta* is required to be dynamically-uniform for all work items in the sub-group, otherwise the behavior is undefined. *clustersize* must be an integer constant expression and a power of two, smaller than or equal to the maximum sub-group size, otherwise the behavior is undefined. The return value is undefined if the work item with sub-group local ID equal to the calculated index is inactive.

48.4. Modifications to the OpenCL SPIR-V Environment Specification

See OpenCL SPIR-V Environment Specification.

48.5. Interactions with Other Extensions

If `cl_khr_il_program` is supported then the SPIR-V environment specification modifications described above apply.

Chapter 49. Work-group Uniform Arithmetic

This extension adds additional work-group collective functions to OpenCL C. Specifically, this extension adds support for work-group scans and reductions for the following operators:

- Logical operations (`and`, `or`, and `xor`).
- Bitwise operations (`and`, `or`, and `xor`).
- Integer multiplication (`mul`).
- Floating-point multiplication (`mul`).

49.1. General Information

49.1.1. Name Strings

`cl_khr_work_group_uniform_arithmetic`

49.1.2. Version History

Date	Version	Description
2022-04-29	1.0.0	Initial version.

49.1.3. Dependencies

This extension is written against the OpenCL Specification Version 3.0.10.

This extension requires OpenCL 2.0.

49.1.4. Contributors

Kevin Petit, Arm Ltd.
Ben Ashbaugh, Intel

49.2. New OpenCL C Functions

The following functions are added to OpenCL C.

```
int work_group_reduce_logical_and(int predicate);
int work_group_reduce_logical_or(int predicate);
int work_group_reduce_logical_xor(int predicate);

int work_group_scan_inclusive_logical_and(int predicate);
int work_group_scan_inclusive_logical_or(int predicate);
int work_group_scan_inclusive_logical_xor(int predicate);

int work_group_scan_exclusive_logical_and(int predicate);
int work_group_scan_exclusive_logical_or(int predicate);
```

```
int work_group_scan_exclusive_logical_xor(int predicate);
```

For the following functions, the generic type name `gentype` may be one of the supported built-in scalar data types `int`, `uint`, `long`, or `ulong`.

```
gentype work_group_reduce_and(gentype value);
gentype work_group_reduce_or(gentype value);
gentype work_group_reduce_xor(gentype value);

gentype work_group_scan_inclusive_and(gentype value);
gentype work_group_scan_inclusive_or(gentype value);
gentype work_group_scan_inclusive_xor(gentype value);

gentype work_group_scan_exclusive_and(gentype value);
gentype work_group_scan_exclusive_or(gentype value);
gentype work_group_scan_exclusive_xor(gentype value);
```

For the following functions, the generic type name `gentype` may be one of the supported built-in scalar data types `int`, `uint`, `long`, `ulong`, `float`, `double` (if double precision is supported), or `half` (if half precision is supported).

```
gentype work_group_reduce_mul(gentype value);
gentype work_group_scan_inclusive_mul(gentype value);
gentype work_group_scan_exclusive_mul(gentype value);
```

49.3. Modifications to the OpenCL C Specification

(Add to Section 6.15.16, Work-group Collective Functions)

The table below describes the OpenCL C programming language built-in functions that perform logical arithmetic operations across work items in a work-group. These functions must be encountered by all work items in a work-group executing the kernel, otherwise the behavior is undefined. For these functions, a non-zero *predicate* argument or return value is logically `true` and a zero *predicate* argument or return value is logically `false`.

Function	Description
`int work_group_reduce_logical_and(int predicate);` `int work_group_reduce_logical_or(int predicate);` `int work_group_reduce_logical_xor(int predicate);`	Returns the logical **and**, **or**, or **xor** of *predicate* for all work items in the work-group.

Function	Description
`int work_group_scan_inclusive_logical_and(int predicate);` `int work_group_scan_inclusive_logical_or(int predicate);` `int work_group_scan_inclusive_logical_xor(int predicate);`	Returns the result of an inclusive scan operation, which is the logical **and**, **or**, or **xor** of *predicate* for all work items in the work-group with a work-group linear local ID less than or equal to this work item's work-group linear local ID.
`int work_group_scan_exclusive_logical_and(int predicate);` `int work_group_scan_exclusive_logical_or(int predicate);` `int work_group_scan_exclusive_logical_xor(int predicate);`	Returns the result of an exclusive scan operation, which is the logical **and**, **or**, or **xor** of *predicate* for all work items in the work-group with a work-group linear local ID less than this work item's work-group linear local ID. If there is no work item in the work-group with a work-group linear local ID less than this work item's work-group linear local ID then an identity value `I` is returned. For **and**, the identity value is `true` (non-zero). For **or** and **xor**, the identity value is `false` (zero).

The table below describes the OpenCL C programming language built-in functions that perform bitwise integer operations across work items in a work-group. These functions must be encountered by all work items in a work-group executing the kernel, otherwise the behavior is undefined. For the functions below, the generic type name `gentype` may be one of the supported built-in scalar data types `int`, `uint`, `long`, and `ulong`.

Function	Description
`gentype work_group_reduce_and(gentype value);` `gentype work_group_reduce_or(gentype value);` `gentype work_group_reduce_xor(gentype value);`	Returns the bitwise **and**, **or**, or **xor** of *value* for all work items in the work-group.

Function	Description
`gentype work_group_scan_inclusive_and(gentype value);` `gentype work_group_scan_inclusive_or(gentype value);` `gentype work_group_scan_inclusive_xor(gentype value);`	Returns the result of an inclusive scan operation, which is the bitwise **and**, **or**, or **xor** of *value* for all work items in the work-group with a work-group linear local ID less than or equal to this work item's work-group linear local ID.
`gentype work_group_scan_exclusive_and(gentype value);` `gentype work_group_scan_exclusive_or(gentype value);` `gentype work_group_scan_exclusive_xor(gentype value);`	Returns the result of an exclusive scan operation, which is the bitwise **and**, **or**, or **xor** of *value* for all work items in the work-group with a work-group linear local ID less than this work item's work-group linear local ID. If there is no work item in the work-group with a work-group linear local ID less than this work item's work-group linear local ID then an identity value `I` is returned. For **and**, the identity value is `~0` (all bits set). For **or** and **xor**, the identity value is `0`.

The table below describes the OpenCL C programming language built-in functions that perform multiplicative operations across work items in a work-group. These functions must be encountered by all work items in a work-group executing the kernel, otherwise the behavior is undefined. For the functions below, the generic type name `gentype` may be one of the supported built-in scalar data types `int`, `uint`, `long`, `ulong`, `float`, `double` (if double precision is supported), or `half` (if half precision is supported).

Function	Description
`gentype work_group_reduce_mul(gentype value);`	Returns the multiplication of *value* for all work items in the work-group.

Function	Description
`gentype work_group_scan_inclusive_mul(gentype value);`	Returns the result of an inclusive scan operation which is the multiplication of *value* for all work items in the work-group with a work-group linear local ID less than or equal to this work item's work-group linear local ID.
`gentype work_group_scan_exclusive_mul(gentype value);`	Returns the result of an exclusive scan operation which is the multiplication of *value* for all work items in the work-group with a work-group linear local ID less than this work item's work-group linear local ID. If there is no work item in the work-group with a work-group linear local ID less than this work item's work-group linear local ID then the identity value `1` is returned.

49.4. Issues

1. For these built-in functions, do we only want to support the types supported by the existing work-group collective functions, or do we want to support the types supported by the sub-group collective functions?

   `RESOLVED`: The extension will require the same types as the existing work-group collective functions.

   The difference are the 8-bit and 16-bit types: `char`, `uchar`, `short`, and `ushort`. Note that `half` is already supported, if half-precision is supported.

Chapter 50. Command Buffers - Mutable Dispatch (Provisional)

This extension enables users to modify the configuration of kernel execution commands between command-buffer enqueues.

50.1. General Information

50.1.1. Name Strings

`cl_khr_command_buffer_mutable_dispatch`

50.1.2. Version History

Date	Version	Description
2022-08-31	0.9.0	First assigned version (provisional).

> This is a provisional OpenCL extension specification that has been Ratified under the Khronos Intellectual Property Framework. It is being made publicly available as a provisional extension to enable review and feedback from the community. While it is a provisional extension features may be added, removed, or changed in non-backward compatible ways.
>
> If you have feedback please create an issue on: https://github.com/KhronosGroup/OpenCL-Docs/

50.1.3. Dependencies

This extension requires the `cl_khr_command_buffer` extension version 0.9.0.

50.1.4. Contributors

Ewan Crawford, Codeplay Software Ltd.
Gordon Brown, Codeplay Software Ltd.
Kenneth Benzie, Codeplay Software Ltd.
Alastair Murray, Codeplay Software Ltd.
Jack Frankland, Codeplay Software Ltd.
Balaji Calidas, Qualcomm Technologies Inc.
Joshua Kelly, Qualcomm Technologies, Inc.
Kevin Petit, Arm Ltd.
Aharon Abramson, Intel.
Ben Ashbaugh, Intel.
Boaz Ouriel, Intel.
Pekka Jääskeläinen, Tampere University
Jan Solanti, Tampere University
Nikhil Joshi, NVIDIA

James Price, Google

50.2. Overview

The `cl_khr_command_buffer` extension separates command construction from enqueue by providing a mechanism to record a set of commands which can then be repeatedly enqueued. However, the commands recorded to the command-buffer are immutable between enqueues.

`cl_khr_command_buffer_mutable_dispatch` removes this restriction, in particular, this extension allows the configuration of a kernel execution command in a command-buffer, called a *mutable-dispatch*, to be modified. This allows inputs and outputs to the kernel, as well as work-item sizes and offsets, to change without having to re-record the entire command sequence in a new command-buffer.

50.3. Interactions with Other Extensions

The `cl_command_buffer_structure_type_khr` type has been added to this extension for the purpose of allowing expansion of mutable functionality in future extensions layered on top of `cl_khr_command_buffer_mutable_dispatch`. Any parameter that is a structure containing a `void* next` member **must** have a value of `next` that is either `NULL`, or is a pointer to a valid structure defined by `cl_khr_command_buffer_mutable_dispatch` or an extension layered on top. To be a valid structure in the pointer chain the first member of the structure **must** be a `cl_command_buffer_structure_type_khr` identifier for the structure being iterated through, and the second member a `void* next` pointer to the next structure in the chain.

> This approach is based on structure pointer chains in Vulkan, for more details see the "Valid Usage for Structure Pointer Chains" section of the Vulkan specification.

This is designed so that another extension layered on `cl_khr_command_buffer_mutable_dispatch` could allow modification of commands recorded to a command-buffer other than kernel execution commands. As all command recording entry-points return a `cl_mutable_command_khr` handle, and aspects like which `cl_mem` object a command uses could also be updated between enqueues of the command-buffer.

50.4. New Types

50.4.1. Mutable Command Types

Types for using mutable-commands objects from Section 5.X.5:

```
// Bitfield covering each aspect of a mutable-dispatch which can be updated
typedef cl_bitfield cl_mutable_dispatch_fields_khr;

// For querying mutable-command objects with clGetMutableCommandInfoKHR
typedef cl_uint cl_mutable_command_info_khr;

// Identifies the type of a structure to allow structure pointer chains
```

```
typedef cl_uint cl_command_buffer_structure_type_khr;
```

Struct type for setting kernel arguments normally passed using **clSetKernelArg** and **clSetKernelArgSVMPointer**:

```
typedef struct cl_mutable_dispatch_arg_khr {
    cl_uint         arg_index;
    size_t          arg_size;
    const void*     arg_value;
} cl_mutable_dispatch_arg_khr;
```

Struct type for setting kernel execution info normally passed using **clSetKernelExecInfo**:

```
typedef struct cl_mutable_dispatch_exec_info_khr {
    cl_uint         param_name;
    size_t          param_value_size;
    const void*     param_value;
} cl_mutable_dispatch_exec_info_khr;
```

> *param_name* is of type `cl_uint` rather than `cl_kernel_exec_info` so that the extension can be implemented on OpenCL 1.2 where the `cl_kernel_exec_info` typedef is unavailable.

Struct type passed to **clUpdateMutableCommandsKHR** for setting the kernel configuration of a mutable **clCommandNDRangeKernelKHR** command:

```
typedef struct cl_mutable_dispatch_config_khr {
    cl_command_buffer_structure_type_khr        type;
    const void*                                 next;
    cl_mutable_command_khr                      command;
    cl_uint                                     num_args;
    cl_uint                                     num_svm_args;
    cl_uint                                     num_exec_infos;
    cl_uint                                     work_dim;
    const cl_mutable_dispatch_arg_khr*          arg_list;
    const cl_mutable_dispatch_arg_khr*          arg_svm_list;
    const cl_mutable_dispatch_exec_info_khr*    exec_info_list;
    const size_t*                               global_work_offset;
    const size_t*                               global_work_size;
    const size_t*                               local_work_size;
} cl_mutable_dispatch_config_khr;
```

type Type of this structure, must be `CL_STRUCTURE_TYPE_MUTABLE_DISPATCH_CONFIG_KHR`.

next Is `NULL` or a pointer to an extending structure.

command A mutable-command object returned by **clCommandNDRangeKernelKHR** representing a kernel execution as part of a command-buffer.

num_args Is the number of kernel arguments being changed.

num_svm_args Is the number of SVM kernel arguments being changed.

num_exec_infos Is the number of kernel execution info objects to set for this dispatch.

work_dim Is the number of dimensions used to specify the global work-items and work-items in the work-group. See **clEnqueueNDRangeKernel** for valid usage.

arg_list Is an array describing the new kernel arguments for this enqueue. It must contain *num_args* array elements, each of which encapsulates parameters passed to **clSetKernelArg**. See **clSetKernelArg** for usage of `cl_mutable_dispatch_arg_khr` members.

arg_svm_list is an array describing the new SVM kernel arguments for this enqueue. It must contain *num_svm_args* array elements, each of which encapsulates parameters passed to **clSetKernelArgSVMPointer**. See **clSetKernelArgSVMPointer** for usage of `cl_mutable_dispatch_arg_khr` members, `arg_size` is ignored.

exec_info_list Is an array containing *num_exec_infos* elements specifying the list of execution info objects use for this command-buffer enqueue. See **clSetKernelExecInfo** for usage of `cl_mutable_dispatch_exec_info_khr` members.

global_work_offset Can be used to specify an array of *work_dim* unsigned values that describe the offset used to calculate the global ID of a work-item. If *global_work_offset* is `NULL` then the global offset of the dispatch is not changed. See **clEnqueueNDRangeKernel** for valid usage.

global_work_size Points to an array of *work_dim* unsigned values that describe the number of global work-items in *work_dim* dimensions that will execute the kernel function. If *global_work_size* is `NULL` then the number of global work-items in the dispatch is not changed. See **clEnqueueNDRangeKernel** for valid usage.

local_work_size Points to an array of *work_dim* unsigned values that describe the number of work-items that make up a work-group that will execute the kernel. If *local_work_size* is `NULL` then the number of local work-items in the dispatch is not changed. See **clEnqueueNDRangeKernel** for valid usage.

```
typedef struct _cl_mutable_base_config_khr {
    cl_command_buffer_structure_type_khr type,
    const void* next,
    cl_uint num_mutable_dispatch,
    const cl_mutable_dispatch_config_khr* mutable_dispatch_list
} cl_mutable_base_config_khr;
```

type Type of this structure, must be `CL_STRUCTURE_TYPE_MUTABLE_BASE_CONFIG_KHR`

next Is `NULL` or a pointer to an extending structure.

num_mutable_dispatch Is the number of mutable-dispatch objects to configure in this enqueue of the command-buffer.

mutable_dispatch_list Is an array containing *num_mutable_dispatch* elements describing the configurations of mutable kernel execution commands in the command-buffer. For a description of struct members making up each array element see `cl_mutable_dispatch_config_khr`.

50.5. New API Functions

Mutable-handle entry points from Section 5.X.5:

```
cl_int clUpdateMutableCommandsKHR(
    cl_command_buffer_khr command_buffer,
    const cl_mutable_base_config_khr* mutable_config);

cl_int clGetMutableCommandInfoKHR(
    cl_mutable_command_khr command,
    cl_mutable_command_info_khr param_name,
    size_t param_value_size,
    void* param_value,
    size_t* param_value_size_ret);
```

50.6. New API Enums

Enums for working with mutable-command objects from Section 5.X.5:

```
// Error code
CL_INVALID_MUTABLE_COMMAND_KHR                     -1141

// Accepted values for the param_name parameter to clGetDeviceInfo
CL_DEVICE_MUTABLE_DISPATCH_CAPABILITIES_KHR        0x12B0

// Property to cl_ndrange_kernel_command_properties_khr
CL_MUTABLE_DISPATCH_UPDATABLE_FIELDS_KHR           0x12B1

// Bits for cl_mutable_dispatch_fields_khr bitfield
CL_MUTABLE_DISPATCH_GLOBAL_OFFSET_KHR              (0x1 << 0)
CL_MUTABLE_DISPATCH_GLOBAL_SIZE_KHR                (0x1 << 1)
CL_MUTABLE_DISPATCH_LOCAL_SIZE_KHR                 (0x1 << 2)
CL_MUTABLE_DISPATCH_ARGUMENTS_KHR                  (0x1 << 3)
CL_MUTABLE_DISPATCH_EXEC_INFO_KHR                  (0x1 << 4)

// cl_mutable_command_info_khr
CL_MUTABLE_COMMAND_COMMAND_QUEUE_KHR               0x12A0
CL_MUTABLE_COMMAND_COMMAND_BUFFER_KHR              0x12A1
CL_MUTABLE_DISPATCH_PROPERTIES_ARRAY_KHR           0x12A2
CL_MUTABLE_DISPATCH_KERNEL_KHR                     0x12A3
CL_MUTABLE_DISPATCH_DIMENSIONS_KHR                 0x12A4
```

```
CL_MUTABLE_DISPATCH_GLOBAL_WORK_OFFSET_KHR          0x12A5
CL_MUTABLE_DISPATCH_GLOBAL_WORK_SIZE_KHR            0x12A6
CL_MUTABLE_DISPATCH_LOCAL_WORK_SIZE_KHR             0x12A7
CL_MUTABLE_COMMAND_COMMAND_TYPE_KHR                 0x12AD

// Bits for cl_command_buffer_flags_khr
CL_COMMAND_BUFFER_MUTABLE_KHR                     (0x1 << 1)
```

Enum values for `cl_command_buffer_structure_type_khr` allowing the structure types used for mutating commands between enqueues to be extended by future extensions built on top of `cl_khr_command_buffer_mutable_dispatch`. Based on structure pointer chains in Vulkan.

```
CL_STRUCTURE_TYPE_MUTABLE_BASE_CONFIG_KHR           0
CL_STRUCTURE_TYPE_MUTABLE_DISPATCH_CONFIG_KHR       1
```

50.7. Modifications to section 4.2 of the OpenCL API Specification

Add to **Table 5**, *Device Queries*, of section 4.2:

Table 5. List of supported param_names by **clGetDeviceInfo**

Device Info	Return Type	Description
`CL_DEVICE_MUTABLE_DISPATCH_CAPABILITIES_KHR`	`cl_mutable_dispatch_fields_khr`	Describes device mutable-dispatch capabilities, encoded as bits in a bitfield. Supported capabilities are: `CL_MUTABLE_DISPATCH_GLOBAL_OFFSET_KHR` Device supports the ability to modify the *global_work_offset* of kernel execution after command recording. `CL_MUTABLE_DISPATCH_GLOBAL_SIZE_KHR` Device supports the ability to modify the *global_work_size* of kernel execution after command recording. `CL_MUTABLE_DISPATCH_LOCAL_SIZE_KHR` Device supports the ability to modify the *local_work_size* of kernel execution after command recording. `CL_MUTABLE_DISPATCH_ARGUMENTS_KHR` Device supports the ability to modify arguments set on a kernel after command recording. `CL_MUTABLE_DISPATCH_EXEC_INFO_KHR` Device supports the ability to modify execution information set on a kernel after command recording.

50.8. Modifications to Section 5.X - Command Buffers of the OpenCL API Specification

50.8.1. Modifications to clCreateCommandBufferKHR

Modify the `CL_COMMAND_BUFFER_FLAGS_KHR` property in the clCreateCommandBufferKHR properties table to introduce a new flag to the bitfield. The following text is now included in the description of property values.

Recording Properties	Property Value	Description
`CL_COMMAND_BUFFER_FLAGS_KHR`	`cl_command_buffer_flags_khr`	`CL_COMMAND_BUFFER_MUTABLE_KHR` - Enables modification of the command-buffer, by default command-buffers are immutable. If set, commands in the command-buffer may be updated via **clUpdateMutableCommandsKHR**.

50.8.2. Modifications to clCommandNDRangeKernelKHR

50.8.2.1. Properties Parameter

Description of the *properties* parameter is changed to:

properties Specifies a list of properties for the kernel command and their corresponding values. Each property name is immediately followed by the corresponding desired value. The list is terminated with 0. If a supported property and its value is not specified in *properties*, its default value will be used. *properties* may be `NULL` in which case the default values for supported properties will be used. The list of supported properties is described in the table below.

Table 71. **clCommandNDRangeKernelKHR** *properties*

Recording Properties	Property Value	Description
CL_MUTABLE_DISPATCH_UPDATABLE_FIELDS_KHR	cl_mutable_dispatch_fields_khr	This is a bitfield and can be set to a combination of the following values: CL_MUTABLE_DISPATCH_GLOBAL_OFFSET_KHR Determines whether the *global_work_offset* of kernel execution can be modified after recording. If set, the *global_work_offset* of the kernel execution can be changed with **clUpdateMutableCommandsKHR** using the cl_mutable_dispatch_config_khr field of the *mutable_config* parameter. Otherwise, the *global_work_offset* cannot be modified. CL_MUTABLE_DISPATCH_GLOBAL_SIZE_KHR Determines whether the *global_work_size* of kernel execution can be modified after recording. If set, the *global_work_size* of the kernel execution can be changed with **clUpdateMutableCommandsKHR** using the cl_mutable_dispatch_config_khr field of the *mutable_config* parameter. Otherwise, the *global_work_size* cannot be modified. CL_MUTABLE_DISPATCH_LOCAL_SIZE_KHR Determines whether the *local_work_size* of kernel execution can be modified after recording. If set, the *local_work_size* of the kernel execution can be changed with **clUpdateMutableCommandsKHR** using the cl_mutable_dispatch_config_khr field of the *mutable_config* parameter. Otherwise, the *local_work_size* cannot be modified. CL_MUTABLE_DISPATCH_ARGUMENTS_

50.8.2.2. Mutable Handle Parameter

Description of the *mutable_handle* parameter is changed to:

mutable_handle Returns a handle to the command that can be used in the `cl_mutable_dispatch_config_khr` struct to update the command configuration between recordings, may be `NULL`. The lifetime of this handle is tied to the parent command-buffer, such that freeing the command-buffer will also free this handle.

50.8.2.3. Additional Errors

The error condition:

- `CL_INVALID_OPERATION` if *mutable_handle* is not `NULL`.

Is replaced with

- `CL_INVALID_OPERATION` if the requested `CL_MUTABLE_DISPATCH_UPDATABLE_FIELDS_KHR` properties are not reported by `CL_DEVICE_MUTABLE_DISPATCH_CAPABILITIES_KHR` for the device associated with *command_queue*. If *command_queue* is `NULL`, the device associated with *command_buffer* must report support for these properties.

50.8.3. New Section in the OpenCL API specification 5.X.5 - Mutable Commands:

A generic `cl_mutable_command_khr` handle is called a *mutable-command* object as it can be returned from any command recording entry-point in the `cl_khr_command_buffer` family of extensions. The mutable-command handles returned by **clCommandNDRangeKernelKHR** in particular are referred to as *mutable-dispatch* objects, and can be modified through the fields of `cl_mutable_dispatch_config_khr`.

Mutable-command handles are updated between enqueues using entry-point **clUpdateMutableCommandsKHR**. To enable performant usage, all aspects of mutation are encapsulated inside a single `cl_mutable_base_config_khr` parameter. This means that the runtime has access to all the information about how the command-buffer will change, allowing the command-buffer to be rebuilt as efficiently as possible. Any modifications to the arguments or execution info of a mutable-dispatch handle using `cl_mutable_dispatch_arg_khr` or `cl_mutable_dispatch_exec_info_khr` have no affect on the original kernel object used when the command was recorded, and only influence the **clCommandNDRangeKernelKHR** command associated with the mutable-dispatch.

> The base `cl_khr_command_buffer` extension notes that a command-buffer does not update the reference count of objects set as arguments on kernels recorded into the command-buffer.
>
> The implications for applications using **clUpdateMutableCommandsKHR** is that it is safe to delete objects used as kernel command arguments, if all the kernel commands using that object as an argument have had their arguments replaced with a different object.

To facilitate performant usage for pipelined work flows, where applications repeatedly call command-buffer update then enqueue, implementations may defer some of the work to allow **clUpdateMutableCommandsKHR** to return immediately. Deferring any recompilation until **clEnqueueCommandBufferKHR** avoids blocking in host code and keeps device occupancy high. This is only possible with a command-buffer created with the CL_COMMAND_BUFFER_SIMULTANEOUS_USE_KHR flag, as without this the enqueued command-buffer must complete before any modification occurs.

The function

```
cl_int clUpdateMutableCommandsKHR(
    cl_command_buffer_khr command_buffer,
    const cl_mutable_base_config_khr* mutable_config);
```

Modifies the configuration of mutable-command handles returned during *command_buffer* recording, updating the behavior of those commands in future enqueues of *command_buffer*. Using this function when *command_buffer* is in the pending state and not created with the CL_COMMAND_BUFFER_SIMULTANEOUS_USE_KHR flag causes undefined behavior.

> Performant usage is to call **clUpdateMutableCommandsKHR** only when the desired state of all commands is known, rather than iteratively updating each command individually.

command_buffer Refers to a valid command-buffer object.

mutable_config Is a pointer to a cl_mutable_base_config_khr structure defining updates to make to mutable-commands.

clUpdateMutableCommandsKHR returns CL_SUCCESS if all the mutable-command objects were updated successfully. Otherwise, none of the updates to mutable-command objects are preserved and one of the errors below is returned:

- CL_INVALID_COMMAND_BUFFER_KHR if *command_buffer* is not a valid command-buffer.
- CL_INVALID_OPERATION if *command_buffer* has not been finalized.
- CL_INVALID_OPERATION if *command_buffer* was not created with the CL_COMMAND_BUFFER_MUTABLE_KHR flag.
- CL_INVALID_VALUE if the *type* member of *mutable_config* is not CL_STRUCTURE_TYPE_MUTABLE_BASE_CONFIG_KHR.
- CL_INVALID_VALUE if the *mutable_dispatch_list* member of *mutable_config* is NULL and *num_mutable_dispatch* > 0, or *mutable_dispatch_list* is not NULL and *num_mutable_dispatch* is 0.
- CL_INVALID_VALUE if the *next* member of *mutable_config* is not NULL and any iteration of the structure pointer chain does not contain valid *type* and *next* members.
- CL_INVALID_VALUE if *mutable_config* is NULL, or if both *next* and *mutable_dispatch_list* members of *mutable_config* are NULL.
- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL

implementation on the device.

- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

If the *mutable_dispatch_list* member of *mutable_config* is non-NULL, then errors defined by **clEnqueueNDRangeKernel**, **clSetKernelExecInfo**, **clSetKernelArg**, and **clSetKernelArgSVMPointer** are returned by **clUpdateMutableCommandsKHR** if any of the array elements are set to an invalid value. Additionally, the following errors are returned if any cl_mutable_dispatch_config_khr element of the array violates the defined conditions:

- CL_INVALID_MUTABLE_COMMAND_KHR if *command* is not a valid mutable command object, or created from *command_buffer*.
- CL_INVALID_VALUE if *type* is not CL_STRUCTURE_TYPE_MUTABLE_DISPATCH_CONFIG_KHR.
- CL_INVALID_OPERATION if values of *local_work_size* and/or *global_work_size* result in an increase to the number of work-groups in the ND-range.
- CL_INVALID_OPERATION if the values of *local_work_size* and/or *global_work_size* result in a change to work-group uniformity.
- CL_INVALID_OPERATION if the *work_dim* is different from the *work_dim* set on *command* recording.
- CL_INVALID_OPERATION if the CL_MUTABLE_DISPATCH_GLOBAL_OFFSET_KHR property was not set on *command* recording and *global_work_offset* is not NULL.
- CL_INVALID_OPERATION if the CL_MUTABLE_DISPATCH_GLOBAL_SIZE_KHR property was not set on *command* recording and *global_work_size* is not NULL.
- CL_INVALID_OPERATION if the CL_MUTABLE_DISPATCH_LOCAL_SIZE_KHR property was not set on *command* recording and *local_work_size* is not NULL.
- CL_INVALID_OPERATION if the CL_MUTABLE_DISPATCH_ARGUMENTS_KHR property was not set on *command* recording and *num_args* or *num_svm_args* is non-zero.
- CL_INVALID_OPERATION if the CL_MUTABLE_DISPATCH_EXEC_INFO_KHR property was not set on *command* recording and *num_exec_infos* is non-zero.
- CL_INVALID_VALUE if *arg_list* is NULL and *num_args* > 0, or *arg_list* is not NULL and *num_args* is 0.
- CL_INVALID_VALUE if *arg_svm_list* is NULL and *num_svm_args* > 0, or *arg_svm_list* is not NULL and *num_svm_args* is 0.
- CL_INVALID_VALUE if *exec_info_list* is NULL and *num_exec_infos* > 0, or *exec_info_list* is not NULL and *num_exec_infos* is 0.

The function

```
cl_int clGetMutableCommandInfoKHR(
    cl_mutable_command_khr command,
    cl_mutable_command_info_khr param_name,
    size_t param_value_size,
    void* param_value,
    size_t* param_value_size_ret);
```

Queries information about the *command* object.

command Specifies the mutable-command object being queried.

param_name Specifies the information to query. The list of supported *param_name* types and the information returned in *param_value* by **clGetMutableCommandInfoKHR** is described in the Mutable Command Object Queries table.

param_value_size Is used to specify the size in bytes of memory pointed to by *param_value*. This size must be ≥ size of return type as described in the Mutable Command Object Queries table.

param_value Is a pointer to memory where the appropriate result being queried is returned. If *param_value* is `NULL`, it is ignored.

param_value_size_ret Returns the actual size in bytes of data being queried by *param_name*. If *param_value_size_ret* is `NULL`, it is ignored.

Table 72. Mutable Command Object Queries

Mutable Command Info	Return Type	Description
`CL_MUTABLE_COMMAND_COMMAND_QUEUE_KHR`	`cl_command_queue`	Return the command-queue associated with *command*. If `NULL` was passed as the queue when *command* was recorded, then the queue associated with the command-buffer that *command* belongs to is returned.
`CL_MUTABLE_COMMAND_COMMAND_BUFFER_KHR`	`cl_command_buffer_khr`	Return the command-buffer associated with *command*.
`CL_MUTABLE_COMMAND_COMMAND_TYPE_KHR`	`cl_command_type`	Return the command-type associated with *command*. The list of supported event command types defined by **clGetEventInfo** is used with the matching command.

Mutable Command Info	Return Type	Description
CL_MUTABLE_DISPATCH_PROPERTIES_ARRAY_KHR	cl_ndrange_kernel_command_properties_khr[]	Return the properties argument specified on *command* recording with **clCommandNDRangeKernelKHR**. If the properties argument specified on creation of *command* was not NULL, the implementation must return the values specified in the properties argument in the same order and without including additional properties. If the properties argument specified on creation of *command* was NULL, or *command* was not recorded from a **clCommandNDRangeKernelKHR** command, the implementation must return *param_value_size_ret* equal to 0, indicating that there are no properties to be returned.
CL_MUTABLE_DISPATCH_KERNEL_KHR	cl_kernel	Return the kernel associated with *command* when recorded with **clCommandNDRangeKernelKHR**. If *command* was not recorded from a **clCommandNDRangeKernelKHR** command, the implementation must return *param_value_size_ret* equal to 0, indicating that the value returned in *param_value* is not valid.
CL_MUTABLE_DISPATCH_DIMENSIONS_KHR	cl_uint	Return the number of work-item dimensions specified when *command* was created. If *command* was not recorded from a **clCommandNDRangeKernelKHR** command, the implementation must return *param_value_size_ret* equal to 0, indicating that the value returned in *param_value* is not valid.

Mutable Command Info	Return Type	Description
CL_MUTABLE_DISPATCH_GLOBAL_WORK_OFFSET_KHR	size_t[]	Return the global work-item offset set on *command* creation, or from the most recent update via **clUpdateMutableCommandsKHR** where this value was modified. The output array contains *work_dim* values, where *work_dim* is returned by the query CL_MUTABLE_DISPATCH_DIMENSIONS_KHR. If a global work-item offset was not set, zero is returned for each element in the array. If *command* was not recorded from a **clCommandNDRangeKernelKHR** command, the implementation must return *param_value_size_ret* equal to 0, indicating that the value returned in *param_value* is not valid.
CL_MUTABLE_DISPATCH_GLOBAL_WORK_SIZE_KHR	size_t[]	Return the global work-item size set on *command* creation, or from the most recent update via **clUpdateMutableCommandsKHR** where this value was modified. The output array contains *work_dim* values, where *work_dim* is returned by the query CL_MUTABLE_DISPATCH_DIMENSIONS_KHR. If a global work-item size was not set, zero is returned for each element in the array. If *command* was not recorded from a **clCommandNDRangeKernelKHR** command, the implementation must return *param_value_size_ret* equal to 0, indicating that the value returned in *param_value* is not valid.
CL_MUTABLE_DISPATCH_LOCAL_WORK_SIZE_KHR	size_t[]	Return the local work-item size set on *command* creation, or from the most recent update via **clUpdateMutableCommandsKHR** where this value was modified. The output array contains *work_dim* values, where *work_dim* is returned by the query CL_MUTABLE_DISPATCH_DIMENSIONS_KHR. If a local work-item size was not set, zero is returned for each element in the array. If *command* was not recorded from a **clCommandNDRangeKernelKHR** command, the implementation must return *param_value_size_ret* equal to 0, indicating that the value returned in *param_value* is not valid.

clGetMutableCommandInfoKHR returns `CL_SUCCESS` if the function is executed successfully. Otherwise, it returns one of the following errors:

- `CL_INVALID_VALUE` if *param_name* is not valid, or if size in bytes specified by *param_value_size* is < size of return type as described in the Mutable Command Object Queries table and *param_value* is not `NULL`.
- `CL_INVALID_MUTABLE_COMMAND_KHR` if *command* is not a valid mutable command object.
- `CL_OUT_OF_RESOURCES` if there is a failure to allocate resources required by the OpenCL implementation on the device.
- `CL_OUT_OF_HOST_MEMORY` if there is a failure to allocate resources required by the OpenCL implementation on the host.

50.9. Sample Code

Sample application updating the arguments to a mutable-dispatch between command-buffer submissions.

```
#define CL_CHECK(ERROR)                             \
  if (ERROR) {                                      \
    std::cerr << "OpenCL error: " << ERROR << "\n"; \
    return ERROR;                                   \
  }

int main() {
  cl_platform_id platform;
  CL_CHECK(clGetPlatformIDs(1, &platform, nullptr));
  cl_device_id device;
  CL_CHECK(clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, 1, &device, nullptr));

  cl_mutable_dispatch_fields_khr mutable_capabilities;
  CL_CHECK(clGetDeviceInfo(device, CL_DEVICE_MUTABLE_DISPATCH_CAPABILITIES_KHR,
                           sizeof(mutable_capabilities), &mutable_capabilities,
                           nullptr));
  if (!(mutable_capabilities & CL_MUTABLE_DISPATCH_ARGUMENTS_KHR)) {
    std::cerr
        << "Device does not support update arguments to a mutable-dispatch, "
           "skipping example.\n";
    return 0;
  }

  cl_int error;
  cl_context context =
      clCreateContext(nullptr, 1, &device, nullptr, nullptr, &error);
  CL_CHECK(error);

  const char* code = R"OpenCLC(
kernel void vector_addition(global int* tile1, global int* tile2,
                            global int* res) {
```

```
    size_t index = get_global_id(0);
    res[index] = tile1[index] + tile2[index];
  }
  )OpenCLC";
    const size_t length = std::strlen(code);

    cl_program program =
        clCreateProgramWithSource(context, 1, &code, &length, &error);
    CL_CHECK(error);

    CL_CHECK(clBuildProgram(program, 1, &device, nullptr, nullptr, nullptr));

    cl_kernel kernel = clCreateKernel(program, "vector_addition", &error);
    CL_CHECK(error);

    // Set the parameters of the frames
    constexpr size_t iterations = 60;
    constexpr size_t elem_size = sizeof(cl_int);
    constexpr size_t frame_width = 32;
    constexpr size_t frame_count = frame_width * frame_width;
    constexpr size_t frame_size = frame_count * elem_size;

    cl_mem input_A_buffers[2] = {nullptr, nullptr};
    cl_mem input_B_buffers[2] = {nullptr, nullptr};
    cl_mem output_buffers[2] = {nullptr, nullptr};

    // Create the buffer to swap between even and odd kernel iterations
    for (size_t i = 0; i < 2; i++) {
      input_A_buffers[i] =
          clCreateBuffer(context, CL_MEM_READ_ONLY, frame_size, nullptr, &error);
      CL_CHECK(error);

      input_B_buffers[i] =
          clCreateBuffer(context, CL_MEM_READ_ONLY, frame_size, nullptr, &error);
      CL_CHECK(error);

      output_buffers[i] =
          clCreateBuffer(context, CL_MEM_WRITE_ONLY, frame_size, nullptr, &error);
      CL_CHECK(error);
    }

    cl_command_queue command_queue =
        clCreateCommandQueue(context, device, 0, &error);
    CL_CHECK(error);

    // Create command-buffer with mutable flag so we can update it
    cl_command_buffer_properties_khr properties[3] = {
        CL_COMMAND_BUFFER_FLAGS_KHR, CL_COMMAND_BUFFER_MUTABLE_KHR, 0};
    cl_command_buffer_khr command_buffer =
        clCreateCommandBufferKHR(1, &command_queue, properties, &error);
    CL_CHECK(error);
```

```
CL_CHECK(clSetKernelArg(kernel, 0, sizeof(cl_mem), &input_A_buffers[0]));
CL_CHECK(clSetKernelArg(kernel, 1, sizeof(cl_mem), &input_B_buffers[0]));
CL_CHECK(clSetKernelArg(kernel, 2, sizeof(cl_mem), &output_buffers[0]));

// Instruct the nd-range command to allow for mutable kernel arguments
cl_ndrange_kernel_command_properties_khr mutable_properties[] = {
    CL_MUTABLE_DISPATCH_UPDATABLE_FIELDS_KHR,
    CL_MUTABLE_DISPATCH_ARGUMENTS_KHR, 0};

// Create command handle for mutating nd-range command
cl_mutable_command_khr command_handle = nullptr;

// Add the nd-range kernel command
error = clCommandNDRangeKernelKHR(
    command_buffer, command_queue, mutable_properties, kernel, 1, nullptr,
    &frame_count, nullptr, 0, nullptr, nullptr, &command_handle);
CL_CHECK(error);

CL_CHECK(clFinalizeCommandBufferKHR(command_buffer));

// Prepare for random input generation
std::random_device random_device;
std::mt19937 random_engine{random_device()};
std::uniform_int_distribution<cl_int> random_distribution{
    std::numeric_limits<cl_int>::min() / 2,
    std::numeric_limits<cl_int>::max() / 2};

// Iterate over each frame
for (size_t i = 0; i < iterations; i++) {
  // Set the buffers for the current frame
  cl_mem input_A_buffer = input_A_buffers[i % 2];
  cl_mem input_B_buffer = input_B_buffers[i % 2];
  cl_mem output_buffer = output_buffers[i % 2];

  // Generate input A data
  std::vector<cl_int> input_a(frame_count);
  std::generate(std::begin(input_a), std::end(input_a),
                [&]() { return random_distribution(random_engine); });

  // Write the generated data to the input A buffer
  error =
      clEnqueueWriteBuffer(command_queue, input_A_buffer, CL_FALSE, 0,
                           frame_size, input_a.data(), 0, nullptr, nullptr);
  CL_CHECK(error);

  // Generate input B data
  std::vector<cl_int> input_b(frame_count);
  std::generate(std::begin(input_b), std::end(input_b),
                [&]() { return random_distribution(random_engine); });
```

```
// Write the generated data to the input B buffer
error =
    clEnqueueWriteBuffer(command_queue, input_B_buffer, CL_FALSE, 0,
                         frame_size, input_b.data(), 0, nullptr, nullptr);
CL_CHECK(error);

// If not executing the first frame
if (i != 0) {
  // Configure the mutable configuration to update the kernel arguments
  cl_mutable_dispatch_arg_khr arg_0{0, sizeof(cl_mem), &input_A_buffer};
  cl_mutable_dispatch_arg_khr arg_1{1, sizeof(cl_mem), &input_B_buffer};
  cl_mutable_dispatch_arg_khr arg_2{2, sizeof(cl_mem), &output_buffer};
  cl_mutable_dispatch_arg_khr args[] = {arg_0, arg_1, arg_2};
  cl_mutable_dispatch_config_khr dispatch_config{
      CL_STRUCTURE_TYPE_MUTABLE_DISPATCH_CONFIG_KHR,
      nullptr,
      command_handle,
      3 /* num_args */,
      0 /* num_svm_arg */,
      0 /* num_exec_infos */,
      0 /* work_dim - 0 means no change to dimensions */,
      args /* arg_list */,
      nullptr /* arg_svm_list - nullptr means no change*/,
      nullptr /* exec_info_list */,
      nullptr /* global_work_offset */,
      nullptr /* global_work_size */,
      nullptr /* local_work_size */};
  cl_mutable_base_config_khr mutable_config{
      CL_STRUCTURE_TYPE_MUTABLE_BASE_CONFIG_KHR, nullptr, 1,
      &dispatch_config};

  // Update the command buffer with the mutable configuration
  error = clUpdateMutableCommandsKHR(command_buffer, &mutable_config);
  CL_CHECK(error);
}

// Enqueue the command buffer
error = clEnqueueCommandBufferKHR(0, nullptr, command_buffer, 0, nullptr,
                                  nullptr);
CL_CHECK(error);

// Allocate memory for the output data
std::vector<cl_int> output(frame_count);

// Read the output data from the output buffer
error = clEnqueueReadBuffer(command_queue, output_buffer, CL_TRUE, 0,
                            frame_size, output.data(), 0, nullptr, nullptr);
CL_CHECK(error);

// Flush and execute the read buffer
error = clFinish(command_queue);
```

```
    CL_CHECK(error);

    // Verify the results of the frame
    for (size_t i = 0; i < frame_count; ++i) {
      const cl_int result = input_a[i] + input_b[i];
      if (output[i] != result) {
        std::cerr << "Error: Incorrect result at index " << i << " - Expected "
                  << output[i] << " was " << result << std::endl;
        std::exit(1);
      }
    }
  }

  std::cout << "Result verified\n";

  CL_CHECK(clReleaseCommandBufferKHR(command_buffer));
  for (size_t i = 0; i < 2; i++) {
    CL_CHECK(clReleaseMemObject(input_A_buffers[i]));
    CL_CHECK(clReleaseMemObject(input_B_buffers[i]));
    CL_CHECK(clReleaseMemObject(output_buffers[i]));
  }
  CL_CHECK(clReleaseCommandQueue(command_queue));
  CL_CHECK(clReleaseKernel(kernel));
  CL_CHECK(clReleaseProgram(program));
  CL_CHECK(clReleaseContext(context));
  CL_CHECK(clReleaseDevice(device));
  return 0;
}
```

50.10. Issues

1. Include simpler, more user friendly, entry-points for updating kernel arguments?

   RESOLVED: Can be implemented in the ecosystem as a layer on top, if that layer proves popular then can be introduced, possibly as another extension on top.

2. Add a command-buffer clone entry-point for deep copying a command-buffer? Arguments could then be updated and both command-buffers used. Useful for techniques like double buffering.

   Resolved: In the use-case we're targeting a user would only have a handle to the original command-buffer, but not the clone, which may limit the usefulness of this capability. Additionally, an implementation could be complicated by non-trivial deep copying of the underlying objects contained in the command-buffer. As a result of this new entry-point being an additive change to the specification it is omitted, and if its functionality has demand later, it may be a introduced as a stand alone extension.

3. Introduce a `CL_MUTABLE_DISPATCH_ADDITIONAL_WORK_GROUPS_KHR` capability to allow the number of work-groups in kernel execution to be increased during update.

Resolved: Can be included in the final release of the extension if there is implementation coverage.

Chapter 51. Command Buffers - Multiple Devices (Provisional)

This extension enables users to record commands across multiple queues in the same command-buffer, providing execution of heterogeneous task graphs from command-queues associated with different devices.

51.1. General Information

51.1.1. Name Strings

`cl_khr_command_buffer_multi_device`

51.1.2. Version History

Date	Version	Description
2023-04-14	0.9.0	First assigned version (provisional).
2024-04-30	0.9.1	Added clCommandSVMMemcpyKHR and clCommandSVMMemFillKHR as affected functions (provisional).

This is a provisional OpenCL extension specification that has been Ratified under the Khronos Intellectual Property Framework. It is being made publicly available as a provisional extension to enable review and feedback from the community. While it is a provisional extension features may be added, removed, or changed in non-backward compatible ways.

If you have feedback please create an issue on: https://github.com/KhronosGroup/OpenCL-Docs/

51.1.3. Dependencies

This extension requires the `cl_khr_command_buffer` extension version 0.9.3.

51.1.4. Contributors

Ewan Crawford, Codeplay Software Ltd.
Gordon Brown, Codeplay Software Ltd.
Kenneth Benzie, Codeplay Software Ltd.
Alastair Murray, Codeplay Software Ltd.
Jack Frankland, Codeplay Software Ltd.
Balaji Calidas, Qualcomm Technologies Inc.
Joshua Kelly, Qualcomm Technologies, Inc.
Kevin Petit, Arm Ltd.
Aharon Abramson, Intel.

Ben Ashbaugh, Intel.
Boaz Ouriel, Intel.
Pekka Jääskeläinen, Tampere University and Intel.
Jan Solanti, Tampere University
Nikhil Joshi, NVIDIA
James Price, Google

51.2. Overview

The cl_khr_command_buffer extension separates command construction from enqueue by providing a mechanism to record a set of commands which can then be repeatedly enqueued. However, the commands in a command-buffer can only be recorded to a single command-queue specified on command-buffer creation.

cl_khr_command_buffer_multi_device extends the scope of a command-buffer to allow commands to be recorded across multiple queues in the same command-buffer, providing execution of heterogeneous task graphs from command-queues associated with different devices.

The ability for a user to deep copy an existing command-buffer so that the commands target a different device is also made possible by cl_khr_command_buffer_multi_device. Depending on platform support the mapping of commands to the new target device can be done either explicitly by the user, or automatically by the OpenCL runtime.

51.3. New Types

Bitfield for querying command-buffer capabilities of an OpenCL Platform with **clGetPlatformInfo**, see the platform queries table:

```
typedef cl_bitfield cl_platform_command_buffer_capabilities_khr
```

51.4. New API Functions

```
cl_command_buffer_khr clRemapCommandBufferKHR(
    cl_command_buffer_khr command_buffer,
    cl_bool automatic,
    cl_uint num_queues,
    const cl_command_queue* queues,
    cl_uint num_handles,
    const cl_mutable_command_khr* handles,
    cl_mutable_command_khr* handles_ret,
    cl_int* errcode_ret);
```

51.5. New API Enums

Enums for querying device command-buffer capabilities with **clGetDeviceInfo**, see the device queries table:

```
// Accepted values for the param_name parameter to clGetDeviceInfo
CL_DEVICE_COMMAND_BUFFER_NUM_SYNC_DEVICES_KHR      0x12AB
CL_DEVICE_COMMAND_BUFFER_SYNC_DEVICES_KHR          0x12AC

// Bits for cl_device_command_buffer_capabilities_khr bitfield
CL_COMMAND_BUFFER_CAPABILITY_MULTIPLE_QUEUE_KHR    (0x1 << 4)

// Bits for cl_command_buffer_flags_khr
CL_COMMAND_BUFFER_DEVICE_SIDE_SYNC_KHR             (0x1 << 2)
```

Enums for querying platform command-buffer capabilities with **clGetPlatformInfo**, see the platform queries table:

```
// Accepted values for the param_name parameter to clGetPlatformInfo
CL_PLATFORM_COMMAND_BUFFER_CAPABILITIES_KHR        0x0908

// Bits for cl_platform_command_buffer_capabilities_khr bitfield
CL_COMMAND_BUFFER_PLATFORM_UNIVERSAL_SYNC_KHR       (0x1 << 0)
CL_COMMAND_BUFFER_PLATFORM_REMAP_QUEUES_KHR        (0x1 << 1)
CL_COMMAND_BUFFER_PLATFORM_AUTOMATIC_REMAP_KHR     (0x1 << 2)
```

51.6. Modifications to section 4.1 of the OpenCL API Specification

Add to **Table 3**, *Platform Queries*,

Platform Info	Return Type	Description
CL_PLATFORM_COMMAND_BUFFER_CAPABILITIES_KHR	cl_platform_command_buffer_capabilities_khr	Describes platform command-buffer capabilities, encoded as bits in a bitfield. Supported capabilities are: CL_COMMAND_BUFFER_PLATFORM_UNIVERSAL_SYNC_KHR - Platform supports the ability to synchronize all commands in a command-buffer using sync-points, irrespective of the queue the individual commands are recorded to. CL_COMMAND_BUFFER_PLATFORM_REMAP_QUEUES_KHR - Platform supports the ability to create a deep copy of an existing command-buffer with the commands explicitly remapped to different, potentially incompatible, queues. CL_COMMAND_BUFFER_PLATFORM_AUTOMATIC_REMAP_KHR - Platform supports the ability to create a remapped command-buffer where the mapping of commands to queues is done by the OpenCL runtime in a way it determines as optimal. If CL_COMMAND_BUFFER_PLATFORM_AUTOMATIC_REMAP_KHR is reported, CL_COMMAND_BUFFER_PLATFORM_REMAP_QUEUES_KHR must also be reported.

51.7. Modifications to section 4.2 of the OpenCL API Specification

Add CL_DEVICE_COMMAND_BUFFER_NUM_SYNC_DEVICES_KHR and CL_DEVICE_COMMAND_BUFFER_SYNC_DEVICES_KHR rows to **Table 5**, *Device Queries*, of section 4.2. Also, add additional text to the CL_DEVICE_COMMAND_BUFFER_CAPABILITIES_KHR row:

cl_device_info	Return Type	Description
CL_DEVICE_COMMAND_BUFFER_CAPABILITIES_KHR	cl_device_command_buffer_capabilities_khr	Describes device command-buffer capabilities, encoded as bits in a bitfield. Supported capabilities are: CL_COMMAND_BUFFER_CAPABILITY_MULTIPLE_QUEUE_KHR Device supports the ability to record commands to more than one command-queue associated with *device* in a single command-buffer.
CL_DEVICE_COMMAND_BUFFER_NUM_SYNC_DEVICES_KHR	cl_uint	Return the number of root devices listed in CL_DEVICE_COMMAND_BUFFER_SYNC_DEVICES_KHR that *device* can use device-side synchronization with.

cl_device_info	Return Type	Description
CL_DEVICE_COMMAND_BUFFER_SYNC_DEVICES_KHR	cl_device_id[]	Return the list of root devices *device* can use device-side synchronization with. A device should list itself only if it has native support for synchronizing commands. Sub-devices are not listed to avoid non-deterministic results as sub-devices are created, instead if a root device is listed, then any of its partitioned sub-devices can also be natively synchronized with.

51.8. Modifications to section 5.11 of the OpenCL API Specification

Add additional wording to the description column of **Table 36**, *Event Object Queries*:

CL_EVENT_COMMAND_QUEUE - For events returned by a command-buffer enqueue operation to multiple command-queues, NULL is returned.

CL_EVENT_COMMAND_EXECUTION_STATUS - For events returned by a command-buffer enqueue operation to multiple command-queues the semantics of execution status is as follows:

- CL_QUEUED - Command-buffer has been enqueued across the command-queues.
- CL_SUBMITTED - Commands from the command-buffer have been submitted by the host to any device associated with one of the command-queues.
- CL_RUNNING - Any command from the command-buffer has started execution on a device.
- CL_COMPLETE - All commands have completed on all devices.

51.9. Modifications to section 5.14 of the OpenCL API Specification

51.9.1. Query Updates

Add additional wording to description column of **Table 38**, *Event Profiling Queries*:

- CL_PROFILING_COMMAND_QUEUED - For events returned by a command-buffer enqueue operation to multiple command-queues, the host time when the command-buffer has been enqueued across the command-queues is used.
- CL_PROFILING_COMMAND_SUBMIT - For events returned by a command-buffer enqueue operation to multiple command-queues, the host time is used when command-buffer commands have been submitted to any command-queue.
- CL_PROFILING_COMMAND_START - For events returned by a command-buffer enqueue operation to multiple command-queues, the host time is used when any device starts executing a command-buffer command.
- CL_PROFILING_COMMAND_END - For events returned by a command-buffer enqueue operation to multiple command-queues, the host time is used when the last command-buffer command finishes execution on any device.

- CL_PROFILING_COMMAND_COMPLETE - For events returned by a command-buffer enqueue operation to multiple command-queues, the host time is used when the command-buffer has completed execution across all command-queues.

> If no reliable device timer sources are available to inform the host side, or parallel runtime scheduling makes it impossible to identify a first/last command, then an implementation may fallback to reporting CL_PROFILING_COMMAND_SUBMIT and CL_PROFILING_COMMAND_COMPLETE for CL_PROFILING_COMMAND_START and CL_PROFILING_COMMAND_END respectively.

51.9.2. Error Updates

Extend the wording defining the CL_PROFILING_INFO_NOT_AVAILABLE error return code from **clGetEventProfilingInfo** to append the following sentence:

- If *event* was created from a call to **clEnqueueCommandBufferKHR**, CL_PROFILING_INFO_NOT_AVAILABLE is returned if all the queues passed do not have CL_QUEUE_PROFILING_ENABLE set.

51.10. Modifications to Section 5.X - Command Buffers of the OpenCL API Specification

51.10.1. Additional Section 5.X Introduction Text

A command-buffer can contain commands recorded to the queues of different devices if a vendor provides support for inter-device cl_sync_point_khr synchronization. This feature is reported either through CL_DEVICE_COMMAND_BUFFER_SYNC_DEVICES_KHR, which informs the user what devices can synchronize with each other natively on the device-side, or through CL_COMMAND_BUFFER_PLATFORM_UNIVERSAL_SYNC_KHR, which allows synchronization between all devices in a platform, falling back to host-side synchronization when device-side synchronization isn't available. These two mechanisms are referred to as **device-side sync** and **universal sync** respectively.

If these mechanisms don't report that more than one device can be used in a command-buffer, it will still be possible to perform multiple queue recording in a command-buffer if the CL_COMMAND_BUFFER_CAPABILITY_MULTIPLE_QUEUE_KHR capability is reported for a device. However, with this capability all the queues commands are recorded to must target the same device.

Commands recorded to different command-queues in the same command-buffer may be executed concurrently to each other unless synchronized explicitly with sync-points. Ordering of other commands submitted to the same command-queues as used to enqueue a command-buffer is the responsibility of the programmer. A command-buffer enqueue spanning multiple queues can return an event to use for synchronization, which will complete once all commands in the command-buffer have completed. If ordering restrictions are required, this event (or command-queue barriers) may be used by the user to synchronize the command-buffer enqueue with regular commands, or another command-buffer enqueue.

51.10.2. Add new section "Section 5.X.Y - Remapping Command Buffers"

Platforms reporting the CL_COMMAND_BUFFER_PLATFORM_REMAP_QUEUES_KHR capability support generating a deep copy of a command-buffer with its commands remapped to a list of command-queues that are potentially incompatible with the queues used to create the command-buffer. That is, the remapped command-buffer can execute on queues that differ in terms of properties and/or associated device from the original command-buffer queues.

This functionality is invoked through a new synchronous entry-point **clRemapCommandBufferKHR** which takes a list of queues to which the commands should now target. It then returns a command-buffer containing the same commands as the original, with the same command dependencies, but targeting different queues. A list of command handles may also be passed to the entry-point, which allows handles to the equivalent commands in the remapped command-buffer to be returned by an output parameter.

Device properties restrict remapping possibilities, as existing commands can have a configuration which is not supported by another device, and so remapping may fail with an error relating to this incompatibility. Examples of command configurations which can introduce incompatibilities when trying to map to a new device are:

- Program language features used in a kernel not supported by the new device.
- ND-Range configuration, e.g exceeds new the device max work-group size.
- Misalignment of sub-buffers based on minimum alignment of new device.

In additional to this functionality, platforms reporting CL_COMMAND_BUFFER_PLATFORM_AUTOMATIC_REMAP_KHR allow the user to create a remapped command-buffer where the mapping of queues to commands is determined by the OpenCL runtime in a way it determines as optimal. This is particularly useful in hot plugging environments where devices may appear and disappear during runtime.

The function

```
cl_command_buffer_khr clRemapCommandBufferKHR(
    cl_command_buffer_khr command_buffer,
    cl_bool automatic,
    cl_uint num_queues,
    const cl_command_queue* queues,
    cl_uint num_handles,
    const cl_mutable_command_khr* handles,
    cl_mutable_command_khr* handles_ret,
    cl_int* errcode_ret);
```

Creates a deep copy of the input command-buffer with the copied commands remapped to target the passed command-queues. The returned command-buffer has the same state as the input command-buffer, unless the input command-buffer is in the Pending state, in which case the returned command-buffer has state Executable.

command_buffer Specifies the command-buffer to create a remapped deep copy of.

automatic Indicates if the remapping is done explicitly by the user, or automatically by the OpenCL runtime. If *automatic* is `CL_FALSE`, then each element of *queues* will replace the queue used on *command_buffer* creation at the same index. If `CL_TRUE` and `CL_COMMAND_BUFFER_PLATFORM_AUTOMATIC_REMAP_KHR` is supported, then the OpenCL runtime will decide in a way it determines optimal which of the elements in *queues* each command in the returned command-buffer will be associated with.

num_queues The number of command-queues listed in *queues*, must not be 0.

queues A pointer to an ordered list of command-queues for the returned command-buffer to target, must be a non-`NULL` value.

num_handles The number of command handles passed in both *handles* and *handles_ret* lists, may be 0.

handles An ordered list of handles belonging to *command_buffer* to create remapped copies of, may be `NULL`.

handles_ret Returns an ordered list of handles where each handle is equivalent to the handle at the same index in *handles*, but belonging to the returned command-buffer.

errcode_ret Returns an appropriate error code. If *errcode_ret* is `NULL`, no error code is returned.

clRemapCommandBufferKHR returns a valid command-buffer with *errcode_ret* set to `CL_SUCCESS` if the command-buffer is created successfully. Otherwise, it returns a `NULL` value without setting *handles_ret*, and with one of the following error values returned in *errcode_ret*:

- `CL_INVALID_COMMAND_BUFFER_KHR` if *command_buffer* is not a valid command-buffer.
- `CL_INVALID_VALUE` if *num_queues* is 0, or if *queues* is `NULL`.
- `CL_INVALID_VALUE` if *automatic* is `CL_FALSE` and *num_queues* is not equal to the number of queues used on creation of *command_buffer*.
- `CL_INVALID_VALUE` if *handles* or *handles_ret* is `NULL` and *num_handles* is > 0, or either *handles* or *handles_ret* is not `NULL` and *num_handles* is 0.
- `CL_INVALID_VALUE` if any handle in *handles* is not a valid command handle belonging to *command_buffer*.
- `CL_INVALID_COMMAND_QUEUE` if any command-queue in *queues* is not a valid command-queue.
- `CL_INVALID_CONTEXT` if *command_buffer* and all the command-queues in *queues* do not have the same OpenCL context.
- `CL_INVALID_OPERATION` if the platform does not support the `CL_COMMAND_BUFFER_PLATFORM_REMAP_QUEUES_KHR` flag.
- `CL_INVALID_OPERATION` if the platform does not support the `CL_COMMAND_BUFFER_PLATFORM_AUTOMATIC_REMAP_KHR` flag and *automatic* is `CL_TRUE`.
- `CL_INCOMPATIBLE_COMMAND_QUEUE_KHR` if such an error would be returned by passing *queues* to **clCreateCommandBufferKHR**.
- Any error relating to device support that can be returned by a command recording entry-point may also be returned. As a command in *command_buffer* can have a configuration that is not supported by a device that is associated with the queue in *queues* the command is being

remapped to.

- CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device.
- CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host.

51.10.3. Modifications to clCreateCommandBufferKHR

51.10.4. New Property Flag

Modify the CL_COMMAND_BUFFER_FLAGS_KHR property in the **clCreateCommandBufferKHR** properties table to introduce a new flag to the bitfield. The following text is now included in the description of property values.

Recording Properties	Property Value	Description
CL_COMMAND_BUFFER_FLAGS_KHR	cl_command_buffer_flags_khr	CL_COMMAND_BUFFER_DEVICE_SIDE_SYNC_KHR - All commands in the command-buffer must use native synchronization, as reported by CL_DEVICE_COMMAND_BUFFER_SYNC_DEVICES_KHR. This can be used as a safeguard for performant applications that don't want to accidentally fallback to host synchronization when passing multiple queues.

51.10.5. Add to clCreateCommandBufferKHR description

Table 73. Summary of command-buffer creation configurations

All devices associated with queues can device-side sync	Platform supports universal sync	Condition	Result
Yes	Yes or No	Any device does not support the multi-queue capability, and has more than one queue targeting it	Error - CL_INCOMPATIBLE_COMMAND_QUEUE_KHR
		User sets CL_COMMAND_BUFFER_DEVICE_SIDE_SYNC_KHR flag	OK
		Otherwise	OK

All devices associated with `queues` can device-side sync	Platform supports universal sync	Condition	Result
No	Yes	Any device does not support the multi-queue capability, and has more than one queue targeting it	Error - `CL_INCOMPATIBLE_COMMAND_QUEUE_KHR`
		User sets `CL_COMMAND_BUFFER_DEVICE_SIDE_SYNC_KHR` flag	Error - `CL_INCOMPATIBLE_COMMAND_QUEUE_KHR`
		Otherwise	OK - May be performance implications when synchronizing commands between devices without device-side sync support.
No	No	Always	Error - `CL_INCOMPATIBLE_COMMAND_QUEUE_KHR`

51.10.5.1. Parameter Updates

Parameter descriptions changed to:

num_queues The number of command-queues listed in *queues*.

queues Is a pointer to a list of command-queues that the command-buffer may be executed on. *queues* must be a non-`NULL` value and length of the list equal to *num_queues*.

51.10.5.2. Error Updates

The returned error:

- `CL_INVALID_VALUE` if *num_queues* is not one.

Is changed to:

- `CL_INVALID_VALUE` if *num_queues* is zero.

Additional errors:

- `CL_INCOMPATIBLE_COMMAND_QUEUE_KHR` if *queues* includes more than one command-queue associated with a device that does not support capability `CL_COMMAND_BUFFER_CAPABILITY_MULTIPLE_QUEUE_KHR`.
- `CL_INCOMPATIBLE_COMMAND_QUEUE_KHR` if the `CL_COMMAND_BUFFER_DEVICE_SIDE_SYNC_KHR` flag is set, and any device associated with a command-queue in *queues* cannot natively synchronize with the

other devices associated with *queues* as reported by CL_DEVICE_COMMAND_BUFFER_SYNC_DEVICES_KHR.

- CL_INCOMPATIBLE_COMMAND_QUEUE_KHR if the platform doesn't support the CL_COMMAND_BUFFER_PLATFORM_UNIVERSAL_SYNC_KHR capability, and any device associated with a command-queue in *queues* cannot natively synchronize with the other devices associated with *queues* as reported by CL_DEVICE_COMMAND_BUFFER_SYNC_DEVICES_KHR.

51.10.6. Command recording entry points

The descriptions of command recording entry-points are modified as described in this section. These changes apply to all of **clCommandCopyBufferKHR**, **clCommandCopyBufferRectKHR**, **clCommandCopyBufferToImageKHR**, **clCommandCopyImageKHR**, **clCommandCopyImageToBufferKHR**, **clCommandFillBufferKHR**, **clCommandFillImageKHR**, **clCommandNDRangeKernelKHR**, **clCommandSVMMemcpyKHR** and **clCommandSVMMemFillKHR**.

51.10.6.1. Parameter Update

Parameter description of *command_queue* is changed to:

command_queue Specifies the command-queue the command will be recorded to. If *command_queue* is NULL then only one command-queue must have been set on *command_buffer* creation, otherwise *command_queue* must be a non-NULL value.

51.10.6.2. Error Update

The error condition:

- CL_INVALID_COMMAND_QUEUE if *command_queue* is not NULL.

Is changed to :

- CL_INVALID_COMMAND_QUEUE if *command_queue* is NULL and *command_buffer* was created with more than one queue, or if *command_queue* is non-NULL and not a command-queue listed on *command_buffer* creation.

51.11. Sample Code

```
#define CL_CHECK(ERROR)                                               \
  if (ERROR) {                                                        \
    std::cerr << "OpenCL error: " << ERROR << "\n";                   \
    return ERROR;                                                     \
  }

int main() {
  cl_platform_id platform;
  CL_CHECK(clGetPlatformIDs(1, &platform, nullptr));
  cl_platform_command_buffer_capabilities_khr platform_caps;
  CL_CHECK(clGetPlatformInfo(platform,
                             CL_PLATFORM_COMMAND_BUFFER_CAPABILITIES_KHR,
```

```
                            sizeof(platform_caps), &platform_caps, NULL));
if (!(platform_caps & CL_COMMAND_BUFFER_PLATFORM_AUTOMATIC_REMAP_KHR)) {
  std::cerr << "Command-buffer remapping not supported but used in example, "
               "skipping\n";
  return 0;
}

cl_uint num_devices = 0;
CL_CHECK(clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, 0, NULL, &num_devices));
std::vector<cl_device_id> devices(num_devices);
CL_CHECK(
    clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, 1, devices.data(), nullptr));

// Checks omitted for brevity that either a) the platform supports
// CL_COMMAND_BUFFER_PLATFORM_UNIVERSAL_SYNC_KHR or b) each device is listed
// in the others CL_DEVICE_COMMAND_BUFFER_SYNC_DEVICES_KHR

cl_int error;
cl_context context =
    clCreateContext(NULL, num_devices, devices.data(), NULL, NULL, &error);
CL_CHECK(error);

std::vector<cl_command_queue> queues(num_devices);
for (cl_uint i = 0; i < num_devices; i++) {
  queues[i] = clCreateCommandQueue(context, devices[i], 0, &error);
  CL_CHECK(error);
}

const char *code = R"OpenCLC(
kernel void vector_addition(global int* tile1, global int* tile2,
                            global int* res) {
  size_t index = get_global_id(0);
  res[index] = tile1[index] + tile2[index];
}
)OpenCLC";
const size_t length = std::strlen(code);

cl_program program =
    clCreateProgramWithSource(context, 1, &code, &length, &error);
CL_CHECK(error);

CL_CHECK(
    clBuildProgram(program, num_devices, devices.data(), NULL, NULL, NULL));

cl_kernel kernel = clCreateKernel(program, "vector_addition", &error);
CL_CHECK(error);

constexpr size_t frame_count = 60;
constexpr size_t frame_elements = 1024;
constexpr size_t frame_size = frame_elements * sizeof(cl_int);
```

```
constexpr size_t tile_count = 16;
constexpr size_t tile_elements = frame_elements / tile_count;
constexpr size_t tile_size = tile_elements * sizeof(cl_int);

cl_mem buffer_tile1 =
    clCreateBuffer(context, CL_MEM_READ_ONLY, tile_size, NULL, &error);
CL_CHECK(error);

cl_mem buffer_tile2 =
    clCreateBuffer(context, CL_MEM_READ_ONLY, tile_size, NULL, &error);
CL_CHECK(error);

cl_mem buffer_res =
    clCreateBuffer(context, CL_MEM_WRITE_ONLY, tile_size, NULL, &error);
CL_CHECK(error);

CL_CHECK(clSetKernelArg(kernel, 0, sizeof(buffer_tile1), &buffer_tile1));
CL_CHECK(clSetKernelArg(kernel, 1, sizeof(buffer_tile2), &buffer_tile2));
CL_CHECK(clSetKernelArg(kernel, 2, sizeof(buffer_res), &buffer_res));

cl_command_buffer_khr original_cmdbuf =
    clCreateCommandBufferKHR(num_devices, queues.data(), nullptr, &error);
CL_CHECK(error);

cl_mem buffer_src1 =
    clCreateBuffer(context, CL_MEM_READ_ONLY, frame_size, NULL, &error);
CL_CHECK(error);

cl_mem buffer_src2 =
    clCreateBuffer(context, CL_MEM_READ_ONLY, frame_size, NULL, &error);
CL_CHECK(error);

cl_mem buffer_dst =
    clCreateBuffer(context, CL_MEM_READ_WRITE, frame_size, NULL, &error);
CL_CHECK(error);

cl_sync_point_khr tile_sync_point = 0;
for (size_t tile_index = 0; tile_index < tile_count; tile_index++) {
  cl_sync_point_khr copy_sync_points[2];
  CL_CHECK(clCommandCopyBufferKHR(
      original_cmdbuf, queues[tile_index % num_devices], buffer_src1,
      buffer_tile1, tile_index * tile_size, 0, tile_size,
      tile_sync_point ? 1 : 0, tile_sync_point ? &tile_sync_point : NULL,
      &copy_sync_points[0], NULL));

  CL_CHECK(clCommandCopyBufferKHR(
             original_cmdbuf, queues[tile_index % num_devices], buffer_src2,
             buffer_tile2, tile_index * tile_size, 0, tile_size,
             tile_sync_point ? 1 : 0,
             tile_sync_point ? &tile_sync_point : nullptr,
             &copy_sync_points[1], NULL));
```

```
    cl_sync_point_khr nd_sync_point;
    CL_CHECK(clCommandNDRangeKernelKHR(
        original_cmdbuf, queues[tile_index % num_devices], NULL, kernel, 1,
        NULL, &tile_elements, NULL, 2, copy_sync_points, &nd_sync_point, NULL));

    CL_CHECK(clCommandCopyBufferKHR(
        original_cmdbuf, queues[tile_index % num_devices], buffer_res,
        buffer_dst, 0, tile_index * tile_size, tile_size, 1, &nd_sync_point,
        &tile_sync_point, NULL));
  }

  CL_CHECK(clFinalizeCommandBufferKHR(original_cmdbuf));

  std::random_device random_device;
  std::mt19937 random_engine{random_device()};
  std::uniform_int_distribution<cl_int> random_distribution{
      0, std::numeric_limits<cl_int>::max() / 2};
  auto random_generator = [&]() { return random_distribution(random_engine); };

  auto enqueue_frame = [&](cl_command_buffer_khr command_buffer) {
    for (size_t frame_index = 0; frame_index < frame_count; frame_index++) {
      std::array<cl_event, 3> enqueue_events;
      std::vector<cl_int> src1(frame_elements);
      std::generate(src1.begin(), src1.end(), random_generator);
      CL_CHECK(clEnqueueWriteBuffer(queues[0], buffer_src1, CL_FALSE, 0,
                                    frame_size, src1.data(), 0, nullptr,
                                    &enqueue_events[0]));
      std::vector<cl_int> src2(frame_elements);
      std::generate(src2.begin(), src2.end(), random_generator);
      CL_CHECK(clEnqueueWriteBuffer(queues[0], buffer_src2, CL_FALSE, 0,
                                    frame_size, src2.data(), 0, nullptr,
                                    &enqueue_events[1]));

      CL_CHECK(clEnqueueCommandBufferKHR(0, NULL, command_buffer, 2,
                                         enqueue_events.data(),
                                         &enqueue_events[2]));

      CL_CHECK(clWaitForEvents(1, enqueue_events[2]));

      for (auto e : enqueue_events) {
        CL_CHECK(clReleaseEvent(e));
      }
    }
    return 0;
  };

  error = enqueue_frame(original_cmdbuf);
  CL_CHECK(error);

  // Remap from N queues to 1 queue and run again
```

```
  cl_command_buffer_khr remapped_cmdbuf = clRemapCommandBufferKHR(
      original_cmdbuf, CL_TRUE, 1, queues.data(), 0, NULL, NULL, &error);
  CL_CHECK(error);

  error = enqueue_frame(remapped_cmdbuf);
  CL_CHECK(error);

  for (unsigned i = 0; i < num_devices; ++i) {
    CL_CHECK(clReleaseCommandQueue(queues[i]));
  }
  CL_CHECK(clReleaseMemObject(buffer_src1));
  CL_CHECK(clReleaseMemObject(buffer_src2));
  CL_CHECK(clReleaseMemObject(buffer_dst));

  CL_CHECK(clReleaseMemObject(buffer_tile1));
  CL_CHECK(clReleaseMemObject(buffer_tile2));
  CL_CHECK(clReleaseMemObject(buffer_res));

  CL_CHECK(clReleaseCommandBufferKHR(original_cmdbuf));
  CL_CHECK(clReleaseCommandBufferKHR(remapped_cmdbuf));

  CL_CHECK(clReleaseKernel(kernel));
  CL_CHECK(clReleaseProgram(program));
  CL_CHECK(clReleaseContext(context));

  return 0;
}
```

51.12. Issues

1. In cl_event profiling info for a command-buffer running across the queues for several devices, how do we know what the first & last commands executed are if there is concurrent execution across devices.

   RESOLVED: Allowed an implementation to fallback to `CL_PROFILING_COMMAND_SUBMIT` and `CL_PROFILING_COMMAND_COMPLETE` when reporting `CL_PROFILING_COMMAND_START` & `CL_PROFILING_COMMAND_END`.

2. Is an atomic constraint required? This would forbid regular clEnqueue* commands, from interleaving execution on a queue which a command-buffer is being executed on.

   RESOLVED: This behavior can block parallelism, and constraint is expressible by the user through existing synchronization mechanisms if they require it.

3. It is currently an error if a set of command-queues passed to **clEnqueueCommandBufferKHR** aren't compatible with those set on recording. Should we relax this as an optional capability that allows an implementation to do a more expensive command-buffer enqueue for this case?

   RESOLVED: Added as an optional feature.

Chapter 52. Extensions to the OpenCL SPIR-V Environment

An OpenCL SPIR-V environment may be modified by OpenCL extensions. Please refer to the OpenCL SPIR-V Environment Specification for descriptions how OpenCL extensions modify an OpenCL SPIR-V environment. In addition to the extensions described in this document, the OpenCL SPIR-V Environment Specification also describes how the following OpenCL extensions modify an OpenCL SPIR-V environment:

- `cl_khr_spirv_no_integer_wrap_decoration`
- `cl_khr_spirv_extended_debug_info`
- `cl_khr_spirv_linkonce_odr`

Index

C

clCreateEventFromEGLSyncKHR, 167
clCreateEventFromGLsyncKHR, 103
clCreateFromEGLImageKHR, 171
clCreateFromGLBuffer, 90
clCreateFromGLRenderbuffer, 95
clCreateFromGLTexture, 91
clEnqueueAcquireEGLObjectsKHR, 173
clEnqueueAcquireGLObjects, 98
clEnqueueReleaseEGLObjectsKHR, 174
clEnqueueReleaseGLObjects, 99
clGetExtensionFunctionAddressForPlatform, 4
clGetGLContextInfoKHR, 85
clGetGLObjectInfo, 96
clGetGLTextureInfo, 97

Appendix A: Extensions Promoted to Core Features

A.1. For OpenCL 1.1:

- The functionality previously described by **cl_khr_byte_addressable_store** is now part of the core feature set.
- The functionality previously described by **cl_khr_global_int32_base_atomics**, **cl_khr_global_int32_extended_atomics**, **cl_khr_local_int32_base_atomics**, and **cl_khr_local_int32_extended_atomics** is now part of the core feature set.

A.2. For OpenCL 1.2:

- The functionality previously described by **cl_khr_fp64** is now an optional core feature.

A.3. For OpenCL 2.0:

- The functionality described by **cl_khr_3d_image_writes** is part of the core feature set.
- The functionality described by **cl_khr_create_command_queue** is part of the core feature set.
- The functionality described by **cl_khr_depth_images** is now part of the core feature set.
- The functionality described by **cl_khr_image2d_from_buffer** is now part of the core feature set.

A.4. For OpenCL 2.1:

- The functionality described by **cl_khr_il_program** is now part of the core feature set.
- The API functionality described by **cl_khr_subgroups** is now part of the core API feature set, but the built-in functions described by **cl_khr_subgroups** must still be accessed as an extension to the OpenCL 2.0 C Language specification.

A.5. For OpenCL 3.0:

- The API functionality described by **cl_khr_extended_versioning** is now part of the core API feature set, with minor modifications.
- The built-in functions described by **cl_khr_subgroups** are now supported in OpenCL C 3.0 when the `__opencl_c_subgroups` feature is supported.

Appendix B: Deprecated Extensions

B.1. For OpenCL 1.1:

- The **cl_khr_select_fprounding_mode** extension has been deprecated. Its use is no longer recommended.

Appendix C: Quick Reference

Extension Name	Brief Description	Status
cl_khr_3d_image_writes	Write to 3D images	Core Feature in OpenCL 2.0
cl_khr_async_work_group_copy_fence	Asynchronous Copy Fences	Extension
cl_khr_byte_addressable_store	Read and write from 8-bit and 16-bit pointers	Core Feature in OpenCL 1.1
cl_khr_command_buffer	Record and Replay Commands	Provisional Extension
cl_khr_command_buffer_multi_device	Allow a command-buffer to contain commands targeting different devices	Provisional Extension
cl_khr_command_buffer_mutable_dispatch	Modify kernel execution commands between enqueues of a command-buffer	Provisional Extension
cl_khr_create_command_queue	API to Create Command-Queues with Properties	Core Feature in OpenCL 2.0
cl_khr_d3d10_sharing	Share Direct3D 10 Buffers and Textures with OpenCL	Extension
cl_khr_d3d11_sharing	Share Direct3D 11 Buffers and Textures with OpenCL	Extension
cl_khr_depth_images	Single Channel Depth Images	Core Feature in OpenCL 2.0
cl_khr_device_enqueue_local_arg_types	Pass Non-Void Local Pointers to Child Kernels	Extension
cl_khr_device_uuid	Unique Device and Driver Identifier Queries	Extension
cl_khr_dx9_media_sharing	Share DirectX 9 Media Surfaces with OpenCL	Extension
cl_khr_egl_event	Share EGL Sync Objects with OpenCL	Extension
cl_khr_egl_image	Share EGL Images with OpenCL	Extension
cl_khr_extended_async_copies	2D and 3D Async Copies	Extension
cl_khr_extended_bit_ops	Bit Insert, Extract, and Reverse Operations	Extension
cl_khr_extended_versioning	Extend versioning of platform, devices, extensions, etc.	Core Feature in OpenCL 3.0 (with minor changes)

Extension Name	Brief Description	Status
cl_khr_external_memory	Common Functionality for External Memory Sharing	Provisional Extension
cl_khr_external_memory_dma_buf	dma_buf External Memory Handles	Provisional Extension
cl_khr_external_memory_dx	Direct3D 11 and 12 External Memory Handles	Provisional Extension
cl_khr_external_memory_opaque_fd	Opaque File Descriptor External Memory Handles	Provisional Extension
cl_khr_external_memory_win32	NT Handle External Memory Handles	Provisional Extension
cl_khr_expect_assume	Kernel Optimization Hints	Extension
cl_khr_external_semaphore	Common Functionality for External Semaphore Sharing	Provisional Extension
cl_khr_external_semaphore_dx_fence	Direct3D 12 External Semaphore Handles	Provisional Extension
cl_khr_external_semaphore_opaque_fd	Opaque File Descriptor External Semaphore Handles	Provisional Extension
cl_khr_external_semaphore_sync_fd	Sync FD External Semaphore Handles	Provisional Extension
cl_khr_external_semaphore_win32	NT Handle External Semaphore Handles	Provisional Extension
cl_khr_fp16	Operations on 16-bit Floating-Point Values	Extension
cl_khr_fp64	Operations on 64-bit Floating-Point Values	Optional Core Feature in OpenCL 1.2
cl_khr_gl_depth_images	Share OpenGL Depth Images with OpenCL	Extension
cl_khr_gl_event	Share OpenGL Fence Sync Objects with OpenCL	Extension
cl_khr_gl_msaa_sharing	Share OpenGL MSAA Textures with OpenCL	Extension
cl_khr_gl_sharing	Sharing OpenGL Buffers and Textures with OpenCL	Extension
cl_khr_global_int32_base_atomics	Basic Atomic Operations on 32-bit Integers in Global Memory	Core Feature in OpenCL 1.1
cl_khr_global_int32_extended_atomics	Extended Atomic Operations on 32-bit Integers in Global Memory	Core Feature in OpenCL 1.1
cl_khr_icd	Installable Client Drivers	Extension

Extension Name	Brief Description	Status
cl_khr_il_program	Support for Intermediate Language (IL) Programs (SPIR-V)	Core Feature in OpenCL 2.1
cl_khr_image2d_from_buffer	Create 2D Images from Buffers	Core Feature in OpenCL 2.0
cl_khr_initialize_memory	Initialize Local and Private Memory on Allocation	Extension
cl_khr_int64_base_atomics	Basic Atomic Operations on 64-bit Integers in Global and Local Memory	Extension
cl_khr_int64_extended_atomics	Extended Atomic Operations on 64-bit Integers in Global and Local Memory	Extension
cl_khr_local_int32_base_atomics	Basic Atomic Operations on 32-bit Integers in Local Memory	Core Feature in OpenCL 1.1
cl_khr_local_int32_extended_atomics	Extended Atomic Operations on 32-bit Integers in Local Memory	Core Feature in OpenCL 1.1
cl_khr_integer_dot_product	Integer dot product operations	Extension
cl_khr_mipmap_image	Create and Use Images with Mipmaps	Extension
cl_khr_mipmap_image_writes	Write to Images with Mipmaps	Extension
cl_khr_pci_bus_info	Query PCI Bus Information for an OpenCL Device	Extension
cl_khr_priority_hints	Create Command-Queues with Different Priorities	Extension
cl_khr_select_fprounding_mode	Set the Current Kernel Rounding Mode	DEPRECATED
cl_khr_semaphore	Semaphore Synchronization Primitives	Provisional Extension
cl_khr_spir	Standard Portable Intermediate Representation Programs	Extension, Superseded by IL Programs / SPIR-V
cl_khr_srgb_image_writes	Write to sRGB Images	Extension
cl_khr_subgroups	Sub-Groupings of Work Items	Core Feature in OpenCL 2.1 (with minor changes)
cl_khr_subgroup_ballot	Exchange Ballots Among Sub-Groupings of Work Items	Extension

Extension Name	Brief Description	Status
cl_khr_subgroup_clustered_reduce	Clustered Reductions for Sub-Groupings of Work Items	Extension
cl_khr_subgroup_extended_types	Additional Type Support for Sub-group Functions	Extension
cl_khr_subgroup_named_barrier	Barriers for Subsets of a Work-group	Extension
cl_khr_subgroup_non_uniform_arithmetic	Sub-group Arithmetic Functions in Non-Uniform Control Flow	Extension
cl_khr_subgroup_non_uniform_vote	Hold Votes Among Sub-Groupings of Work Items	Extension
cl_khr_subgroup_rotate	Rotation Among Sub-Groupings of Work Items	Extension
cl_khr_subgroup_shuffle	General-Purpose Shuffles Among Sub-Groupings of Work Items	Extension
cl_khr_subgroup_shuffle_relative	Relative Shuffles Among Sub-Groupings of Work Items	Extension
cl_khr_suggested_local_work_size	Query a Suggested Local Work Size	Extension
cl_khr_terminate_context	Terminate an OpenCL Context	Extension
cl_khr_throttle_hints	Create Command-Queues with Different Throttle Policies	Extension
cl_khr_work_group_uniform_arithmetic	Work-group Uniform Arithmetic	Extension