



# The OpenCL<sup>™</sup> Extension Specification

Khronos<sup>®</sup> OpenCL Working Group

Version V2.2-11, Fri, 19 Jul 2019 14:10:32 +0000

# Table of Contents

1. Extensions Overview	2
2. Installable Client Drivers	6
3. Byte Addressable Stores	12
4. Writing to 3D Image Objects	13
5. Half Precision Floating-Point	15
6. Double Precision Floating-Point	44
7. 32-bit Atomics	64
8. 64-bit Atomics	68
9. Selecting the Rounding Mode ( <b>DEPRECATED</b> )	70
10. Creating an OpenCL Context from an OpenGL Context or Share Group	72
11. Creating OpenCL Memory Objects from OpenGL Objects	79
12. Creating OpenCL Event Objects from OpenGL Sync Objects	91
13. Creating OpenCL Memory Objects from Direct3D 10 Buffers and Textures	95
14. Creating OpenCL Memory Objects from Direct3D 11 Buffers and Textures	111
15. Creating OpenCL Memory Objects from DirectX 9 Media Surfaces	126
16. Depth Images	138
17. Sharing OpenGL and OpenGL ES Depth and Depth-Stencil Images	144
18. Creating OpenCL Memory Objects from OpenGL MSAA Textures	146
19. Creating OpenCL Event Objects from EGL Sync Objects	153
20. Creating OpenCL Memory Objects from EGL Images	157
21. Creating a 2D Image From A Buffer	164
22. Local and Private Memory Initialization	166
23. Terminating OpenCL contexts	167
24. Standard Portable Intermediate Representation Binaries	169
25. Intermediate Language Programs	171
26. Creating Command Queues with Properties	175
27. Device Enqueue Local Argument Types	178
28. Subgroups	179
29. Mipmaps	187
30. sRGB Image Writes	189
31. Priority Hints	190
32. Throttle Hints	191
33. Named Barriers for Subgroups	192
34. Extensions to the OpenCL SPIR-V Environment	193
Index	194
Appendix A: Extensions Promoted to Core Features	195
Appendix B: Deprecated Extensions	196
Appendix C: Quick Reference	197

## Copyright 2008-2019 The Khronos Group.

This specification is protected by copyright laws and contains material proprietary to the Khronos Group, Inc. Except as described by these terms, it or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of Khronos Group.

Khronos Group grants a conditional copyright license to use and reproduce the unmodified specification for any purpose, without fee or royalty, EXCEPT no licenses to any patent, trademark or other intellectual property rights are granted under these terms. Parties desiring to implement the specification and make use of Khronos trademarks in relation to that implementation, and receive reciprocal patent license protection under the Khronos IP Policy must become Adopters and confirm the implementation as conformant under the process defined by Khronos for this specification; see <https://www.khronos.org/adopters>.

Khronos Group makes no, and expressly disclaims any, representations or warranties, express or implied, regarding this specification, including, without limitation: merchantability, fitness for a particular purpose, non-infringement of any intellectual property, correctness, accuracy, completeness, timeliness, and reliability. Under no circumstances will the Khronos Group, or any of its Promoters, Contributors or Members, or their respective partners, officers, directors, employees, agents or representatives be liable for any damages, whether direct, indirect, special or consequential damages for lost revenues, lost profits, or otherwise, arising from or in connection with these materials.

Vulkan and Khronos are registered trademarks, and OpenXR, SPIR, SPIR-V, SYCL, WebGL, WebCL, OpenVX, OpenVG, EGL, COLLADA, glTF, NNEF, OpenKODE, OpenKCAM, StreamInput, OpenWF, OpenSL ES, OpenMAX, OpenMAX AL, OpenMAX IL, OpenMAX DL, OpenML and DevU are trademarks of the Khronos Group Inc. ASTC is a trademark of ARM Holdings PLC, OpenCL is a trademark of Apple Inc. and OpenGL and OpenML are registered trademarks and the OpenGL ES and OpenGL SC logos are trademarks of Silicon Graphics International used under license by Khronos. All other product names, trademarks, and/or company names are used solely for identification and belong to their respective owners.

# 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
<b>enable</b>	Behave as specified by the extension <i>extension_name</i> .  Report an error on the <b>#pragma OPENCL EXTENSION</b> if the <i>extension_name</i> is not supported, or if <b>all</b> is specified.
<b>disable</b>	Behave (including issuing errors and warnings) as if the extension <i>extension_name</i> is not part of the language definition.  If <b>all</b> is specified, then behavior must revert back to that of the non-extended core version of the language being compiled to.  Warn on the <b>#pragma OPENCL EXTENSION</b> if the extension <i>extension_name</i> 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 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 CL_API_ENTRY 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_API_ENTRY 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_ids` 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. 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 order of the functions in `_cl_icd_dispatch` is determined by the ICD Loader's source. The ICD Loader's source's `_cl_icd_dispatch` table is to be appended to only.

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

## 2.3. 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.



## 2.4. 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 will open the Vendor ICD library:

```
c:\vendor_a\vndra_ocl.dll
```

## 2.5. 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`.

## 2.6. 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`.

## 2.7. 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.

## 2.8. New Procedures and Functions

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

## 2.9. 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
```

## 2.10. 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*:

<b>cl_platform_info enum</b>	<b>Return Type</b>	<b>Description</b>
<b>CL_PLATFORM_ICD_SUFFIX_KHR</b>	char[]	The function name suffix used to identify extension functions to be directed to this platform by the ICD Loader.

## 2.11. 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 **icd\_dispatch.h**, which is available as a part of the source code.

## 2.12. 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.

2. How are OpenCL extension to be handled?

RESOLVED: OpenCL extension functions may be added to the ICD as soon as they are implemented by any vendor. The suffix mechanism provides access for vendor extensions which are not yet added to the ICD.

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

RESOLVED: The ICD will by default choose the first enumerated platform as the **NULL** platform. The user can override this default by setting an environment variable `OPENCL_ICD_DEFAULT_PLATFORM` to the desired platform index. The API calls that deal with platforms will return `CL_INVALID_PLATFORM` if the index is not between zero and (number of platforms - 1), both inclusive.

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

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

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 `CL_INVALID_*` error values 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.

# 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.

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

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

Function	Description
<pre>void <b>write_imagef</b> ( image3d_t <i>image</i>, int4 <i>coord</i>, float4 <i>color</i>)  void <b>write_imagei</b> ( image3d_t <i>image</i>, int4 <i>coord</i>, int4 <i>color</i>)  void <b>write_imageui</b> ( image3d_t <i>image</i>, int4 <i>coord</i>, uint4 <i>color</i>)</pre>	<p>Write <i>color</i> value to the location specified by coordinate <math>(x, y, z)</math> in the 3D image specified by <i>image</i>. The appropriate data format conversion to the specified image format is done before writing the color value. <i>coord.x</i>, <i>coord.y</i>, and <i>coord.z</i> 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.</p> <p><b>write_imagef</b> can only be used with image objects created with <i>image_channel_data_type</i> 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.</p> <p><b>write_imagei</b> can only be used with image objects created with <i>image_channel_data_type</i> set to one of the following values: CL_SIGNED_INT8, CL_SIGNED_INT16, or CL_SIGNED_INT32.</p> <p><b>write_imageui</b> can only be used with image objects created with <i>image_channel_data_type</i> set to one of the following values: CL_UNSIGNED_INT8, CL_UNSIGNED_INT16, or CL_UNSIGNED_INT32.</p> <p>The behavior of <b>write_imagef</b>, <b>write_imagei</b>, and <b>write_imageui</b> for image objects created with <i>image_channel_data_type</i> values not specified in the description above, or with <math>(x, y, z)</math> coordinate values that are not in the range (0 ... image width - 1, 0 ... image height - 1, 0 ... image depth - 1) respectively, is undefined.</p>



# 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. 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
<b>half2</b>	A 2-component half-precision floating-point vector.
<b>half3</b>	A 3-component half-precision floating-point vector.
<b>half4</b>	A 4-component half-precision floating-point vector.
<b>half8</b>	A 8-component half-precision floating-point vector.
<b>half16</b>	A 16-component half-precision floating-point vector.

The built-in vector data types for `halffn` 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 `halffn` 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
<b>half2</b>	<code>cl_half2</code>
<b>half3</b>	<code>cl_half3</code>
<b>half4</b>	<code>cl_half4</code>
<b>half8</b>	<code>cl_half8</code>
<b>half16</b>	<code>cl_half16</code>

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

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

### 5.1.1. Conversions

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

The explicit casts described in *section 6.2.2* are extended to take a `half` scalar data type and a `halffn` 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_type()` 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.

### 5.1.2. 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
<code>gentype acos</code> ( <code>gentype x</code> )	Arc cosine function.
<code>gentype acosh</code> ( <code>gentype x</code> )	Inverse hyperbolic cosine.
<code>gentype acospi</code> ( <code>gentype x</code> )	Compute <code>acos</code> ( $x$ ) / $\pi$ .
<code>gentype asin</code> ( <code>gentype x</code> )	Arc sine function.
<code>gentype asinh</code> ( <code>gentype x</code> )	Inverse hyperbolic sine.
<code>gentype asinpi</code> ( <code>gentype x</code> )	Compute <code>asin</code> ( $x$ ) / $\pi$ .
<code>gentype atan</code> ( <code>gentype y_over_x</code> )	Arc tangent function.
<code>gentype atan2</code> ( <code>gentype y</code> , <code>gentype x</code> )	Arc tangent of $y / x$ .
<code>gentype atanh</code> ( <code>gentype x</code> )	Hyperbolic arc tangent.
<code>gentype atanpi</code> ( <code>gentype x</code> )	Compute <code>atan</code> ( $x$ ) / $\pi$ .
<code>gentype atan2pi</code> ( <code>gentype y</code> , <code>gentype x</code> )	Compute <code>atan2</code> ( $y, x$ ) / $\pi$ .
<code>gentype cbrt</code> ( <code>gentype x</code> )	Compute cube-root.
<code>gentype ceil</code> ( <code>gentype x</code> )	Round to integral value using the round to positive infinity rounding mode.
<code>gentype copysign</code> ( <code>gentype x</code> , <code>gentype y</code> )	Returns $x$ with its sign changed to match the sign of $y$ .
<code>gentype cos</code> ( <code>gentype x</code> )	Compute cosine.
<code>gentype cosh</code> ( <code>gentype x</code> )	Compute hyperbolic cosine.
<code>gentype cospi</code> ( <code>gentype x</code> )	Compute <code>cos</code> ( $\pi x$ ).
<code>gentype erfc</code> ( <code>gentype x</code> )	Complementary error function.
<code>gentype erf</code> ( <code>gentype x</code> )	Error function encountered in integrating the normal distribution.
<code>gentype exp</code> ( <code>gentype x</code> )	Compute the base- $e$ exponential of $x$ .
<code>gentype exp2</code> ( <code>gentype x</code> )	Exponential base 2 function.
<code>gentype exp10</code> ( <code>gentype x</code> )	Exponential base 10 function.

Function	Description
gentype <b>expm1</b> (gentype <i>x</i> )	Compute $e^x - 1.0$ .
gentype <b>fabs</b> (gentype <i>x</i> )	Compute absolute value of a floating-point number.
gentype <b>fdim</b> (gentype <i>x</i> , gentype <i>y</i> )	$x - y$ if $x > y$ , $+0$ if $x$ is less than or equal to $y$ .
gentype <b>floor</b> (gentype <i>x</i> )	Round to integral value using the round to negative infinity rounding mode.
gentype <b>fma</b> (gentype <i>a</i> , gentype <i>b</i> , gentype <i>c</i> )	Returns the correctly rounded floating-point representation of the sum of <i>c</i> with the infinitely precise product of <i>a</i> and <i>b</i> . Rounding of intermediate products shall not occur. Edge case behavior is per the IEEE 754-2008 standard.
gentype <b>fmax</b> (gentype <i>x</i> , gentype <i>y</i> ) gentype <b>fmax</b> (gentype <i>x</i> , half <i>y</i> )	Returns <i>y</i> if $x < y$ , otherwise it returns <i>x</i> . If one argument is a NaN, <b>fmax()</b> returns the other argument. If both arguments are NaNs, <b>fmax()</b> returns a NaN.
gentype <b>fmin</b> (gentype <i>x</i> , gentype <i>y</i> ) gentype <b>fmin</b> (gentype <i>x</i> , half <i>y</i> )	Returns <i>y</i> if $y < x$ , otherwise it returns <i>x</i> . If one argument is a NaN, <b>fmin()</b> returns the other argument. If both arguments are NaNs, <b>fmin()</b> returns a NaN.
gentype <b>fmod</b> (gentype <i>x</i> , gentype <i>y</i> )	Modulus. Returns $x - y * \text{trunc}(x/y)$ .
gentype <b>fract</b> (gentype <i>x</i> , __global gentype * <i>iptr</i> ) gentype <b>fract</b> (gentype <i>x</i> , __local gentype * <i>iptr</i> ) gentype <b>fract</b> (gentype <i>x</i> , __private gentype * <i>iptr</i> )	Returns <b>fmin</b> ( $x - \text{floor}(x)$ , $0x1.\text{ffcp}-1f$ ). <b>floor</b> ( <i>x</i> ) is returned in <i>iptr</i> .
If OpenCL 2.0 or newer:  gentype <b>fract</b> (gentype <i>x</i> , gentype * <i>iptr</i> )	
halfn <b>frexp</b> (halfn <i>x</i> , __global intrn * <i>exp</i> ) halfn <b>frexp</b> (halfn <i>x</i> , __local intrn * <i>exp</i> ) halfn <b>frexp</b> (halfn <i>x</i> , __private intrn * <i>exp</i> ) half <b>frexp</b> (half <i>x</i> , __global int * <i>exp</i> ) half <b>frexp</b> (half <i>x</i> , __local int * <i>exp</i> ) half <b>frexp</b> (half <i>x</i> , __private int * <i>exp</i> )	Extract mantissa and exponent from <i>x</i> . For each component the mantissa returned is a float with magnitude in the interval $[1/2, 1)$ or 0. Each component of <i>x</i> equals mantissa returned * $2^{\text{exp}}$ .
If OpenCL 2.0 or newer:  halfn <b>frexp</b> (halfn <i>x</i> , intrn * <i>exp</i> ) half <b>frexp</b> (half <i>x</i> , int * <i>exp</i> )	
gentype <b>hypot</b> (gentype <i>x</i> , gentype <i>y</i> )	Compute the value of the square root of $x^2 + y^2$ without undue overflow or underflow.
intrn <b>ilogb</b> (halfn <i>x</i> ) int <b>ilogb</b> (half <i>x</i> )	Return the exponent as an integer value.
halfn <b>ldexp</b> (halfn <i>x</i> , intrn <i>k</i> ) halfn <b>ldexp</b> (halfn <i>x</i> , int <i>k</i> ) half <b>ldexp</b> (half <i>x</i> , int <i>k</i> )	Multiply <i>x</i> by 2 to the power <i>k</i> .

Function	Description
gentype <b>lgamma</b> (gentype <i>x</i> ) halfn <b>lgamma_r</b> (halfn <i>x</i> , __global intrn * <i>signp</i> ) halfn <b>lgamma_r</b> (halfn <i>x</i> , __local intrn * <i>signp</i> ) halfn <b>lgamma_r</b> (halfn <i>x</i> , __private intrn * <i>signp</i> ) half <b>lgamma_r</b> (half <i>x</i> , __global int * <i>signp</i> ) half <b>lgamma_r</b> (half <i>x</i> , __local int * <i>signp</i> ) half <b>lgamma_r</b> (half <i>x</i> , __private int * <i>signp</i> )  If OpenCL 2.0 or newer:  halfn <b>lgamma_r</b> (halfn <i>x</i> , intrn * <i>signp</i> ) half <b>lgamma_r</b> (half <i>x</i> , int * <i>signp</i> )	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 <i>signp</i> argument of <b>lgamma_r</b> .
gentype <b>log</b> (gentype <i>x</i> )	Compute natural logarithm.
gentype <b>log2</b> (gentype <i>x</i> )	Compute a base 2 logarithm.
gentype <b>log10</b> (gentype <i>x</i> )	Compute a base 10 logarithm.
gentype <b>log1p</b> (gentype <i>x</i> )	Compute $\log_e(1.0 + x)$ .
gentype <b>logb</b> (gentype <i>x</i> )	Compute the exponent of <i>x</i> , which is the integral part of $\log_r  x $ .
gentype <b>mad</b> (gentype <i>a</i> , gentype <i>b</i> , gentype <i>c</i> )	<p><b>mad</b> computes <math>a * b + c</math>. The function may compute <math>a * b + c</math> with reduced accuracy in the embedded profile. See the SPIR-V OpenCL environment specification for details. On some hardware the mad instruction may provide better performance than expanded computation of <math>a * b + c</math>.</p> <p>Note: For some usages, e.g. <b>mad</b>(<i>a</i>, <i>b</i>, -<i>a</i>*<i>b</i>), the half precision definition of <b>mad</b>() is loose enough that almost any result is allowed from <b>mad</b>() for some values of <i>a</i> and <i>b</i>.</p>
gentype <b>maxmag</b> (gentype <i>x</i> , gentype <i>y</i> )	Returns <i>x</i> if $ x  >  y $ , <i>y</i> if $ y  >  x $ , otherwise <b>fmax</b> ( <i>x</i> , <i>y</i> ).
gentype <b>minmag</b> (gentype <i>x</i> , gentype <i>y</i> )	Returns <i>x</i> if $ x  <  y $ , <i>y</i> if $ y  <  x $ , otherwise <b>fmin</b> ( <i>x</i> , <i>y</i> ).
gentype <b>modf</b> (gentype <i>x</i> , __global gentype * <i>iptr</i> ) gentype <b>modf</b> (gentype <i>x</i> , __local gentype * <i>iptr</i> ) gentype <b>modf</b> (gentype <i>x</i> , __private gentype * <i>iptr</i> )  If OpenCL 2.0 or newer:  gentype <b>modf</b> (gentype <i>x</i> , gentype * <i>iptr</i> )	Decompose a floating-point number. The <b>modf</b> function breaks the argument <i>x</i> 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 <i>iptr</i> .
halfn <b>nan</b> (ushortn <i>nancode</i> ) half <b>nan</b> (ushort <i>nancode</i> )	Returns a quiet NaN. The <i>nancode</i> may be placed in the significand of the resulting NaN.

Function	Description
gentype <b>nextafter</b> (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, <b>nextafter</b> () returns the largest representable floating-point number less than x.
gentype <b>pow</b> (gentype x, gentype y)	Compute x to the power y.
halfn <b>pown</b> (halfn x, intn y) half <b>pown</b> (half x, int y)	Compute x to the power y, where y is an integer.
gentype <b>powr</b> (gentype x, gentype y)	Compute x to the power y, where x is $\geq 0$ .
gentype <b>remainder</b> (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.
halfn <b>remquo</b> (halfn x, halfn y, __global intrn *quo) halfn <b>remquo</b> (halfn x, halfn y, __local intrn *quo) halfn <b>remquo</b> (halfn x, halfn y, __private intrn *quo) half <b>remquo</b> (half x, half y, __global int *quo) half <b>remquo</b> (half x, half y, __local int *quo) half <b>remquo</b> (half x, half y, __private int *quo)  If OpenCL 2.0 or newer:  halfn <b>remquo</b> (halfn x, halfn y, intrn *quo) half <b>remquo</b> (half x, half y, int *quo)	The <b>remquo</b> 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 <b>remainder</b> function. <b>remquo</b> 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 <b>rint</b> (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.
halfn <b>rootn</b> (halfn x, intn y) half <b>rootn</b> (half x, int y)	Compute x to the power $1/y$ .
gentype <b>round</b> (gentype x)	Return the integral value nearest to x rounding halfway cases away from zero, regardless of the current rounding direction.
gentype <b>rsqrt</b> (gentype x)	Compute inverse square root.
gentype <b>sin</b> (gentype x)	Compute sine.

Function	Description
gentype <b>sincos</b> (gentype x, __global gentype *cosval) gentype <b>sincos</b> (gentype x, __local gentype *cosval) gentype <b>sincos</b> (gentype x, __private gentype *cosval)	Compute sine and cosine of x. The computed sine is the return value and computed cosine is returned in <i>cosval</i> .
If OpenCL 2.0 or newer:	
gentype <b>sincos</b> (gentype x, gentype *cosval)	
gentype <b>sinh</b> (gentype x)	
gentype <b>sinpi</b> (gentype x)	Compute <b>sin</b> ( $\pi x$ ).
gentype <b>sqrt</b> (gentype x)	Compute square root.
gentype <b>tan</b> (gentype x)	Compute tangent.
gentype <b>tanh</b> (gentype x)	Compute hyperbolic tangent.
gentype <b>tanpi</b> (gentype x)	Compute <b>tan</b> ( $\pi x$ ).
gentype <b>tgamma</b> (gentype x)	Compute the gamma function.
gentype <b>trunc</b> (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
<b>HALF_DIG</b>	<b>CL_HALF_DIG</b>

Macro in OpenCL Language	Macro for application
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_EPSILON	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 $\pi$
M_PI_2_H	Value of $\pi / 2$
M_PI_4_H	Value of $\pi / 4$
M_1_PI_H	Value of $1 / \pi$
M_2_PI_H	Value of $2 / \pi$
M_2_SQRTPI_H	Value of $2 / \sqrt{\pi}$
M_SQRT2_H	Value of $\sqrt{2}$
M_SQRT1_2_H	Value of $1 / \sqrt{2}$

### 5.1.3. 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 <b>clamp</b> ( gentype <i>x</i> , gentype <i>minval</i> , gentype <i>maxval</i> )  gentype <b>clamp</b> ( gentype <i>x</i> , half <i>minval</i> , half <i>maxval</i> )	Returns <b>fmin(fmax(<i>x</i>, <i>minval</i>), <i>maxval</i>)</b> .  Results are undefined if <i>minval</i> > <i>maxval</i> .
gentype <b>degrees</b> (gentype <i>radians</i> )	Converts <i>radians</i> to degrees, i.e. $(180 / \pi) * \textit{radians}$ .
gentype <b>max</b> (gentype <i>x</i> , gentype <i>y</i> ) gentype <b>max</b> (gentype <i>x</i> , half <i>y</i> )	Returns <i>y</i> if <i>x</i> < <i>y</i> , otherwise it returns <i>x</i> . If <i>x</i> and <i>y</i> are infinite or NaN, the return values are undefined.
gentype <b>min</b> (gentype <i>x</i> , gentype <i>y</i> ) gentype <b>min</b> (gentype <i>x</i> , half <i>y</i> )	Returns <i>y</i> if <i>y</i> < <i>x</i> , otherwise it returns <i>x</i> . If <i>x</i> and <i>y</i> are infinite or NaN, the return values are undefined.
gentype <b>mix</b> (gentype <i>x</i> , gentype <i>y</i> , gentype <i>a</i> ) gentype <b>mix</b> (gentype <i>x</i> , gentype <i>y</i> , half <i>a</i> )	Returns the linear blend of <i>x</i> and <i>y</i> implemented as:  $x + (y - x) * a$  <i>a</i> must be a value in the range 0.0 ... 1.0. If <i>a</i> is not in the range 0.0 ... 1.0, the return values are undefined.  Note: The half precision <b>mix</b> function can be implemented using contractions such as <b>mad</b> or <b>fma</b> .
gentype <b>radians</b> (gentype <i>degrees</i> )	Converts <i>degrees</i> to radians, i.e. $(\pi / 180) * \textit{degrees}$ .
gentype <b>step</b> (gentype <i>edge</i> , gentype <i>x</i> ) gentype <b>step</b> (half <i>edge</i> , gentype <i>x</i> )	Returns 0.0 if <i>x</i> < <i>edge</i> , otherwise it returns 1.0.
gentype <b>smoothstep</b> ( gentype <i>edge0</i> , gentype <i>edge1</i> , gentype <i>x</i> )  gentype <b>smoothstep</b> ( half <i>edge0</i> , half <i>edge1</i> , gentype <i>x</i> )	Returns 0.0 if <i>x</i> <= <i>edge0</i> and 1.0 if <i>x</i> >= <i>edge1</i> and performs smooth Hermite interpolation between 0 and 1 when <i>edge0</i> < <i>x</i> < <i>edge1</i> . This is useful in cases where you would want a threshold function with a smooth transition.  This is equivalent to:  <pre>gentype t; t = clamp ((x - edge0) / (edge1 - edge0), 0, 1); return t * t * (3 - 2 * t);</pre>  Results are undefined if <i>edge0</i> >= <i>edge1</i> .  Note: The half precision <b>smoothstep</b> function can be implemented using contractions such as <b>mad</b> or <b>fma</b> .



Function	Description
gentype <b>sign</b> (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.

### 5.1.4. 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 <b>cross</b> (half4 $p0$ , half4 $p1$ ) half3 <b>cross</b> (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 <b>dot</b> (gentype $p0$ , gentype $p1$ )	Compute the dot product of $p0$ and $p1$ .
half <b>distance</b> (gentype $p0$ , gentype $p1$ )	Returns the distance between $p0$ and $p1$ . This is calculated as <code>length(p0 - p1)</code> .
half <b>length</b> (gentype $p$ )	Return the length of vector $x$ , i.e., $\sqrt{p.x^2 + p.y^2 + \dots}$
gentype <b>normalize</b> (gentype $p$ )	Returns a vector in the same direction as $p$ but with a length of 1.

### 5.1.5. 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 <b>isequal</b> (half <i>x</i> , half <i>y</i> ) shortn <b>isequal</b> (halfn <i>x</i> , halfn <i>y</i> )	Returns the component-wise compare of $x == y$ .
int <b>isnotequal</b> (half <i>x</i> , half <i>y</i> ) shortn <b>isnotequal</b> (halfn <i>x</i> , halfn <i>y</i> )	Returns the component-wise compare of $x != y$ .
int <b>isgreater</b> (half <i>x</i> , half <i>y</i> ) shortn <b>isgreater</b> (halfn <i>x</i> , halfn <i>y</i> )	Returns the component-wise compare of $x > y$ .
int <b>isgreaterequal</b> (half <i>x</i> , half <i>y</i> ) shortn <b>isgreaterequal</b> (halfn <i>x</i> , halfn <i>y</i> )	Returns the component-wise compare of $x >= y$ .
int <b>isless</b> (half <i>x</i> , half <i>y</i> ) shortn <b>isless</b> (halfn <i>x</i> , halfn <i>y</i> )	Returns the component-wise compare of $x < y$ .
int <b>islessequal</b> (half <i>x</i> , half <i>y</i> ) shortn <b>islessequal</b> (halfn <i>x</i> , halfn <i>y</i> )	Returns the component-wise compare of $x <= y$ .
int <b>islessgreater</b> (half <i>x</i> , half <i>y</i> ) shortn <b>islessgreater</b> (halfn <i>x</i> , halfn <i>y</i> )	Returns the component-wise compare of $(x < y)    (x > y)$ .
int <b>isfinite</b> (half) shortn <b>isfinite</b> (halfn)	Test for finite value.
int <b>isinf</b> (half) shortn <b>isinf</b> (halfn)	Test for infinity value (positive or negative).
int <b>isnan</b> (half) shortn <b>isnan</b> (halfn)	Test for a NaN.
int <b>isnormal</b> (half) shortn <b>isnormal</b> (halfn)	Test for a normal value.
int <b>isordered</b> (half <i>x</i> , half <i>y</i> ) shortn <b>isordered</b> (halfn <i>x</i> , halfn <i>y</i> )	Test if arguments are ordered. <b>isordered()</b> takes arguments <i>x</i> and <i>y</i> , and returns the result <b>isequal(x, x) &amp;&amp; isequal(y, y)</b> .
int <b>isunordered</b> (half <i>x</i> , half <i>y</i> ) shortn <b>isunordered</b> (halfn <i>x</i> , halfn <i>y</i> )	Test if arguments are unordered. <b>isunordered()</b> takes arguments <i>x</i> and <i>y</i> , returning non-zero if <i>x</i> or <i>y</i> is a NaN, and zero otherwise.
int <b>signbit</b> (half) shortn <b>signbit</b> (halfn)	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 halfn: -1 (i.e all bits set) if the sign bit in the half is set else returns 0.
halfn <b>bitselect</b> (halfn <i>a</i> , halfn <i>b</i> , halfn <i>c</i> )	Each bit of the result is the corresponding bit of <i>a</i> if the corresponding bit of <i>c</i> is 0. Otherwise it is the corresponding bit of <i>b</i> .
halfn <b>select</b> (halfn <i>a</i> , halfn <i>b</i> , shortn <i>c</i> ) halfn <b>select</b> (halfn <i>a</i> , halfn <i>b</i> , ushortn <i>c</i> )	For each component, $result[i] = \text{if MSB of } c[i] \text{ is set ? } b[i] : a[i]$ .

### 5.1.6. Vector Data Load and Store Functions

The vector data load (**vloadn**) and store (**vstoren**) functions described in *table 6.13* (also listed

below) are extended to include versions that read from or write to half scalar or 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  $(p + (\text{offset} * 3))$  into a 3-component vector and `vstore3` writes  $x, y, z$  components from a 3-component vector to address  $(p + (\text{offset} * 3))$ .

Table 6. Half Precision Vector Data Load and Store Functions

Function	Description
<code>gentypen vloadn(size_t offset, const __global gentype *p)</code>	Return <code>sizeof(gentypen)</code> bytes of data read from address $(p + (\text{offset} * n))$ . The read address computed as $(p + (\text{offset} * n))$ must be 16-bit aligned.
<code>gentypen vloadn(size_t offset, const __local gentype *p)</code>	
<code>gentypen vloadn(size_t offset, const __constant gentype *p)</code>	
<code>gentypen vloadn(size_t offset, const __private gentype *p)</code>	
If OpenCL 2.0 or newer: <code>gentypen vloadn(size_t offset, const gentype *p)</code>	
<code>void vstoren(gentypen data, size_t offset, __global gentype *p)</code>	Write <code>sizeof(gentypen)</code> bytes given by <code>data</code> to address $(p + (\text{offset} * n))$ . The write address computed as $(p + (\text{offset} * n))$ must be 16-bit aligned.
<code>void vstoren(gentypen data, size_t offset, __local gentype *p)</code>	
<code>void vstoren(gentypen data, size_t offset, __private gentype *p)</code>	
If OpenCL 2.0 or newer: <code>void vstoren(gentypen data, size_t offset, gentype *p)</code>	

### 5.1.7. 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
<pre> event_t <b>async_work_group_copy</b> ( __local gentype *dst, const __global gentype *src, size_t num_gentypes, event_t event)  event_t <b>async_work_group_copy</b> ( __global gentype *dst, const __local gentype *src, size_t num_gentypes, event_t event) </pre>	<p>Perform an async copy of <i>num_gentypes</i> gentype elements from <i>src</i> to <i>dst</i>. 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.</p> <p>Returns an event object that can be used by <b>wait_group_events</b> to wait for the async copy to finish. The <i>event</i> argument can also be used to associate the <b>async_work_group_copy</b> with a previous async copy allowing an event to be shared by multiple async copies; otherwise <i>event</i> should be zero.</p> <p>If <i>event</i> argument is not zero, the event object supplied in <i>event</i> argument will be returned.</p> <p>This function does not perform any implicit synchronization of source data such as using a <b>barrier</b> before performing the copy.</p>

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

## 5.1.8. 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.

## 5.1.9. Built-in Image Read Functions

Table 8. Half Precision Built-in Image Read Functions

Function	Description
<p>half4 <b>read_imageh</b> (  read_only image2d_t <i>image</i>,  sampler_t <i>sampler</i>,  int2 <i>coord</i>)</p> <p>half4 <b>read_imageh</b> (  read_only image2d_t <i>image</i>,  sampler_t <i>sampler</i>,  float2 <i>coord</i>)</p>	<p>Use the coordinate (<i>coord.x</i>, <i>coord.y</i>) to do an element lookup in the 2D image object specified by <i>image</i>.</p> <p><b>read_imageh</b> returns half precision floating-point values in the range [0.0 ... 1.0] for image objects created with <i>image_channel_data_type</i> set to one of the pre-defined packed formats, CL_UNORM_INT8, or CL_UNORM_INT16.</p> <p><b>read_imageh</b> returns half precision floating-point values in the range [-1.0 ... 1.0] for image objects created with <i>image_channel_data_type</i> set to CL_SNORM_INT8, or CL_SNORM_INT16.</p> <p><b>read_imageh</b> returns half precision floating-point values for image objects created with <i>image_channel_data_type</i> set to CL_HALF_FLOAT.</p> <p>The <b>read_imageh</b> 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.</p> <p>Values returned by <b>read_imageh</b> for image objects with <i>image_channel_data_type</i> values not specified in the description above are undefined.</p>

Function	Description
<p>half4 <b>read_imageh</b> (  read_only image3d_t <i>image</i>,  sampler_t <i>sampler</i>,  int4 <i>coord</i> )</p> <p>half4 <b>read_imageh</b> (  read_only image3d_t <i>image</i>,  sampler_t <i>sampler</i>,  float4 <i>coord</i> )</p>	<p>Use the coordinate (<i>coord.x</i>, <i>coord.y</i>, <i>coord.z</i>) to do an elementlookup in the 3D image object specified by <i>image</i>. <i>coord.w</i> is ignored.</p> <p><b>read_imageh</b> returns half precision floating-point values in the range [0.0 ... 1.0] for image objects created with <i>image_channel_data_type</i> set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16.</p> <p><b>read_imageh</b> returns half precision floating-point values in the range [-1.0 ... 1.0] for image objects created with <i>image_channel_data_type</i> set to CL_SNORM_INT8, or CL_SNORM_INT16.</p> <p><b>read_imageh</b> returns half precision floating-point values for image objects created with <i>image_channel_data_type</i> set to CL_HALF_FLOAT.</p> <p>The <b>read_imageh</b> 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.</p> <p>Values returned by <b>read_imageh</b> for image objects with <i>image_channel_data_type</i> values not specified in the description are undefined.</p>

Function	Description
<p>half4 <b>read_imageh</b> (  read_only image2d_array_t <i>image</i>,  sampler_t <i>sampler</i>,  int4 <i>coord</i>)</p> <p>half4 <b>read_imageh</b> (  read_only image2d_array_t <i>image</i>,  sampler_t <i>sampler</i>,  float4 <i>coord</i>)</p>	<p>Use <i>coord.xy</i> to do an element lookup in the 2D image identified by <i>coord.z</i> in the 2D image array specified by <i>image</i>.</p> <p><b>read_imageh</b> returns half precision floating-point values in the range [0.0 ... 1.0] for image objects created with <i>image_channel_data_type</i> set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16.</p> <p><b>read_imageh</b> returns half precision floating-point values in the range [-1.0 ... 1.0] for image objects created with <i>image_channel_data_type</i> set to CL_SNORM_INT8, or CL_SNORM_INT16.</p> <p><b>read_imageh</b> returns half precision floating-point values for image objects created with <i>image_channel_data_type</i> set to CL_HALF_FLOAT.</p> <p>The <b>read_imageh</b> 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.</p> <p>Values returned by <b>read_imageh</b> for image objects with <i>image_channel_data_type</i> values not specified in the description above are undefined.</p>



Function	Description
<p>half4 <b>read_imageh</b> (  read_only image1d_t <i>image</i>,  sampler_t <i>sampler</i>,  int <i>coord</i>)</p> <p>half4 <b>read_imageh</b> (  read_only image1d_t <i>image</i>,  sampler_t <i>sampler</i>,  float <i>coord</i>)</p>	<p>Use <i>coord</i> to do an element lookup in the 1D image object specified by <i>image</i>.</p> <p><b>read_imageh</b> returns half precision floating-point values in the range [0.0 ... 1.0] for image objects created with <i>image_channel_data_type</i> set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16.</p> <p><b>read_imageh</b> returns half precision floating-point values in the range [-1.0 ... 1.0] for image objects created with <i>image_channel_data_type</i> set to CL_SNORM_INT8, or CL_SNORM_INT16.</p> <p><b>read_imageh</b> returns half precision floating-point values for image objects created with <i>image_channel_data_type</i> set to CL_HALF_FLOAT.</p> <p>The <b>read_imageh</b> 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.</p> <p>Values returned by <b>read_imageh</b> for image objects with <i>image_channel_data_type</i> values not specified in the description above are undefined.</p>

Function	Description
<p>half4 <b>read_imageh</b> (  read_only image1d_array_t <i>image</i>,  sampler_t <i>sampler</i>,  int2 <i>coord</i>)</p> <p>half4 <b>read_imageh</b> (  read_only image1d_array_t <i>image</i>,  sampler_t <i>sampler</i>,  float2 <i>coord</i>)</p>	<p>Use <i>coord.x</i> to do an element lookup in the 1D image identified by <i>coord.y</i> in the 1D image array specified by <i>image</i>.</p> <p><b>read_imageh</b> returns half precision floating-point values in the range [0.0 ... 1.0] for image objects created with <i>image_channel_data_type</i> set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16.</p> <p><b>read_imageh</b> returns half precision floating-point values in the range [-1.0 ... 1.0] for image objects created with <i>image_channel_data_type</i> set to CL_SNORM_INT8, or CL_SNORM_INT16.</p> <p><b>read_imageh</b> returns half precision floating-point values for image objects created with <i>image_channel_data_type</i> set to CL_HALF_FLOAT.</p> <p>The <b>read_imageh</b> 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.</p> <p>Values returned by <b>read_imageh</b> for image objects with <i>image_channel_data_type</i> values not specified in the description above are undefined.</p>

### 5.1.10. 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
<p>half4 <b>read_imageh</b> (  <i>aQual</i> image2d_t <i>image</i>,  int2 <i>coord</i>)</p>	<p>Use the coordinate (<i>coord.x</i>, <i>coord.y</i>) to do an element lookup in the 2D image object specified by <i>image</i>.</p> <p><b>read_imageh</b> returns half precision floating-point values in the range [0.0 ... 1.0] for image objects created with <i>image_channel_data_type</i> set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16.</p> <p><b>read_imageh</b> returns half precision floating-point values in the range [-1.0 ... 1.0] for image objects created with <i>image_channel_data_type</i> set to CL_SNORM_INT8, or CL_SNORM_INT16.</p> <p><b>read_imageh</b> returns half precision floating-point values for image objects created with <i>image_channel_data_type</i> set to CL_HALF_FLOAT.</p> <p>Values returned by <b>read_imageh</b> for image objects with <i>image_channel_data_type</i> values not specified in the description above are undefined.</p>
<p>half4 <b>read_imageh</b> (  <i>aQual</i> image3d_t <i>image</i>,  int4 <i>coord</i> )</p>	<p>Use the coordinate (<i>coord.x</i>, <i>coord.y</i>, <i>coord.z</i>) to do an element lookup in the 3D image object specified by <i>image</i>. <i>coord.w</i> is ignored.</p> <p><b>read_imageh</b> returns half precision floating-point values in the range [0.0 ... 1.0] for image objects created with <i>image_channel_data_type</i> set to one of the pre-defined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16.</p> <p><b>read_imageh</b> returns half precision floating-point values in the range [-1.0 ... 1.0] for image objects created with <i>image_channel_data_type</i> set to CL_SNORM_INT8, or CL_SNORM_INT16.</p> <p><b>read_imageh</b> returns half precision floating-point values for image objects created with <i>image_channel_data_type</i> set to CL_HALF_FLOAT.</p> <p>Values returned by <b>read_imageh</b> for image objects with <i>image_channel_data_type</i> values not specified in the description are undefined.</p>

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

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

### 5.1.11. 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
<pre>void <b>write_imageh</b> ( aQual image2d_t <i>image</i>, int2 <i>coord</i>, half4 <i>color</i>)</pre>	<p>Write <i>color</i> value to location specified by <i>coord.xy</i> in the 2D image specified by <i>image</i>.</p> <p>Appropriate data format conversion to the specified image format is done before writing the color value. <i>x</i> &amp; <i>y</i> are considered to be unnormalized coordinates and must be in the range 0 ... width - 1, and 0 ... height - 1.</p> <p><b>write_imageh</b> can only be used with image objects created with <i>image_channel_data_type</i> 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.</p> <p>The behavior of <b>write_imageh</b> for image objects created with <i>image_channel_data_type</i> values not specified in the description above or with (<i>x</i>, <i>y</i>) coordinate values that are not in the range (0 ... width - 1, 0 ... height - 1) respectively, is undefined.</p>
<pre>void <b>write_imageh</b> ( aQual image2d_array_t <i>image</i>, int4 <i>coord</i>, half4 <i>color</i>)</pre>	<p>Write <i>color</i> value to location specified by <i>coord.xy</i> in the 2D image identified by <i>coord.z</i> in the 2D image array specified by <i>image</i>.</p> <p>Appropriate data format conversion to the specified image format is done before writing the color value. <i>coord.x</i>, <i>coord.y</i> and <i>coord.z</i> 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.</p> <p><b>write_imageh</b> can only be used with image objects created with <i>image_channel_data_type</i> 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.</p> <p>The behavior of <b>write_imageh</b> for image objects created with <i>image_channel_data_type</i> values not specified in the description above or with (<i>x</i>, <i>y</i>, <i>z</i>) 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.</p>

Function	Description
<pre>void write_imageh ( aQual image1d_t image, int coord, half4 color)  void write_imageh ( aQual image1d_buffer_t image, int coord, half4 color)</pre>	<p>Write <i>color</i> value to location specified by <i>coord</i> in the 1D image or 1D image buffer object specified by <i>image</i>. Appropriate data format conversion to the specified image format is done before writing the color value. <i>coord</i> is considered to be unnormalized coordinates and must be in the range 0 ... image width - 1.</p> <p><b>write_imageh</b> can only be used with image objects created with <i>image_channel_data_type</i> 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.</p> <p>The behavior of <b>write_imageh</b> for image objects created with <i>image_channel_data_type</i> values not specified in the description above or with coordinate values that is not in the range (0 ... image width - 1), is undefined.</p>
<pre>void write_imageh ( aQual image1d_array_t image, int2 coord, half4 color)</pre>	<p>Write <i>color</i> value to location specified by <i>coord.x</i> in the 1D image identified by <i>coord.y</i> in the 1D image array specified by <i>image</i>. Appropriate data format conversion to the specified image format is done before writing the color value. <i>coord.x</i> and <i>coord.y</i> are considered to be unnormalized coordinates and must be in the range 0 ... image width - 1 and 0 ... image number of layers - 1.</p> <p><b>write_imageh</b> can only be used with image objects created with <i>image_channel_data_type</i> 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.</p> <p>The behavior of <b>write_imageh</b> for image objects created with <i>image_channel_data_type</i> values not specified in the description above or with (<i>x</i>, <i>y</i>) coordinate values that are not in the range (0 ... image width - 1, 0 ... image number of layers - 1), respectively, is undefined.</p>

Function	Description
<pre>void <b>write_imageh</b> (     aQual image3d_t image,     int4 coord,     half4 color)</pre>	<p>Write color value to location specified by coord.xyz in the 3D image object specified by image.</p> <p>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.</p> <p><b>write_imageh</b> 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.</p> <p>The behavior of <b>write_imageh</b> 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.</p> <p>Note: This built-in function is only available if the cl_khr_3d_image_writes extension is also supported by the device.</p>

### 5.1.12. 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
<b>CL_DEVICE_HALF_FP_CONFIG</b>	cl_device_fp_config	<p>Describes half precision floating-point capability of the OpenCL device. This is a bit-field that describes one or more of the following values:</p> <p>CL_FP_DENORM — denorms are supported</p> <p>CL_FP_INF_NAN — INF and NaNs are supported</p> <p>CL_FP_ROUND_TO_NEAREST — round to nearest even rounding mode supported</p> <p>CL_FP_ROUND_TO_ZERO — round to zero rounding mode supported</p> <p>CL_FP_ROUND_TO_INF — round to positive and negative infinity rounding modes supported</p> <p>CL_FP_FMA — IEEE754-2008 fused multiply-add is supported</p> <p>CL_FP_SOFT_FLOAT — Basic floating-point operations (such as addition, subtraction, multiplication) are implemented in software.</p> <p>The required minimum half precision floating-point capability as implemented by this extension is:</p> <p>CL_FP_ROUND_TO_ZERO, or CL_FP_ROUND_TO_NEAREST   CL_FP_INF_NAN.</p>

### 5.1.13. 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.

### 5.1.14. 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	$\leq 1$ ulp
$x / y$	Correctly rounded	$\leq 1$ ulp
<b>acos</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>acosh</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>acospi</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>asin</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>asinh</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>asinpi</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>atan</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>atanh</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>atanpi</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>atan2</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>atan2pi</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>cbrt</b>	$\leq 2$ ulp	$\leq 2$ ulp
<b>ceil</b>	Correctly rounded	Correctly rounded
<b>clamp</b>	0 ulp	0 ulp
<b>copysign</b>	0 ulp	0 ulp
<b>cos</b>	$\leq 2$ ulp	$\leq 2$ ulp
<b>cosh</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>cospi</b>	$\leq 2$ ulp	$\leq 2$ ulp
<b>cross</b>	absolute error tolerance of 'max * max * (3 * HLF_EPSILON)', where max is the maximum input operand value	Implementation-defined
<b>degrees</b>	$\leq 2$ ulp	$\leq 2$ ulp

Function	Min Accuracy - Full Profile	Min Accuracy - Embedded Profile
<b>distance</b>	$\leq 0.5 + (1.5 * n) + (0.5 * (n - 1))$ ulp, for gentype with vector width n	Implementation-defined
<b>dot</b>	absolute error tolerance of 'max * max * (2 * (n - 1)) * HLF_EPSILON', For vector width n and maximum input operand value 'max'	Implementation-defined
<b>erfc</b>	$\leq 4$ ulp	$\leq 4$ ulp
<b>erf</b>	$\leq 4$ ulp	$\leq 4$ ulp
<b>exp</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>exp2</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>exp10</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>expm1</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>fabs</b>	0 ulp	0 ulp
<b>fdim</b>	Correctly rounded	Correctly rounded
<b>floor</b>	Correctly rounded	Correctly rounded
<b>fma</b>	Correctly rounded	Correctly rounded
<b>fmax</b>	0 ulp	0 ulp
<b>fmin</b>	0 ulp	0 ulp
<b>fmod</b>	0 ulp	0 ulp
<b>fract</b>	Correctly rounded	Correctly rounded
<b>frexp</b>	0 ulp	0 ulp
<b>hypot</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>ilogb</b>	0 ulp	0 ulp
<b>ldexp</b>	Correctly rounded	Correctly rounded
<b>length</b>	$\leq 0.5 + 0.5 * 0.5 * n) + (0.5 * (n - 1))$ ulp, for gentype with vector width n	Implementation-defined
<b>log</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>log2</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>log10</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>log1p</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>logb</b>	0 ulp	0 ulp
<b>mad</b>	Implementation-defined	Implementation-defined
<b>max</b>	0 ulp	0 ulp
<b>maxmag</b>	0 ulp	0 ulp
<b>min</b>	0 ulp	0 ulp

Function	Min Accuracy - Full Profile	Min Accuracy - Embedded Profile
<b>minmag</b>	0 ulp	0 ulp
<b>mix</b>	Implementation-defined	Implementation-defined
<b>modf</b>	0 ulp	0 ulp
<b>nan</b>	0 ulp	0 ulp
<b>nextafter</b>	0 ulp	0 ulp
<b>normalize</b>	$\leq 1.5 + (0.5 * n) + (0.5 * (n - 1))$ ulp, for gentype with vector width n	Implementation-defined
<b>pow(x, y)</b>	$\leq 4$ ulp	$\leq 5$ ulp
<b>pown(x, y)</b>	$\leq 4$ ulp	$\leq 5$ ulp
<b>powr(x, y)</b>	$\leq 4$ ulp	$\leq 5$ ulp
<b>radians</b>	$\leq 2$ ulp	$\leq 2$ ulp
<b>remainder</b>	0 ulp	0 ulp
<b>remquo</b>	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
<b>rint</b>	Correctly rounded	Correctly rounded
<b>rootn</b>	$\leq 4$ ulp	$\leq 5$ ulp
<b>round</b>	Correctly rounded	Correctly rounded
<b>rsqrt</b>	$\leq 1$ ulp	$\leq 1$ ulp
<b>sign</b>	0 ulp	0 ulp
<b>sin</b>	$\leq 2$ ulp	$\leq 2$ ulp
<b>sincos</b>	$\leq 2$ ulp for sine and cosine values	$\leq 2$ ulp for sine and cosine values
<b>sinh</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>sinpi</b>	$\leq 2$ ulp	$\leq 2$ ulp
<b>smoothstep</b>	Implementation-defined	Implementation-defined
<b>sqrt</b>	Correctly rounded	$\leq 1$ ulp
<b>step</b>	0 ulp	0 ulp
<b>tan</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>tanh</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>tanpi</b>	$\leq 2$ ulp	$\leq 3$ ulp
<b>tgamma</b>	$\leq 4$ ulp	$\leq 4$ ulp
<b>trunc</b>	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.*

# 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. 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
<b>double</b>	A double precision float.
<b>double2</b>	A 2-component double-precision floating-point vector.
<b>double3</b>	A 3-component double-precision floating-point vector.
<b>double4</b>	A 4-component double-precision floating-point vector.
<b>double8</b>	A 8-component double-precision floating-point vector.
<b>double16</b>	A 16-component double-precision floating-point vector.

The built-in scalar and vector data types for `double_n` 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 `double_n` 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
<b>double</b>	<b>cl_double</b>
<b>double2</b>	<b>cl_double2</b>
<b>double3</b>	<b>cl_double3</b>
<b>double4</b>	<b>cl_double4</b>
<b>double8</b>	<b>cl_double8</b>
<b>double16</b>	<b>cl_double16</b>

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.

### 6.1.1. Conversions

The implicit conversion rules specified in *section 6.2.1* now include the `double` scalar and `double_n` 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_typed()` 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.

## 6.1.2. 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
<code>gentype acos</code> ( <code>gentype x</code> )	Arc cosine function.
<code>gentype acosh</code> ( <code>gentype x</code> )	Inverse hyperbolic cosine.
<code>gentype acospi</code> ( <code>gentype x</code> )	Compute <code>acos</code> ( $x$ ) / $\pi$ .
<code>gentype asin</code> ( <code>gentype x</code> )	Arc sine function.
<code>gentype asinh</code> ( <code>gentype x</code> )	Inverse hyperbolic sine.
<code>gentype asinpi</code> ( <code>gentype x</code> )	Compute <code>asin</code> ( $x$ ) / $\pi$ .
<code>gentype atan</code> ( <code>gentype y_over_x</code> )	Arc tangent function.
<code>gentype atan2</code> ( <code>gentype y</code> , <code>gentype x</code> )	Arc tangent of $y / x$ .
<code>gentype atanh</code> ( <code>gentype x</code> )	Hyperbolic arc tangent.
<code>gentype atanpi</code> ( <code>gentype x</code> )	Compute <code>atan</code> ( $x$ ) / $\pi$ .
<code>gentype atan2pi</code> ( <code>gentype y</code> , <code>gentype x</code> )	Compute <code>atan2</code> ( $y, x$ ) / $\pi$ .
<code>gentype cbrt</code> ( <code>gentype x</code> )	Compute cube-root.
<code>gentype ceil</code> ( <code>gentype x</code> )	Round to integral value using the round to positive infinity rounding mode.
<code>gentype copysign</code> ( <code>gentype x</code> , <code>gentype y</code> )	Returns $x$ with its sign changed to match the sign of $y$ .
<code>gentype cos</code> ( <code>gentype x</code> )	Compute cosine.
<code>gentype cosh</code> ( <code>gentype x</code> )	Compute hyperbolic cosine.
<code>gentype cospi</code> ( <code>gentype x</code> )	Compute <code>cos</code> ( $\pi x$ ).
<code>gentype erfc</code> ( <code>gentype x</code> )	Complementary error function.
<code>gentype erf</code> ( <code>gentype x</code> )	Error function encountered in integrating the normal distribution.
<code>gentype exp</code> ( <code>gentype x</code> )	Compute the base- $e$ exponential of $x$ .

Function	Description
gentype <b>exp2</b> (gentype <i>x</i> )	Exponential base 2 function.
gentype <b>exp10</b> (gentype <i>x</i> )	Exponential base 10 function.
gentype <b>expm1</b> (gentype <i>x</i> )	Compute $e^x - 1.0$ .
gentype <b>fabs</b> (gentype <i>x</i> )	Compute absolute value of a floating-point number.
gentype <b>fdim</b> (gentype <i>x</i> , gentype <i>y</i> )	$x - y$ if $x > y$ , $+0$ if $x$ is less than or equal to $y$ .
gentype <b>floor</b> (gentype <i>x</i> )	Round to integral value using the round to negative infinity rounding mode.
gentype <b>fma</b> (gentype <i>a</i> , gentype <i>b</i> , gentype <i>c</i> )	Returns the correctly rounded floating-point representation of the sum of <i>c</i> with the infinitely precise product of <i>a</i> and <i>b</i> . Rounding of intermediate products shall not occur. Edge case behavior is per the IEEE 754-2008 standard.
gentype <b>fmax</b> (gentype <i>x</i> , gentype <i>y</i> ) gentype <b>fmax</b> (gentype <i>x</i> , double <i>y</i> )	Returns <i>y</i> if $x < y$ , otherwise it returns <i>x</i> . If one argument is a NaN, <b>fmax()</b> returns the other argument. If both arguments are NaNs, <b>fmax()</b> returns a NaN.
gentype <b>fmin</b> (gentype <i>x</i> , gentype <i>y</i> ) gentype <b>fmin</b> (gentype <i>x</i> , double <i>y</i> )	Returns <i>y</i> if $y < x$ , otherwise it returns <i>x</i> . If one argument is a NaN, <b>fmin()</b> returns the other argument. If both arguments are NaNs, <b>fmin()</b> returns a NaN.
gentype <b>fmod</b> (gentype <i>x</i> , gentype <i>y</i> )	Modulus. Returns $x - y * \text{trunc}(x/y)$ .
gentype <b>fract</b> (gentype <i>x</i> , __global gentype <i>iptr</i> ) gentype * <b>fract</b> (gentype <i>x</i> , __local gentype <i>iptr</i> ) gentype * <b>fract</b> (gentype <i>x</i> , __private gentype * <i>iptr</i> )	Returns <b>fmin</b> ( $x - \text{floor}(x)$ , $0 \times 1.\text{fffffffffffp-1}$ ). <b>floor</b> ( <i>x</i> ) is returned in <i>iptr</i> .
doublen <b>frexp</b> (doublen <i>x</i> , __global intn <b>exp</b> ) doublen * <b>frexp</b> (doublen <i>x</i> , __local intn <b>exp</b> ) doublen * <b>frexp</b> (doublen <i>x</i> , __private intn <b>exp</b> ) double * <b>frexp</b> (double <i>x</i> , __global int <b>exp</b> ) double * <b>frexp</b> (double <i>x</i> , __local int <b>exp</b> ) double * <b>frexp</b> (double <i>x</i> , __private int * <b>exp</b> )	Extract mantissa and exponent from <i>x</i> . For each component the mantissa returned is a float with magnitude in the interval $[1/2, 1)$ or 0. Each component of <i>x</i> equals mantissa returned * $2^{\text{exp}}$ .
gentype <b>hypot</b> (gentype <i>x</i> , gentype <i>y</i> )	Compute the value of the square root of $x^2 + y^2$ without undue overflow or underflow.
intn <b>ilogb</b> (doublen <i>x</i> ) int <b>ilogb</b> (double <i>x</i> )	Return the exponent as an integer value.
doublen <b>ldexp</b> (doublen <i>x</i> , intn <i>k</i> ) doublen <b>ldexp</b> (doublen <i>x</i> , int <i>k</i> ) double <b>ldexp</b> (double <i>x</i> , int <i>k</i> )	Multiply <i>x</i> by 2 to the power <i>k</i> .



Function	Description
gentype <b>lgamma</b> (gentype <i>x</i> ) doublen <b>lgamma_r</b> (doublen <i>x</i> , __global intrn * <i>signp</i> ) doublen <b>lgamma_r</b> (doublen <i>x</i> , __local intrn * <i>signp</i> ) doublen <b>lgamma_r</b> (doublen <i>x</i> , __private intrn * <i>signp</i> ) double <b>lgamma_r</b> (double <i>x</i> , __global int * <i>signp</i> ) double <b>lgamma_r</b> (double <i>x</i> , __local int * <i>signp</i> ) double <b>lgamma_r</b> (double <i>x</i> , __private int * <i>signp</i> )	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 <i>signp</i> argument of <b>lgamma_r</b> .
gentype <b>log</b> (gentype <i>x</i> )	Compute natural logarithm.
gentype <b>log2</b> (gentype <i>x</i> )	Compute a base 2 logarithm.
gentype <b>log10</b> (gentype <i>x</i> )	Compute a base 10 logarithm.
gentype <b>log1p</b> (gentype <i>x</i> )	Compute $\log_e(1.0 + x)$ .
gentype <b>logb</b> (gentype <i>x</i> )	Compute the exponent of <i>x</i> , which is the integral part of $\log_r  x $ .
gentype <b>mad</b> (gentype <i>a</i> , gentype <i>b</i> , gentype <i>c</i> )	<b>mad</b> computes $a * b + c$ . The function may compute $a * b + c$ with reduced accuracy in the embedded profile. See the SPIR-V OpenCL environment specification for details. On some hardware the mad instruction may provide better performance than expanded computation of $a * b + c$ .
gentype <b>maxmag</b> (gentype <i>x</i> , gentype <i>y</i> )	Returns <i>x</i> if $ x  >  y $ , <i>y</i> if $ y  >  x $ , otherwise <b>fmax</b> ( <i>x</i> , <i>y</i> ).
gentype <b>minmag</b> (gentype <i>x</i> , gentype <i>y</i> )	Returns <i>x</i> if $ x  <  y $ , <i>y</i> if $ y  <  x $ , otherwise <b>fmin</b> ( <i>x</i> , <i>y</i> ).
gentype <b>modf</b> (gentype <i>x</i> , __global gentype <i>iptr</i> ) gentype * <b>modf</b> (gentype <i>x</i> , __local gentype <i>iptr</i> ) gentype * <b>modf</b> (gentype <i>x</i> , __private gentype * <i>iptr</i> )	Decompose a floating-point number. The <b>modf</b> function breaks the argument <i>x</i> 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 <i>iptr</i> .
doublen <b>nan</b> (ulongn <i>nancode</i> ) double <b>nan</b> (ulong <i>nancode</i> )	Returns a quiet NaN. The <i>nancode</i> may be placed in the significand of the resulting NaN.
gentype <b>nextafter</b> (gentype <i>x</i> , gentype <i>y</i> )	Computes the next representable double-precision floating-point value following <i>x</i> in the direction of <i>y</i> . Thus, if <i>y</i> is less than <i>x</i> , <b>nextafter</b> () returns the largest representable floating-point number less than <i>x</i> .
gentype <b>pow</b> (gentype <i>x</i> , gentype <i>y</i> )	Compute <i>x</i> to the power <i>y</i> .
doublen <b>pown</b> (doublen <i>x</i> , intrn <i>y</i> ) double <b>pown</b> (double <i>x</i> , int <i>y</i> )	Compute <i>x</i> to the power <i>y</i> , where <i>y</i> is an integer.
gentype <b>powr</b> (gentype <i>x</i> , gentype <i>y</i> )	Compute <i>x</i> to the power <i>y</i> , where <i>x</i> is $\geq 0$ .

Function	Description
gentype <b>remainder</b> (gentype <i>x</i> , gentype <i>y</i> )	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 $x$ .
doublen <b>remquo</b> (doublen <i>x</i> , doublen <i>y</i> , __global intn * <i>quo</i> ) doublen <b>remquo</b> (doublen <i>x</i> , doublen <i>y</i> , __local intn * <i>quo</i> ) doublen <b>remquo</b> (doublen <i>x</i> , doublen <i>y</i> , __private intn * <i>quo</i> ) double <b>remquo</b> (double <i>x</i> , double <i>y</i> , __global int * <i>quo</i> ) double <b>remquo</b> (double <i>x</i> , double <i>y</i> , __local int * <i>quo</i> ) double <b>remquo</b> (double <i>x</i> , double <i>y</i> , __private int * <i>quo</i> )	The <b>remquo</b> 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 <b>remainder</b> function. <b>remquo</b> 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 <i>quo</i> .
gentype <b>rint</b> (gentype <i>x</i> )	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.
doublen <b>rootn</b> (doublen <i>x</i> , intn <i>y</i> ) doublen <b>rootn</b> (double <i>x</i> , int <i>y</i> )	Compute $x$ to the power $1/y$ .
gentype <b>round</b> (gentype <i>x</i> )	Return the integral value nearest to $x$ rounding halfway cases away from zero, regardless of the current rounding direction.
gentype <b>rsqrt</b> (gentype <i>x</i> )	Compute inverse square root.
gentype <b>sin</b> (gentype <i>x</i> )	Compute sine.
gentype <b>sincos</b> (gentype <i>x</i> , __global gentype <i>cosval</i> ) gentype * <b>sincos</b> (gentype <i>x</i> , __local gentype <i>cosval</i> ) gentype * <b>sincos</b> (gentype <i>x</i> , __private gentype * <i>cosval</i> )	Compute sine and cosine of $x$ . The computed sine is the return value and computed cosine is returned in <i>cosval</i> .
gentype <b>sinh</b> (gentype <i>x</i> )	Compute hyperbolic sine.
gentype <b>sinpi</b> (gentype <i>x</i> )	Compute <b>sin</b> ( $\pi x$ ).
gentype <b>sqrt</b> (gentype <i>x</i> )	Compute square root.
gentype <b>tan</b> (gentype <i>x</i> )	Compute tangent.
gentype <b>tanh</b> (gentype <i>x</i> )	Compute hyperbolic tangent.
gentype <b>tanpi</b> (gentype <i>x</i> )	Compute <b>tan</b> ( $\pi x$ ).
gentype <b>tgamma</b> (gentype <i>x</i> )	Compute the gamma function.
gentype <b>trunc</b> (gentype <i>x</i> )	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
<b>DBL_DIG</b>	<b>CL_DBL_DIG</b>
<b>DBL_MANT_DIG</b>	<b>CL_DBL_MANT_DIG</b>
<b>DBL_MAX_10_EXP</b>	<b>CL_DBL_MAX_10_EXP</b>
<b>DBL_MAX_EXP</b>	<b>CL_DBL_MAX_EXP</b>
<b>DBL_MIN_10_EXP</b>	<b>CL_DBL_MIN_10_EXP</b>
<b>DBL_MIN_EXP</b>	<b>CL_DBL_MIN_EXP</b>
<b>DBL_RADIX</b>	<b>CL_DBL_RADIX</b>
<b>DBL_MAX</b>	<b>CL_DBL_MAX</b>
<b>DBL_MIN</b>	<b>CL_DBL_MIN</b>
<b>DBL_EPSILSON</b>	<b>CL_DBL_EPSILON</b>

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

Constant	Description
<b>M_E</b>	Value of e
<b>M_LOG2E</b>	Value of log <sub>2</sub> e
<b>M_LOG10E</b>	Value of log <sub>10</sub> e

Constant	Description
<b>M_LN2</b>	Value of $\log_e 2$
<b>M_LN10</b>	Value of $\log_e 10$
<b>M_PI</b>	Value of $\pi$
<b>M_PI_2</b>	Value of $\pi / 2$
<b>M_PI_4</b>	Value of $\pi / 4$
<b>M_1_PI</b>	Value of $1 / \pi$
<b>M_2_PI</b>	Value of $2 / \pi$
<b>M_2_SQRTPI</b>	Value of $2 / \sqrt{\pi}$
<b>M_SQRT2</b>	Value of $\sqrt{2}$
<b>M_SQRT1_2</b>	Value of $1 / \sqrt{2}$

### 6.1.3. 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
<code>gentype clamp (</code> <code>gentype x, gentype minval, gentype maxval)</code>	Returns <code>fmin(fmax(x, minval), maxval)</code> .  Results are undefined if <code>minval &gt; maxval</code> .
<code>gentype clamp (</code> <code>gentype x, double minval, double maxval)</code>	
<code>gentype degrees (gentype radians)</code>	Converts <i>radians</i> to degrees, i.e. $(180 / \pi) * \textit{radians}$ .
<code>gentype max (gentype x, gentype y)</code> <code>gentype max (gentype x, double y)</code>	Returns <code>y</code> if <code>x &lt; y</code> , otherwise it returns <code>x</code> . If <code>x</code> and <code>y</code> are infinite or NaN, the return values are undefined.
<code>gentype min (gentype x, gentype y)</code> <code>gentype min (gentype x, double y)</code>	Returns <code>y</code> if <code>y &lt; x</code> , otherwise it returns <code>x</code> . If <code>x</code> and <code>y</code> are infinite or NaN, the return values are undefined.

Function	Description
gentype <b>mix</b> (gentype <i>x</i> , gentype <i>y</i> , gentype <i>a</i> ) gentype <b>mix</b> (gentype <i>x</i> , gentype <i>y</i> , double <i>a</i> )	Returns the linear blend of <i>x</i> and <i>y</i> implemented as:  $x + (y - x) * a$ <p><i>a</i> must be a value in the range 0.0 ... 1.0. If <i>a</i> is not in the range 0.0 ... 1.0, the return values are undefined.</p> <p>Note: The double precision <b>mix</b> function can be implemented using contractions such as <b>mad</b> or <b>fma</b>.</p>
gentype <b>radians</b> (gentype <i>degrees</i> )	Converts <i>degrees</i> to radians, i.e. $(\pi / 180) * \textit{degrees}$ .
gentype <b>step</b> (gentype <i>edge</i> , gentype <i>x</i> ) gentype <b>step</b> (double <i>edge</i> , gentype <i>x</i> )	Returns 0.0 if $x < \textit{edge}$ , otherwise it returns 1.0.
gentype <b>smoothstep</b> ( gentype <i>edge0</i> , gentype <i>edge1</i> , gentype <i>x</i> )  gentype <b>smoothstep</b> ( double <i>edge0</i> , double <i>edge1</i> , gentype <i>x</i> )	Returns 0.0 if $x \leq \textit{edge0}$ and 1.0 if $x \geq \textit{edge1}$ and performs smooth Hermite interpolation between 0 and 1 when $\textit{edge0} < x < \textit{edge1}$ . This is useful in cases where you would want a threshold function with a smooth transition.  This is equivalent to:  <pre>gentype t; t = clamp ((x - edge0) / (edge1 - edge0), 0, 1); return t * t * (3 - 2 * t);</pre> <p>Results are undefined if <math>\textit{edge0} \geq \textit{edge1}</math>.</p> <p>Note: The double precision <b>smoothstep</b> function can be implemented using contractions such as <b>mad</b> or <b>fma</b>.</p>
gentype <b>sign</b> (gentype <i>x</i> )	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 <i>x</i> is a NaN.

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

### 6.1.5. 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 <b>isequal</b> (double <i>x</i> , double <i>y</i> ) longn <b>isequal</b> (doublen <i>x</i> , doublen <i>y</i> )	Returns the component-wise compare of $x == y$ .
int <b>isnotequal</b> (double <i>x</i> , double <i>y</i> ) longn <b>isnotequal</b> (doublen <i>x</i> , doublen <i>y</i> )	Returns the component-wise compare of $x != y$ .
int <b>isgreater</b> (double <i>x</i> , double <i>y</i> ) longn <b>isgreater</b> (doublen <i>x</i> , doublen <i>y</i> )	Returns the component-wise compare of $x > y$ .
int <b>isgreaterequal</b> (double <i>x</i> , double <i>y</i> ) longn <b>isgreaterequal</b> (doublen <i>x</i> , doublen <i>y</i> )	Returns the component-wise compare of $x >= y$ .
int <b>isless</b> (double <i>x</i> , double <i>y</i> ) longn <b>isless</b> (doublen <i>x</i> , doublen <i>y</i> )	Returns the component-wise compare of $x < y$ .
int <b>islessequal</b> (double <i>x</i> , double <i>y</i> ) longn <b>islessequal</b> (doublen <i>x</i> , doublen <i>y</i> )	Returns the component-wise compare of $x <= y$ .

Function	Description
int <b>islessgreater</b> (double <i>x</i> , double <i>y</i> ) longn <b>islessgreater</b> (doublen <i>x</i> , doublen <i>y</i> )	Returns the component-wise compare of ( $x < y$ )    ( $x > y$ ) .
int <b>isfinite</b> (double) longn <b>isfinite</b> (doublen)	Test for finite value.
int <b>isinf</b> (double) longn <b>isinf</b> (doublen)	Test for infinity value (positive or negative) .
int <b>isnan</b> (double) longn <b>isnan</b> (doublen)	Test for a NaN.
int <b>isnormal</b> (double) longn <b>isnormal</b> (doublen)	Test for a normal value.
int <b>isordered</b> (double <i>x</i> , double <i>y</i> ) longn <b>isordered</b> (doublen <i>x</i> , doublen <i>y</i> )	Test if arguments are ordered. <b>isordered()</b> takes arguments <i>x</i> and <i>y</i> , and returns the result <b>isequal</b> ( <i>x</i> , <i>x</i> ) && <b>isequal</b> ( <i>y</i> , <i>y</i> ).
int <b>isunordered</b> (double <i>x</i> , double <i>y</i> ) longn <b>isunordered</b> (doublen <i>x</i> , doublen <i>y</i> )	Test if arguments are unordered. <b>isunordered()</b> takes arguments <i>x</i> and <i>y</i> , returning non-zero if <i>x</i> or <i>y</i> is a NaN, and zero otherwise.
int <b>signbit</b> (double) longn <b>signbit</b> (doublen)	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 doublen: -1 (i.e all bits set) if the sign bit in the double is set else returns 0.
doublen <b>bitselect</b> (doublen <i>a</i> , doublen <i>b</i> , doublen <i>c</i> )	Each bit of the result is the corresponding bit of <i>a</i> if the corresponding bit of <i>c</i> is 0. Otherwise it is the corresponding bit of <i>b</i> .
doublen <b>select</b> (doublen <i>a</i> , doublen <i>b</i> , longn <i>c</i> ) doublen <b>select</b> (doublen <i>a</i> , doublen <i>b</i> , ulongn <i>c</i> )	For each component, $result[i] = \text{if MSB of } c[i] \text{ is set ? } b[i] : a[i]$ .

### 6.1.6. Vector Data Load and Store Functions

The vector data load (**vloadn**) and store (**vstoren**) 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\_halfn** and **vstorea\_halfn** 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 ( $p + (\text{offset} * 3)$ ) into a 3-component vector. **vstore3**, and **vstore\_half3** write (*x,y,z*) components from a 3-component vector to address ( $p + (\text{offset} * 3)$ ). In addition, **vloada\_half3** reads (*x,y,z*) components from address ( $p + (\text{offset} * 4)$ ) into a 3-component vector and **vstorea\_half3** writes (*x,y,z*) components from a 3-component vector to address ( $p + (\text{offset} * 4)$ ). 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
<code>gentypen <b>vloadn</b>(size_t <i>offset</i>, const __global gentype *<i>p</i>)</code>	Return sizeof ( <i>gentypen</i> ) bytes of data read from address ( $p + (offset * n)$ ). The read address computed as ( $p + (offset * n)$ ) must be 8-bit aligned if <i>gentype</i> is char, uchar; 16-bit aligned if <i>gentype</i> is short, ushort; 32-bit aligned if <i>gentype</i> is int, uint, float; 64-bit aligned if <i>gentype</i> is long, ulong or double.
<code>gentypen <b>vloadn</b>(size_t <i>offset</i>, const __local gentype *<i>p</i>)</code>	
<code>gentypen <b>vloadn</b>(size_t <i>offset</i>, const __constant gentype *<i>p</i>)</code>	
<code>gentypen <b>vloadn</b>(size_t <i>offset</i>, const __private gentype *<i>p</i>)</code>	
<code>void <b>vstoren</b>(gentypen <i>data</i>, size_t <i>offset</i>, __global gentype *<i>p</i>)</code>	Write sizeof ( <i>gentypen</i> ) bytes given by <i>data</i> to address ( $p + (offset * n)$ ). The address computed as ( $p + (offset * n)$ ) must be 8-bit aligned if <i>gentype</i> is char, uchar; 16-bit aligned if <i>gentype</i> is short, ushort; 32-bit aligned if <i>gentype</i> is int, uint, float; 64-bit aligned if <i>gentype</i> is long, ulong or double.
<code>void <b>vstoren</b>(gentypen <i>data</i>, size_t <i>offset</i>, __local gentype *<i>p</i>)</code>	
<code>void <b>vstoren</b>(gentypen <i>data</i>, size_t <i>offset</i>, __private gentype *<i>p</i>)</code>	



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

Function	Description
<pre> void <b>vstore_halfn</b>(doublen <i>data</i>, size_t <i>offset</i>, __global_half *<i>p</i>) void <b>vstore_halfn_rte</b>(doublen <i>data</i>, size_t <i>offset</i>, __global_half *<i>p</i>) void <b>vstore_halfn_rtz</b>(doublen <i>data</i>, size_t <i>offset</i>, __global_half *<i>p</i>) void <b>vstore_halfn_rtp</b>(doublen <i>data</i>, size_t <i>offset</i>, __global_half *<i>p</i>) void <b>vstore_halfn_rtn</b>(doublen <i>data</i>, size_t <i>offset</i>, __global_half *<i>p</i>)  void <b>vstore_halfn</b>(doublen <i>data</i>, size_t <i>offset</i>, __local_half *<i>p</i>) void <b>vstore_halfn_rte</b>(doublen <i>data</i>, size_t <i>offset</i>, __local_half *<i>p</i>) void <b>vstore_halfn_rtz</b>(doublen <i>data</i>, size_t <i>offset</i>, __local_half *<i>p</i>) void <b>vstore_halfn_rtp</b>(doublen <i>data</i>, size_t <i>offset</i>, __local_half *<i>p</i>) void <b>vstore_halfn_rtn</b>(doublen <i>data</i>, size_t <i>offset</i>, __local_half *<i>p</i>)  void <b>vstore_halfn</b>(doublen <i>data</i>, size_t <i>offset</i>, __private_half *<i>p</i>) void <b>vstore_halfn_rte</b>(doublen <i>data</i>, size_t <i>offset</i>, __private_half *<i>p</i>) void <b>vstore_halfn_rtz</b>(doublen <i>data</i>, size_t <i>offset</i>, __private_half *<i>p</i>) void <b>vstore_halfn_rtp</b>(doublen <i>data</i>, size_t <i>offset</i>, __private_half *<i>p</i>) void <b>vstore_halfn_rtn</b>(doublen <i>data</i>, size_t <i>offset</i>, __private_half *<i>p</i>) </pre>	<p>The doublen value given by <i>data</i> is converted to a halfn value using the appropriate rounding mode. The halfn value is then written to the address computed as <math>(p + (offset * n))</math>. The address computed as <math>(p + (offset * n))</math> must be 16-bit aligned.</p> <p><b>vstore_halfn</b> uses the current rounding mode. The default current rounding mode is round to nearest even.</p>

Function	Description
<pre>void vstorea_halfn(doublen data, size_t offset, __global half *p) void vstorea_halfn_rte(doublen data, size_t offset, __global half *p) void vstorea_halfn_rtz(doublen data, size_t offset, __global half *p) void vstorea_halfn_rtp(doublen data, size_t offset, __global half *p) void vstorea_halfn_rtn(doublen data, size_t offset, __global half *p)  void vstorea_halfn(doublen data, size_t offset, __local half *p) void vstorea_halfn_rte(doublen data, size_t offset, __local half *p) void vstorea_halfn_rtz(doublen data, size_t offset, __local half *p) void vstorea_halfn_rtp(doublen data, size_t offset, __local half *p) void vstorea_halfn_rtn(doublen data, size_t offset, __local half *p)  void vstorea_halfn(doublen data, size_t offset, __private half *p) void vstorea_halfn_rte(doublen data, size_t offset, __private half *p) void vstorea_halfn_rtz(doublen data, size_t offset, __private half *p) void vstorea_halfn_rtp(doublen data, size_t offset, __private half *p) void vstorea_halfn_rtn(doublen data, size_t offset, __private half *p)</pre>	<p>The <i>doublen</i> value is converted to a <i>halfn</i> value using the appropriate rounding mode.</p> <p>For <math>n = 1, 2, 4, 8</math> or <math>16</math>, the <i>halfn</i> value is written to the address computed as <math>(p + (offset * n))</math>. The address computed as <math>(p + (offset * n))</math> must be aligned to <code>sizeof (halfn)</code> bytes.</p> <p>For <math>n = 3</math>, the <i>half3</i> value is written to the address computed as <math>(p + (offset * 4))</math>. The address computed as <math>(p + (offset * 4))</math> must be aligned to <code>sizeof (half) * 4</code> bytes.</p> <p><b>vstorea_halfn</b> uses the current rounding mode. The default current rounding mode is round to nearest even.</p>

### 6.1.7. 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
<pre> event_t <b>async_work_group_copy</b> ( __local gentype *dst, const __global gentype *src, size_t num_gentypes, event_t event)  event_t <b>async_work_group_copy</b> ( __global gentype *dst, const __local gentype *src, size_t num_gentypes, event_t event) </pre>	<p>Perform an async copy of <i>num_gentypes</i> gentype elements from <i>src</i> to <i>dst</i>. 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.</p> <p>Returns an event object that can be used by <b>wait_group_events</b> to wait for the async copy to finish. The <i>event</i> argument can also be used to associate the <b>async_work_group_copy</b> with a previous async copy allowing an event to be shared by multiple async copies; otherwise <i>event</i> should be zero.</p> <p>If <i>event</i> argument is not zero, the event object supplied in <i>event</i> argument will be returned.</p> <p>This function does not perform any implicit synchronization of source data such as using a <b>barrier</b> before performing the copy.</p>

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

### 6.1.8. 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
<code>CL_DEVICE_DOUBLE_FP_CONFIG</code>	<code>cl_device_fp_config</code>	<p>Describes double precision floating-point capability of the OpenCL device. This is a bit-field that describes one or more of the following values:</p> <p><code>CL_FP_DENORM</code> — denorms are supported</p> <p><code>CL_FP_INF_NAN</code> — INF and NaNs are supported</p> <p><code>CL_FP_ROUND_TO_NEAREST</code> — round to nearest even rounding mode supported</p> <p><code>CL_FP_ROUND_TO_ZERO</code> — round to zero rounding mode supported</p> <p><code>CL_FP_ROUND_TO_INF</code> — round to positive and negative infinity rounding modes supported</p> <p><code>CL_FP_FMA</code> — IEEE754-2008 fused multiply-add is supported</p> <p><code>CL_FP_SOFT_FLOAT</code> — Basic floating-point operations (such as addition, subtraction, multiplication) are implemented in software.</p> <p>The required minimum double precision floating-point capability as implemented by this extension is:</p> <p><code>CL_FP_FMA  </code>  <code>CL_FP_ROUND_TO_NEAREST  </code>  <code>CL_FP_ROUND_TO_ZERO  </code>  <code>CL_FP_ROUND_TO_INF  </code>  <code>CL_FP_INF_NAN  </code>  <code>CL_FP_DENORM.</code></p>

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.

### 6.1.9. 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*

<b>Function</b>	<b>Min Accuracy</b>
$x + y$	Correctly rounded
$x - y$	Correctly rounded
$x * y$	Correctly rounded
$1.0 / x$	Correctly rounded
$x / y$	Correctly rounded
<b>acos</b>	<= 4 ulp
<b>acosh</b>	<= 4 ulp
<b>acospi</b>	<= 5 ulp
<b>asin</b>	<= 4 ulp
<b>asinh</b>	<= 4 ulp
<b>asinpi</b>	<= 5 ulp
<b>atan</b>	<= 5 ulp
<b>atanh</b>	<= 5 ulp
<b>atanpi</b>	<= 5 ulp
<b>atan2</b>	<= 6 ulp
<b>atan2pi</b>	<= 6 ulp
<b>cbrt</b>	<= 2 ulp
<b>ceil</b>	Correctly rounded
<b>clamp</b>	0 ulp
<b>copysign</b>	0 ulp
<b>cos</b>	<= 4 ulp
<b>cosh</b>	<= 4 ulp
<b>cospi</b>	<= 4 ulp
<b>cross</b>	absolute error tolerance of 'max * max * (3 * FLT_EPSILON)', where max is the maximum input operand value
<b>degrees</b>	<= 2 ulp
<b>distance</b>	<= 2 * (3 + 0.5 * 1.5 * n) + (0.5 * (n - 1)) ulp, for gentype with vector width n

Function	Min Accuracy
<b>dot</b>	absolute error tolerance of 'max * max * (2 * (n - 1)) * FLT_EPSILON', For vector width n and maximum input operand value 'max'
<b>erfc</b>	<= 16 ulp
<b>erf</b>	<= 16 ulp
<b>exp</b>	<= 3 ulp
<b>exp2</b>	<= 3 ulp
<b>exp10</b>	<= 3 ulp
<b>expm1</b>	<= 3 ulp
<b>fabs</b>	0 ulp
<b>fdim</b>	Correctly rounded
<b>floor</b>	Correctly rounded
<b>fma</b>	Correctly rounded
<b>fmax</b>	0 ulp
<b>fmin</b>	0 ulp
<b>fmod</b>	0 ulp
<b>fract</b>	Correctly rounded
<b>frexp</b>	0 ulp
<b>hypot</b>	<= 4 ulp
<b>ilogb</b>	0 ulp
<b>ldexp</b>	Correctly rounded
<b>length</b>	<= 2 * (3 + 0.5 * 0.5 * n) + (0.5 * (n - 1)) ulp, for gentype with vector width n
<b>log</b>	<= 3 ulp
<b>log2</b>	<= 3 ulp
<b>log10</b>	<= 3 ulp
<b>log1p</b>	<= 2 ulp
<b>logb</b>	0 ulp
<b>mad</b>	Implementation-defined
<b>max</b>	0 ulp
<b>maxmag</b>	0 ulp
<b>min</b>	0 ulp
<b>minmag</b>	0 ulp
<b>mix</b>	Implementation-defined
<b>modf</b>	0 ulp
<b>nan</b>	0 ulp
<b>nextafter</b>	0 ulp



Function	Min Accuracy
<b>normalize</b>	$\leq 2 * (2.5 + 0.5 * 0.5 * n) + (0.5 * (n - 1))$ ulp, for gentype with vector width n
<b>pow(x, y)</b>	$\leq 16$ ulp
<b>pown(x, y)</b>	$\leq 16$ ulp
<b>powr(x, y)</b>	$\leq 16$ ulp
<b>radians</b>	$\leq 2$ ulp
<b>remainder</b>	0 ulp
<b>remquo</b>	0 ulp for the remainder, at least the lower 7 bits of the integral quotient
<b>rint</b>	Correctly rounded
<b>rootn</b>	$\leq 16$ ulp
<b>round</b>	Correctly rounded
<b>rsqrt</b>	$\leq 2$ ulp
<b>sign</b>	0 ulp
<b>sin</b>	$\leq 4$ ulp
<b>sincos</b>	$\leq 4$ ulp for sine and cosine values
<b>sinh</b>	$\leq 4$ ulp
<b>sinpi</b>	$\leq 4$ ulp
<b>smoothstep</b>	Implementation-defined
<b>sqrt</b>	Correctly rounded
<b>step</b>	0 ulp
<b>tan</b>	$\leq 5$ ulp
<b>tanh</b>	$\leq 5$ ulp
<b>tanpi</b>	$\leq 6$ ulp
<b>tgamma</b>	$\leq 16$ ulp
<b>trunc</b>	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. Global Atomics for 32-bit Integers

### 7.1.1. Base Atomics

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

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

## 7.1.2. Extended Atomics

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

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

## 7.2. Local Atomics for 32-bit Integers

### 7.2.1. Base Atomics

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

Function	Description
<code>int atom_add (__local int *p, int val)</code> <code>uint atom_add (__local uint *p, uint val)</code>	Read the 32-bit value (referred to as <i>old</i> ) stored at location pointed by <i>p</i> . Compute ( <i>old</i> + <i>val</i> ) and store result at location pointed by <i>p</i> . The function returns <i>old</i> .

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

## 7.2.2. Extended Atomics

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

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

Function	Description
<pre>int <b>atom_or</b> (__local int *p, int val) uint <b>atom_or</b> (__local uint *p, uint val)</pre>	<p>Read the 32-bit value (referred to as <i>old</i>) stored at location pointed by <i>p</i>. Compute (<math>old \mid val</math>) and store result at location pointed by <i>p</i>. The function returns <i>old</i>.</p>
<pre>int <b>atom_xor</b> (__local int *p, int val) uint <b>atom_xor</b> (__local uint *p, uint val)</pre>	<p>Read the 32-bit value (referred to as <i>old</i>) stored at location pointed by <i>p</i>. Compute (<math>old \wedge val</math>) and store result at location pointed by <i>p</i>. The function returns <i>old</i>.</p>

# 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.

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

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

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

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

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.**

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 extension 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__ rtn
```

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. 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);
```

## 10.3. 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
```

## 10.4. 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
<code>CL_GL_CONTEXT_KHR</code>	<b>0</b> , OpenGL context handle	OpenGL context to associated the OpenCL context with

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

## 10.5. 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**

<b>param_name</b>	<b>Return Type</b>	<b>Information returned in param_value</b>
CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR	<b>cl_device_id</b>	Return the OpenCL device currently associated with the specified OpenGL context.
CL_DEVICES_FOR_GL_CONTEXT_KHR	<b>cl_device_id[]</b>	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 OpenGL 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 OpenGL implementation on the device; or CL\_OUT\_OF\_HOST\_MEMORY if there is a failure to allocate resources required by the OpenGL implementation on the host."

## 10.6. 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 heterogeneous 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. 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

## 11.2. 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 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.

## 11.3. 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 greater 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.

### 11.3.1. 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

<b>GL internal format</b>	<b>CL image format (channel order, channel data type)</b>
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_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 internal format	CL image format (channel order, channel data type)
GL_RG16UI	CL_RG, CL_UNSIGNED_INT16
GL_RG32UI	CL_RG, CL_UNSIGNED_INT32

## 11.4. 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.

## 11.5. 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  $\geq$  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**

<b>cl_gl_texture_info</b>	<b>Return Type</b>	<b>Info. returned in <i>param_value</i></b>
<b>CL_GL_TEXTURE_TARGET</b>	GLenum	The <i>texture_target</i> argument specified in <b>clCreateFromGLTexture</b> .
<b>CL_GL_MIPMAP_LEVEL</b>	GLint	The <i>miplevel</i> argument specified in <b>clCreateFromGLTexture</b> .

**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.

## 11.6. 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. 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 or queue a wait for the 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. If the *event\_wait\_list* and the *event* arguments are not **NULL**, the *event* argument should 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 particular command and can be used to query or queue a wait for the 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. If the *event\_wait\_list* and the *event* arguments are not **NULL**, the *event* argument should 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.

### 11.6.1. 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.

# 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. New Procedures and Functions

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

## 12.3. New Tokens

Returned by `clGetEventInfo` when *param\_name* is `CL_EVENT_COMMAND_TYPE`:

```
CL_COMMAND_GL_FENCE_SYNC_OBJECT_KHR
```

## 12.4. 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. `"

## 12.5. 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. ``

## 12.6. 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. 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)

```

### 13.3. 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

## 13.4. 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 <code>NULL</code> .

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*:

<b>cl_context_info</b>	<b>Return Type</b>	<b>Information returned in param_value</b>
<b>CL_CONTEXT_D3D10_PREFER_SHARED_RESOURCES_KHR</b>	<b>cl_bool</b>	Returns CL_TRUE if Direct3D 10 resources created as shared by setting <i>MiscFlags</i> 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.

## 13.5. 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.

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

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

- CL\_INVALID\_D3D10\_RESOURCE\_KHR if *param\_name* is CL\_MEM\_D3D10\_SUBRESOURCE\_KHR and *image* was not created by the function **clCreateFromD3D10Texture2DKHR**, or **clCreateFromD3D10Texture3DKHR**.

Extend *table 5.9* to include the following entry.

<b>cl_image_info</b>	<b>Return type</b>	<b>Info. returned in param_value</b>
<b>CL_MEM_D3D10_SUBRESOURCE_KHR</b>	UINT	If <i>image</i> was created using <b>clCreateFromD3D10Texture2DKHR</b> , or <b>clCreateFromD3D10Texture3DKHR</b> , returns the <i>subresource</i> argument specified when <i>image</i> 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
```

## 13.6. 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.

### 13.6.1. 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 29. Direct3D 10 object types that may be used by **clGetDeviceIDsFromD3D10KHR**

<b>cl_d3d_device_source_khr</b>	<b>Type of <i>d3d_object</i></b>
CL_D3D10_DEVICE_KHR	ID3D10Device *
CL_D3D10_DXGI_ADAPTER_KHR	IDXGIAdapter *

Table 30. Sets of devices querable using **clGetDeviceIDsFromD3D10KHR**

<b>cl_d3d_device_set_khr</b>	<b>Devices returned in <i>devices</i></b>
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.

### 13.6.2. 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.

### 13.6.3. 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.

### 13.6.4. 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_IMMUTABLE` flag, 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_IMMUTABLE` flag, 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 31. Direct3D 10 formats and corresponding OpenCL image formats

<b>DXGI format</b>	<b>CL image format (channel order, channel data type)</b>
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_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	CL image format (channel order, channel data type)
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

### 13.6.5. 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*.

### 13.6.6. 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 call attempting to use that OpenCL memory object will return `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 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. If the *event\_wait\_list* and the *event* arguments are not **NULL**, the *event* argument should 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. The event specified by *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. If the *event\_wait\_list* and the *event* arguments are not `NULL`, the *event* argument should 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. 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? If so the command might correspond to the 2D / 3D versions:

```
cl_mem clCreateFromD3D10Texture1D(cl_context context,
                                  cl_mem_flags flags,
                                  ID3D10Texture2D *resource,
                                  UINT subresource,
                                  cl_int *errcode_ret)
```

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. 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)

```

## 14.3. 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

## 14.4. 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 <code>NULL</code> .

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*:

<b>cl_context_info</b>	<b>Return Type</b>	<b>Information returned in param_value</b>
<b>CL_CONTEXT_D3D11_PREFER_SHARED_RESOURCES_KHR</b>	<b>cl_bool</b>	Returns CL_TRUE if Direct3D 11 resources created as shared by setting <i>MiscFlags</i> 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.

## 14.5. 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.

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

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

- CL\_INVALID\_D3D11\_RESOURCE\_KHR if *param\_name* is CL\_MEM\_D3D11\_SUBRESOURCE\_KHR and *image* was not created by the function **clCreateFromD3D11Texture2DKHR**, or **clCreateFromD3D11Texture3DKHR**.

Extend *table 5.9* to include the following entry.

<b>cl_image_info</b>	<b>Return type</b>	<b>Info. returned in param_value</b>
<b>CL_MEM_D3D11_SUBRESOURCE_KHR</b>	UINT	If <i>image</i> was created using <b>clCreateFromD3D11Texture2DKHR</b> , or <b>clCreateFromD3D11Texture3DKHR</b> , returns the <i>subresource</i> argument specified when <i>image</i> 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
```

## 14.6. 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.

### 14.6.1. 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.

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

<b>cl_d3d_device_source_khr</b>	<b>Type of <i>d3d_object</i></b>
CL_D3D11_DEVICE_KHR	ID3D11Device *
CL_D3D11_DXGI_ADAPTER_KHR	IDXGIAdapter *

Table 33. Sets of devices querable using **clGetDeviceIDsFromD3D11KHR**

<b>cl_d3d_device_set_khr</b>	<b>Devices returned in <i>devices</i></b>
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.

## 14.6.2. 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.

### 14.6.3. 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.

### 14.6.4. 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_IMMUTABLE` flag, 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 10 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_IMMUTABLE` flag, 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 34. Direct3D 11 formats and corresponding OpenCL image formats*

<b>DXGI format</b>	<b>CL image format (channel order, channel data type)</b>
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_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	CL image format (channel order, channel data type)
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

### 14.6.5. 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*.

### 14.6.6. 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 call attempting to use that OpenCL memory object will return `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 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. If the *event\_wait\_list* and the *event* arguments are not **NULL**, the *event* argument should 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. The event specified by *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. If the *event\_wait\_list* and the *event* arguments are not `NULL`, the *event* argument should 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.

# 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. 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)
```



## 15.3. 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

## 15.4. 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_ADAPTER_D3D9_EX_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\_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**.

## 15.5. 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:

<b>cl_mem_info</b>	<b>Return type</b>	<b>Info. returned in <i>param_value</i></b>
<b>CL_MEM_DX9_MEDIA_ADAPTER_TYPE_KHR</b>	<b>cl_dx9_media_adapter_type_khr</b>	Returns the <i>cl_dx9_media_adapter_type_khr</i> argument value specified when <i>memobj</i> is created using <b>clCreateFromDX9MediaSurfaceKHR</b> .
<b>CL_MEM_DX9_MEDIA_SURFACE_INFO_KHR</b>	<b>cl_dx9_surface_info_khr</b>	Returns the <i>cl_dx9_surface_info_khr</i> argument value specified when <i>memobj</i> is created using <b>clCreateFromDX9MediaSurfaceKHR</b> .

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.

<b>cl_image_info</b>	<b>Return type</b>	<b>Info. returned in <i>param_value</i></b>
<b>CL_IMAGE_DX9_MEDIA_PLANE_KHR</b>	<b>cl_uint</b>	Returns the <i>plane</i> argument value specified when <i>memobj</i> is created using <b>clCreateFromDX9MediaSurfaceKHR</b> .

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
```

## 15.6. 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.

### 15.6.1. 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_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)
```

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 35. *cl\_dx9\_media\_adapter\_type\_khr* values

<code>cl_dx9_media_adapter_type_khr</code>	Type of media adapters
<code>CL_ADAPTER_D3D9_KHR</code>	IDirect3DDevice9 *
<code>CL_ADAPTER_D3D9EX_KHR</code>	IDirect3DDevice9Ex *
<code>CL_ADAPTER_DXVA_KHR</code>	IDXVAHD_Device *

Table 36. *cl\_dx9\_media\_adapter\_set\_khr* values

<code>cl_dx9_media_adapter_set_khr</code>	Description
<code>CL_PREFERRED_DEVICES_FOR_DX9_MEDIA_ADAPTER_KHR</code>	The preferred OpenCL devices associated with the media adapter.
<code>CL_ALL_DEVICES_FOR_MEDIA_DX9_ADAPTER_KHR</code>	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.

## 15.6.2. 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 this 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.

### 15.6.3. 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](#).

### 15.6.4. 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 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. If the *event\_wait\_list* and the *event* arguments are not `NULL`, the *event* argument should 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. The event specified by *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. If the *event\_wait\_list* and the *event* arguments are not **NULL**, the *event* argument should not refer to an element of the *event\_wait\_list* array.

**clEnqueueReleaseDX9MediaSurfaceKHR** 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* > 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.

### 15.6.5. 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 DirectX9 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 37. YUV FourCC codes and corresponding OpenCL image format

<b>FOUR CC code</b>	<b>CL image format (channel order, channel data type)</b>
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 38. Direct3D formats and corresponding OpenCL image formats

<b>D3D format</b>	<b>CL image format (channel order, channel data type)</b>
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\_A32B32G32R32F 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. 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 <code>channel_order</code>	
<code>CL_DEPTH</code> . This format can only be used if channel data type = <code>CL_UNORM_INT16</code> or <code>CL_FLOAT</code> .	
Image Channel Data Type	Description
<code>CL_UNORM_INT16</code>	Each channel component is a normalized unsigned 16-bit integer value
<code>CL_FLOAT</code>	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 39. Required Image Formats for `cl_khr_depth_images`

<code>num_channels</code>	<code>channel_order</code>	<code>channel_data_type</code>
1	<code>CL_DEPTH</code>	<code>CL_UNORM_INT</code> <code>CL_FLOAT</code>

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.

## 16.2. 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
<code>image2d_depth_t</code>	A 2D depth image. Refer to <i>section 6.12.14</i> for a detailed description of the built-in functions that use this type.
<code>image2d_array_depth_t</code>	A 2D depth image array. Refer to <i>section 6.12.14</i> 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* - BuiltIn Image Read Functions:

Function	Description
<code>float read_imagef(read_only image2d_depth_t image, sampler_t sampler, int2 coord)</code> <code>float read_imagef(read_only image2d_depth_t image, sampler_t sampler, float2 coord)</code>	<p>Use the coordinate (<i>coord.x</i>, <i>coord.y</i>) to do an element lookup in the 2D depth image object specified by <i>image</i>.</p> <p><b>read_imagef</b> returns a floating-point value in the range [0.0, 1.0] for depth image objects created with <i>image_channel_data_type</i> set to <code>CL_UNORM_INT16</code> or <code>CL_UNORM_INT24</code>.</p> <p><b>read_imagef</b> returns a floating-point value for depth image objects created with <i>image_channel_data_type</i> set to <code>CL_FLOAT</code>.</p> <p>The <b>read_imagef</b> calls that take integer coordinates must use a sampler with filter mode set to <code>CLK_FILTER_NEAREST</code>, normalized coordinates set to <code>CLK_NORMALIZED_COORDS_FALSE</code> and addressing mode set to <code>CLK_ADDRESS_CLAMP_TO_EDGE</code>, <code>CLK_ADDRESS_CLAMP</code> or <code>CLK_ADDRESS_NONE</code>; otherwise the values returned are undefined.</p> <p>Values returned by <b>read_imagef</b> for depth image objects with <i>image_channel_data_type</i> values not specified in the description above are undefined.</p>

Function	Description
<p>float <b>read_imagef</b>(read_only image2d_array_depth_t <i>image</i>, sampler_t <i>sampler</i>, int4 <i>coord</i>)</p> <p>float <b>read_imagef</b>(read_only image2d_array_depth_t <i>image</i>, sampler_t <i>sampler</i>, float4 <i>coord</i>)</p>	<p>Use <i>coord.xy</i> to do an element lookup in the 2D image identified by <i>coord.z</i> in the 2D depth image array specified by <i>image</i>.</p> <p><b>read_imagef</b> returns a floating-point value in the range [0.0, 1.0] for depth image objects created with <i>image_channel_data_type</i> set to <code>CL_UNORM_INT16</code> or <code>CL_UNORM_INT24</code>.</p> <p><b>read_imagef</b> returns a floating-point value for depth image objects created with <i>image_channel_data_type</i> set to <code>CL_FLOAT</code>.</p> <p>The <b>read_imagef</b> calls that take integer coordinates must use a sampler with filter mode set to <code>CLK_FILTER_NEAREST</code>, normalized coordinates set to <code>CLK_NORMALIZED_COORDS_FALSE</code> and addressing mode set to <code>CLK_ADDRESS_CLAMP_TO_EDGE</code>, <code>CLK_ADDRESS_CLAMP</code> or <code>CLK_ADDRESS_NONE</code>; otherwise the values returned are undefined.</p> <p>Values returned by <b>read_imagef</b> for image objects with <i>image_channel_data_type</i> values not specified in the description above are undefined.</p>

Add the following built-in functions to section 6.12.14.3 - BuiltIn Image Sampler-less Read Functions:

Function	Description
<p>float <b>read_imagef</b>(image2d_depth_t <i>image</i>, int2 <i>coord</i>)</p>	<p>Use the coordinate (<i>coord.x</i>, <i>coord.y</i>) to do an element lookup in the 2D depth image object specified by <i>image</i>.</p> <p><b>read_imagef</b> returns a floating-point value in the range [0.0, 1.0] for depth image objects created with <i>image_channel_data_type</i> set to <code>CL_UNORM_INT16</code> or <code>CL_UNORM_INT24</code>.</p> <p><b>read_imagef</b> returns a floating-point value for depth image objects created with <i>image_channel_data_type</i> set to <code>CL_FLOAT</code>.</p> <p>Values returned by <b>read_imagef</b> for image objects with <i>image_channel_data_type</i> values not specified in the description above are undefined.</p>

Function	Description
float <b>read_imagef</b> (image2d_array_depth_t <i>image</i> , int4 <i>coord</i> )	<p>Use <i>coord.xy</i> to do an element lookup in the 2D image identified by <i>coord.z</i> in the 2D depth image array specified by <i>image</i>.</p> <p><b>read_imagef</b> returns a floating-point value in the range [0.0, 1.0] for depth image objects created with <i>image_channel_data_type</i> set to <b>CL_UNORM_INT16</b> or <b>CL_UNORM_INT24</b>.</p> <p><b>read_imagef</b> returns a floating-point value for depth image objects created with <i>image_channel_data_type</i> set to <b>CL_FLOAT</b>.</p> <p>Values returned by <b>read_imagef</b> for image objects with <i>image_channel_data_type</i> values not specified in the description above are undefined.</p>

Add the following built-in functions to section 6.12.14.4 – BuiltIn Image Write Functions:

Function	Description
void <b>write_imagef</b> (image2d_depth_t <i>image</i> , int2 <i>coord</i> , float <i>depth</i> )	<p>Write <i>depth</i> value to location specified by <i>coord.xy</i> in the 2D depth image object specified by <i>image</i>. Appropriate data format conversion to the specified image format is done before writing the depth value. <i>coord.x</i> and <i>coord.y</i> are considered to be unnormalized coordinates, and must be in the range [0, image width-1], and [0, image height-1], respectively.</p> <p><b>write_imagef</b> can only be used with image objects created with <i>image_channel_data_type</i> set to <b>CL_UNORM_INT16</b>, <b>CL_UNORM_INT24</b> or <b>CL_FLOAT</b>. Appropriate data format conversion will be done to convert depth value from a floating-point value to actual data format associated with the image.</p> <p>The behavior of <b>write_imagef</b>, <b>write_imagei</b> and <b>write_imageui</b> for image objects created with <i>image_channel_data_type</i> 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.</p>

Function	Description
void <b>write_imagef</b> (image2d_array_depth_t <i>image</i> , int4 <i>coord</i> , float <i>depth</i> )	<p>Write <i>depth</i> value to location specified by <i>coord.xy</i> in the 2D image identified by <i>coord.z</i> in the 2D depth image array specified by <i>image</i>. Appropriate data format conversion to the specified image format is done before writing the depth value. <i>coord.x</i>, <i>coord.y</i> and <i>coord.z</i> 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.</p> <p><b>write_imagef</b> can only be used with image objects created with <i>image_channel_data_type</i> set to <b>CL_UNORM_INT16</b>, <b>CL_UNORM_INT24</b> or <b>CL_FLOAT</b>. Appropriate data format conversion will be done to convert depth value from a floating-point value to actual data format associated with the image.</p> <p>The behavior of <b>write_imagef</b>, <b>write_imagei</b> and <b>write_imageui</b> for image objects created with <i>image_channel_data_type</i> values not specified in the description above or with (<i>x</i>, <i>y</i>, <i>z</i>) 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.</p>

Add the following built-in functions to section 6.12.14.5 – BuiltIn Image Query Functions:

Function	Description
int <b>get_image_width</b> (image2d_depth_t <i>image</i> ) int <b>get_image_width</b> (image2d_array_dept h_t <i>image</i> )	Return the image width in pixels.
int <b>get_image_height</b> (image2d_depth_t <i>image</i> ) int <b>get_image_height</b> (image2d_array_dep th_t <i>image</i> )	Return the image height in pixels.
int <b>get_image_channel_data_type</b> (image 2d_depth_t <i>image</i> ) int <b>get_image_channel_data_type</b> (image 2d_array_depth_t <i>image</i> )	Return the channel data type. Valid values are:  <b>CLK_UNORM_INT16</b> <b>CLK_FLOAT</b>



int <b>get_image_channel_order</b> (image2d_d eph_t <i>image</i> ) int <b>get_image_channel_order</b> (image2d_a rray_depth_t <i>image</i> )	Return the image channel order. Valid values are:  <b>CLK_DEPTH</b>
int2 <b>get_image_dim</b> (image2d_depth_t <i>image</i> ) int2 <b>get_image_dim</b> (image2d_array_depth _t <i>image</i> )	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 <b>get_image_array_size</b> (image2d_array _depth_t <i>image</i> )	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 **CLK\_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. 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 <code>channel_order</code>
<code>CL_DEPTH_STENCIL</code> . This format can only be used if channel data type = <code>CL_UNORM_INT24</code> or <code>CL_FLOAT</code> .

Image Channel Data Type	Description
<code>CL_UNORM_INT24</code>	Each channel component is a normalized unsigned 24-bit integer value
<code>CL_FLOAT</code>	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 40. Required Image Formats for `cl_khr_gl_depth_images`

<code>num_channels</code>	<code>channel_order</code>	<code>channel_data_type</code>	<code>read / write</code>
1	<code>CL_DEPTH_STENCIL</code>	<code>CL_UNORM_INT24</code> <code>CL_FLOAT</code>	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.

## 17.2. 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.

<b>GL internal format</b>	<b>CL image format (channel order, channel data type)</b>
<code>GL_DEPTH_COMPONENT32F</code>	<code>CL_DEPTH, CL_FLOAT</code>
<code>GL_DEPTH_COMPONENT16</code>	<code>CL_DEPTH, CL_UNORM_INT16</code>
<code>GL_DEPTH24_STENCIL8</code>	<code>CL_DEPTH_STENCIL, CL_UNORM_INT24</code>
<code>GL_DEPTH32F_STENCIL8</code>	<code>CL_DEPTH_STENCIL, CL_FLOAT</code>

# 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. 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`](#):

<code>cl_gl_texture_info</code>	Return Type	Info. returned in <i>param_value</i>
<code>CL_GL_NUM_SAMPLES</code>	<code>GLsizei</code>	The <i>samples</i> argument passed to <code>glTexImage2DMultisample</code> or <code>glTexImage3DMultisample</code> .  If <i>image</i> is not a MSAA texture, 1 is returned.

## 18.2. 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.”

## 18.3. 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
<code>image2d_msaa_t</code>	A 2D multi-sample color image. Refer to <i>section 6.13.14</i> for a detailed description of the built-in functions that use this type.
<code>image2d_array_msaa_t</code>	A 2D multi-sample color image array. Refer to <i>section 6.13.14</i> for a detailed description of the built-in functions that use this type.
<code>image2d_msaa_depth_t</code>	A 2D multi-sample depth image. Refer to <i>section 6.13.14</i> for a detailed description of the built-in functions that use this type.
<code>image2d_array_msaa_depth_t</code>	A 2D multi-sample depth image array. Refer to <i>section 6.13.14</i> 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* — BuiltIn 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_msa_t image,
                 int2 coord,
                 int sample)

uint4 read_imageui(image2d_msa_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_msa_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_msa_t image,
                 int4 coord,
                 int sample)

uint4 read_imageui(image2d_array_msa_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 — BuiltIn 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. New Procedures and Functions

```
cl_event clCreateEventFromEGLSyncKHR(cl_context context,  
                                     CeglSyncKHR sync,  
                                     CeglDisplayKHR display,  
                                     cl_int *errcode_ret);
```

## 19.3. 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
```

## 19.4. 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.``

## 19.5. 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.``

## 19.6. Issues

Most issues are shared with `cl_khr_egl_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. New Procedures and Functions

```
cl_mem clCreateFromEGLImageKHR(cl_context context,
                               CeglDisplayKHR display,
                               CeglImageKHR 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)
```

## 20.3. 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
```

## 20.4. 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,
                               CeglDisplayKHR display,
                               CeglImageKHR 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 `CeglDisplayKHR`.

*image* should be of type `EGLImageKHR`, cast into the type `CeglImageKHR`. 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. `"

### 20.4.1. 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.

### 20.4.2. 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 `clEnqueueReleaseGLObjets`, 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.

### 20.4.3. 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 call attempting to use that OpenCL memory object will return `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 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.

**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 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.

**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. 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. 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.

<code>cl_device_info</code>	Return Type	Description
<code>CL_DEVICE_IMAGE_PITCH_ALIGNMENT_KHR</code>	<code>cl_uint</code>	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.
<code>CL_DEVICE_IMAGE_BASE_ADDRESS_ALIGNMENT_KHR</code>	<code>cl_uint</code>	This query should be used when an image is created from a buffer which was created using <code>CL_MEM_USE_HOST_PTR</code> . The value returned must be a power of 2.  This query specifies the minimum alignment in pixels of the <i>host_ptr</i> specified to <code>clCreateBuffer</code> .  If the device does not support images, this value should be 0.

## 21.2. 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. Additions to Chapter 4 of the OpenCL 2.2 Specification

Add a new context property to *table 4.5* in *section 4.4*.

<code>cl_context_properties</code> enum	Property value	Description
<code>CL_CONTEXT_MEMORY_INITIALIZE_KHR</code>	<code>cl_context_memory_initialize_khr</code>	Describes which memory types for the context must be initialized. This is a bit-field, where the following values are currently supported:  <code>CL_CONTEXT_MEMORY_INITIALIZE_LOCAL_KHR</code> — Initialize local memory to zeros.  <code>CL_CONTEXT_MEMORY_INITIALIZE_PRIVATE_KHR</code> — Initialize private memory to zeros.

## 22.2. 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. Additions to Chapter 4 of the OpenCL 2.2 Specification

Add a new device property to *table 4.3* in *section 4.2*.

<code>cl_device_info</code>	Return Type	Description
<code>CL_DEVICE_TERMINATE_CAPABILITY_KHR</code>	<code>cl_device_terminate_capability_khr</code>	Describes the termination capability of the OpenCL device. This is a bit-field, where the following values are currently supported:  <code>CL_DEVICE_TERMINATE_CAPABILITY_CONTEXT_KHR</code> - Indicates that context termination is supported.

Add a new context property to *table 4.5* in *section 4.4*.

<code>cl_context_properties enum</code>	Property value	Description
<code>CL_CONTEXT_TERMINATE_KHR</code>	<code>cl_bool</code>	Specifies whether the context can be terminated. The default value is <code>CL_FALSE</code> .

`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_TERMINATED_KHR`. Event objects can be queried using **`clGetEventInfo`**. Event callbacks can be registered and registered event callbacks will be called with *event\_command\_exec\_status* set to `CL_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. Additions to Chapter 4 of the OpenCL 2.2 Specification

Add a new device property to *table 4.3* in *section 4.2*:

cl_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.

## 24.2. 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* (**clGetProgramBuildInfo**) to add a potential value CL\_PROGRAM\_BINARY\_TYPE\_INTERMEDIATE:

<code>cl_program_build_info</code>	Return Type	Info. returned in <i>param_value</i>
<code>CL_PROGRAM_BINARY_TYPE</code>	<code>cl_program_binary_type</code>	<p><code>CL_PROGRAM_BINARY_TYPE_INTERMEDIATE</code> — An intermediate (non-source) representation for the program is loaded as a binary. The program must be further processed with <code>clCompileProgram</code> or <code>clBuildProgram</code>.</p> <p>If processed with <code>clCompileProgram</code>, the result will be a binary of type <code>CL_PROGRAM_BINARY_TYPE_COMPILED_OBJECT</code> or <code>CL_PROGRAM_BINARY_TYPE_LIBRARY</code>. If processed with <code>clBuildProgram</code>, the result will be a binary of type <code>CL_PROGRAM_BINARY_TYPE_EXECUTABLE</code>.</p>

#### 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 5.9.3 — 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. New Procedures and Functions

```
cl_program clCreateProgramWithILKHR(cl_context context,
                                   const void *il,
                                   size_t length,
                                   cl_int *errcode_ret);
```

## 25.2. 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
```

## 25.3. 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."

## 25.4. Additions to Chapter 4 of the OpenCL 2.0 Specification

Add a new device property to **Table 4.3 OpenCL Device Queries**:

cl_device_info	Return Type	Description
CL_DEVICE_IL_VERSION_KHR	char[]	<p>The intermediate languages that are be supported by <b>clCreateProgramWithILKHR</b> for this device.</p> <p>Returns a space separated list of IL version strings of the form:</p> <p>&lt;IL_Prefix&gt;_&lt;Major_version&gt;.&lt;Minor_version&gt;</p> <p>A device that supports the <b>cl_khr_il_program</b> extension must support the “SPIR-V” IL prefix.</p>

## 25.5. 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 language.

*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:

cl_program_info	Return Type	Info returned in <i>param_value</i>
CL_PROGRAM_SOURCE	char[]	<p>Return the program source code specified by <b>clCreateProgramWithSource</b>. The source string returned is a concatenation of all source strings specified to <b>clCreateProgramWithSource</b> with a null terminator. The concatenation strips any nulls in the original source strings.</p> <p>If program is created using <b>clCreateProgramWithBinary</b>, <b>clCreateProgramWithBuiltInKernels</b>, or <b>clCreateProgramWithILKHR</b> 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.</p> <p>The actual number of characters that represents the program source code including the null terminator is returned in <i>param_value_size_ret</i>.</p>
CL_PROGRAM_IL_KHR	unsigned char[]	<p>Returns the program IL for programs created with <b>clCreateProgramWithILKHR</b>.</p> <p>If program is created with <b>clCreateProgramWithSource</b>, <b>clCreateProgramWithBinary</b>, or <b>clCreateProgramWithBuiltInKernels</b>, the memory pointed to by <i>param_value</i> will be unchanged and <i>param_value_size_ret</i> will be set to zero.</p>



# 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. New API Functions

```
cl_command_queue clCreateCommandQueueWithPropertiesKHR(  
    cl_context context,  
    cl_device_id device,  
    const cl_queue_properties_khr *properties,  
    cl_int *errcode_ret)
```

## 26.3. New API Types

```
typedef cl_bitfield cl_queue_properties_khr;
```

## 26.4. 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*

<b>cl_command_queue_info</b>	<b>Return Type</b>	<b>Information returned in param_value</b>
<b>CL_QUEUE_PROPERTIES</b>	cl_command_queue_properties	Returns the currently specified properties for the command-queue. These properties are specified by the <i>properties</i> argument in <b>clCreateCommandQueue</b> , or by the CL_QUEUE_PROPERTIES property value in <b>clCreateCommandQueueWithPropertiesKHR</b> .

### (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 *cl\_queue\_properties\_khr* values and description

Queue Properties	Property Value	Description
CL_QUEUE_PROPERTIES	cl_bitfield	<p>This is a bitfield and can be set to a combination of the following values:</p> <p>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.</p> <p>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.</p> <p>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.</p>

*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. 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. Subgroups

This section describes the `cl_khr_subgroups` extension.

This extension adds support for implementation-controlled groups of work items, known as subgroups. Subgroups behave similarly to work groups and have their own sets of builtins and synchronization primitives, but subgroups 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.

Subgroups became a core feature in OpenCL 2.1.

## 28.1. Additions to Chapter 5 of the OpenCL 2.0 Specification

The function

```
cl_int clGetKernelSubGroupInfoKHR(cl_kernel kernel,
                                   cl_device_id 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 Subgroup 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  $\geq$  size of return type as described in the [Kernel Object Subgroup 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 41. `clGetKernelSubGroupInfoKHR` parameter queries

<b>cl_kernel_sub_group_info</b>	<b>Input Type</b>	<b>Return Type</b>	<b>Info. returned in <i>param_value</i></b>
<b>CL_KERNEL_MAX_SUBGROUP_SIZE_FOR_NDRANGE_KHR</b>	size_t *	size_t	<p>Returns the maximum sub-group size for this kernel. All sub-groups must be the same size, while the last subgroup in any work-group (i.e. the subgroup with the maximum index) could be the same or smaller size.</p> <p>The <i>input_value</i> 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 <i>input_value_size</i>.</p>

<b>cl_kernel_sub_group_info</b>	<b>Input Type</b>	<b>Return Type</b>	<b>Info. returned in <i>param_value</i></b>
<b>CL_KERNEL_SUB_GROUP_COUNT_FOR_NDRANGE_KHR</b>	size_t *	size_t	<p>Returns the number of sub-groups that will be present in each workgroup for a given local work size. All workgroups, apart from the last workgroup in each dimension in the presence of non-uniform work-group sizes, will have the same number of sub-groups.</p> <p>The <i>input_value</i> 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 <i>input_value_size</i>.</p>

**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 Subgroup Queries](#) table and *param\_value* is not **NULL**.
- CL\_INVALID\_VALUE if *param\_name* is CL\_KERNEL\_MAX\_SUB\_GROUP\_SIZE\_FOR\_NDRANGE 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.

## 28.2. Additions to Chapter 6 of the OpenCL 2.0 C Specification

### 28.2.1. Additions to section 6.13.1 — Work Item Functions

Function	Description
<code>uint get_sub_group_size ()</code>	Returns the number of work items in the subgroup. This value is no more than the maximum subgroup 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 subgroup.
<code>uint get_max_sub_group_size ()</code>	Returns the maximum size of a subgroup within the dispatch. This value will be invariant for a given set of dispatch dimensions and a kernel object compiled for a given device.
<code>uint get_num_sub_groups ()</code>	Returns the number of subgroups 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 <code>global_work_size</code> values specified to <code>clEnqueueNDRangeKernel</code> are not evenly divisible by the <code>local_work_size</code> 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.
<code>uint get_enqueued_num_sub_groups ()</code>	Returns the same value as that returned by <code>get_num_sub_groups</code> 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 subgroups in each of the work groups that make up the uniform region of the global range.
<code>uint get_sub_group_id ()</code>	<code>get_sub_group_id</code> returns the subgroup ID which is a number from 0 .. <code>get_num_sub_groups() - 1</code> .  For <code>clEnqueueTask</code> , this returns 0.
<code>uint get_sub_group_local_id ()</code>	Returns the unique work item ID within the current subgroup. The mapping from <code>get_local_id(dimindx)</code> to <code>get_sub_group_local_id</code> will be invariant for the lifetime of the work group.



## 28.2.2. Additions to section 6.13.8 — Synchronization Functions

Function	Description
<code>void sub_group_barrier (cl_mem_fence_flags flags)</code>  <code>void sub_group_barrier (cl_mem_fence_flags flags, memory_scope scope)</code>	<p>All work items in a subgroup executing the kernel on a processor must execute this function before any are allowed to continue execution beyond the subgroup barrier. This function must be encountered by all work items in a subgroup executing the kernel. These rules apply to ND-ranges implemented with uniform and non-uniform work groups.</p> <p>If <b>subgroup_barrier</b> is inside a conditional statement, then all work items within the subgroup must enter the conditional if any work item in the subgroup enters the conditional statement and executes the <b>subgroup_barrier</b>.</p> <p>If <b>subgroup_barrier</b> is inside a loop, all work items within the subgroup must execute the <b>subgroup_barrier</b> for each iteration of the loop before any are allowed to continue execution beyond the <b>subgroup_barrier</b>.</p> <p>The <b>subgroup_barrier</b> function also queues a memory fence (reads and writes) to ensure correct ordering of memory operations to local or global memory.</p> <p>The flags argument specifies the memory address space and can be set to a combination of the following values:</p> <p>CLK_LOCAL_MEM_FENCE - The <b>subgroup_barrier</b> 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.</p> <p>CLK_GLOBAL_MEM_FENCE — The <b>subgroup_barrier</b> 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.</p> <p>CLK_IMAGE_MEM_FENCE — The <b>subgroup_barrier</b> 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.</p>

## 28.2.3. 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 subgroup. This memory scope can be used when performing atomic operations to global or local memory.

#### 28.2.4. Additions to section 6.13.15 — Work Group Functions

The OpenCL C programming language implements the following built-in functions that operate on a subgroup level. These built-in functions must be encountered by all work items in a subgroup executing the kernel. We use the generic type name `gentype` to indicate the built-in data types `half` (only if the `cl_khr_fp16` extension is supported), `int`, `uint`, `long`, `ulong`, `float` or `double` (only if double precision is supported) as the type for the arguments.

Function	Description
<code>int sub_group_all (int predicate)</code>	Evaluates <i>predicate</i> for all work items in the subgroup and returns a non-zero value if <i>predicate</i> evaluates to non-zero for all work items in the subgroup.
<code>int sub_group_any (int predicate)</code>	Evaluates <i>predicate</i> for all work items in the subgroup and returns a non-zero value if <i>predicate</i> evaluates to non-zero for any work items in the subgroup.
<code>gentype sub_group_broadcast (gentype x, uint sub_group_local_id)</code>	Broadcast the value of <i>x</i> for work item identified by <i>sub_group_local_id</i> (value returned by <code>get_sub_group_local_id</code> ) to all work items in the subgroup.  <i>sub_group_local_id</i> must be the same value for all work items in the subgroup.
<code>gentype sub_group_reduce_&lt;op&gt; (gentype x)</code>	Return result of reduction operation specified by <code>&lt;op&gt;</code> for all values of <i>x</i> specified by work items in a subgroup.
<code>gentype sub_group_scan_exclusive_&lt;op&gt; (gentype x)</code>	Do an exclusive scan operation specified by <code>&lt;op&gt;</code> of all values specified by work items in a subgroup. The scan results are returned for each work item.  The scan order is defined by increasing subgroup local ID within the subgroup.
<code>gentype sub_group_scan_inclusive_&lt;op&gt; (gentype x)</code>	Do an inclusive scan operation specified by <code>&lt;op&gt;</code> of all values specified by work items in a subgroup. The scan results are returned for each work item.  The scan order is defined by increasing subgroup local ID within the subgroup.

## 28.2.5. Additions to section 6.13.16 — Pipe Functions

The OpenCL C programming language implements the following built-in pipe functions that operate at a subgroup level. These built-in functions must be encountered by all work items in a subgroup 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
<pre>reserve_id_t sub_group_reserve_read_pipe ( read_only pipe gentype pipe, uint num_packets)</pre>	Reserve <i>num_packets</i> entries for reading from or writing to <i>pipe</i> . Returns a valid non-zero reservation ID if the reservation is successful and 0 otherwise.
<pre>reserve_id_t sub_group_reserve_write_pipe ( write_only pipe gentype pipe, uint num_packets)</pre>	The reserved pipe entries are referred to by indices that go from 0 ... <i>num_packets</i> - 1.
<pre>void sub_group_commit_read_pipe ( read_only pipe gentype pipe, reserve_id_t reserve_id)</pre> <pre>void sub_group_commit_write_pipe ( write_only pipe gentype pipe, reserve_id_t reserve_id)</pre>	Indicates that all reads and writes to <i>num_packets</i> associated with reservation <i>reserve_id</i> are completed.

Note: Reservations made by a subgroup are ordered in the pipe as they are ordered in the program. Reservations made by different subgroups that belong to the same work group can be ordered using subgroup synchronization. The order of subgroup based reservations that belong to different work groups is implementation defined.

## 28.2.6. Additions to section 6.13.17.6 — Enqueuing Kernels (Kernel Query Functions)

Built-in Function	Description
<pre>uint get_kernel_sub_group_count_for_ndrange ( const ndrange_t ndrange, void (^block)(void));</pre> <pre>uint get_kernel_sub_group_count_for_ndrange ( const ndrange_t ndrange, void (^block)(local void *, ...));</pre>	<p>Returns the number of subgroups 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 <i>ndrange</i> and <i>block</i>.</p> <p><i>block</i> specifies the block to be enqueued.</p>

Built-in Function	Description
<pre>uint <b>get_kernel_max_sub_group_size_for_ndrange</b> ( const ndrange_t <i>ndrange</i>, void (^block)(void));  uint <b>get_kernel_max_sub_group_size_for_ndrange</b> ( const ndrange_t <i>ndrange</i>, void (^block)(local void *, ...));</pre>	<p>Returns the maximum subgroup size for a block.</p>

# 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. Additions to Chapter 5 of the OpenCL 2.2 Specification

### 29.1.1. 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 *cl\_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.

### 29.1.2. 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`.

<code>cl_sampler_properties</code> enum	Property Value	Default Value
<code>CL_SAMPLER_MIP_FILTER_MODE_KHR</code>	<code>cl_filter_mode</code>	<code>CL_FILTER_NEAREST</code>
<code>CL_SAMPLER_LOD_MIN_KHR</code>	<code>cl_float</code>	0.0f
<code>CL_SAMPLER_LOD_MAX_KHR</code>	<code>cl_float</code>	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.

## 29.2. 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 `CL_MEM_WRITE_ONLY` is specified in the *flags* argument.

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.

# 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 user guides associated with each implementation which supports this extension will describe the scheduling behavior guarantees.

## 31.1. 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`).

## 32.1. 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 Subgroups

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 subgroups named barriers in the SPIR-V intermediate representation, and to the OpenCL C++ specification for descriptions of the subgroup named barrier built-in functions in the OpenCL C++ kernel language.

Add to *table 4.3*:

<b>cl_device_info</b>	<b>Return Type</b>	<b>Description</b>
<b>CL_DEVICE_MAX_NAMED_BARRIER_COUNT_KHR</b>	<b>cl_uint</b>	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. 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`

# Index

## @

$0.5 * n) + (0.5 * (n - 1)$ , [41](#)

$0.5 * n) + (0.5 * (n - 1)$ , [62](#), [63](#)

$1.5 * n) + (0.5 * (n - 1)$ , [61](#)

## C

[clCreateEventFromEGLSyncKHR](#), [154](#)

[clCreateEventFromGLsyncKHR](#), [92](#)

[clCreateFromD3D10BufferKHR](#), [102](#)

[clCreateFromD3D10Texture2DKHR](#), [102](#)

[clCreateFromD3D10Texture3DKHR](#), [103](#)

[clCreateFromD3D11BufferKHR](#), [118](#)

[clCreateFromD3D11Texture2DKHR](#), [118](#)

[clCreateFromD3D11Texture3DKHR](#), [119](#)

[clCreateFromDX9MediaSurfaceKHR](#), [131](#)

[clCreateFromEGLImageKHR](#), [158](#)

[clCreateFromGLBuffer](#), [79](#)

[clCreateFromGLRenderbuffer](#), [84](#)

[clCreateFromGLTexture](#), [80](#)

[clCreateProgramWithILKHR](#), [172](#)

[clEnqueueAcquireD3D10ObjectsKHR](#), [106](#)

[clEnqueueAcquireD3D11ObjectsKHR](#), [122](#)

[clEnqueueAcquireDX9MediaSurfacesKHR](#), [133](#)

[clEnqueueAcquireEGLObjectsKHR](#), [160](#)

[clEnqueueAcquireGLObjects](#), [87](#)

[clEnqueueReleaseD3D10ObjectsKHR](#), [107](#)

[clEnqueueReleaseD3D11ObjectsKHR](#), [123](#)

[clEnqueueReleaseDX9MediaSurfacesKHR](#), [135](#)

[clEnqueueReleaseEGLObjectsKHR](#), [161](#)

[clEnqueueReleaseGLObjects](#), [88](#)

[clGetDeviceIDsFromD3D10KHR](#), [100](#)

[clGetDeviceIDsFromD3D11KHR](#), [116](#)

[clGetDeviceIDsFromDX9MediaAdapterKHR](#), [130](#)

[clGetExtensionFunctionAddressForPlatform](#), [4](#)

[clGetGLContextInfoKHR](#), [75](#)

[clGetGLObjectInfo](#), [85](#)

[clGetGLTextureInfo](#), [86](#)

[clGetKernelSubGroupInfoKHR](#), [179](#)

[clIcdGetPlatformIDsKHR](#), [10](#)

[clTerminateContextKHR](#), [167](#)

# 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.

# 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
<a href="#">cl_khr_3d_image_writes</a>	Write to 3D images	Core Feature in OpenCL 2.0
<a href="#">cl_khr_byte_addressable_store</a>	Read and write from 8-bit and 16-bit pointers	Core Feature in OpenCL 1.1
<a href="#">cl_khr_create_command_queue</a>	API to Create Command Queues with Properties	Core Feature in OpenCL 2.0
<a href="#">cl_khr_d3d10_sharing</a>	Share Direct3D 10 Buffers and Textures with OpenCL	Extension
<a href="#">cl_khr_d3d11_sharing</a>	Share Direct3D 11 Buffers and Textures with OpenCL	Extension
<a href="#">cl_khr_depth_images</a>	Single Channel Depth Images	Core Feature in OpenCL 2.0
<a href="#">cl_khr_device_enqueue_local_arg_types</a>	Pass Non-Void Local Pointers to Child Kernels	Extension
<a href="#">cl_khr_dx9_media_sharing</a>	Share DirectX 9 Media Surfaces with OpenCL	Extension
<a href="#">cl_khr_egl_event</a>	Share EGL Sync Objects with OpenCL	Extension
<a href="#">cl_khr_egl_image</a>	Share EGL Images with OpenCL	Extension
<a href="#">cl_khr_fp16</a>	Operations on 16-bit Floating-Point Values	Extension
<a href="#">cl_khr_fp64</a>	Operations on 64-bit Floating-Point Values	Optional Core Feature in OpenCL 1.2
<a href="#">cl_khr_gl_depth_images</a>	Share OpenGL Depth Images with OpenCL	Extension
<a href="#">cl_khr_gl_event</a>	Share OpenGL Fence Sync Objects with OpenCL	Extension
<a href="#">cl_khr_gl_msaa_sharing</a>	Share OpenGL MSAA Textures with OpenCL	Extension
<a href="#">cl_khr_gl_sharing</a>	Sharing OpenGL Buffers and Textures with OpenCL	Extension
<a href="#">cl_khr_global_int32_base_atomics</a>	Basic Atomic Operations on 32-bit Integers in Global Memory	Core Feature in OpenCL 1.1
<a href="#">cl_khr_global_int32_extended_atomics</a>	Extended Atomic Operations on 32-bit Integers in Global Memory	Core Feature in OpenCL 1.1
<a href="#">cl_khr_icd</a>	Installable Client Drivers	Extension
<a href="#">cl_khr_il_program</a>	Support for Intermediate Language (IL) Programs (SPIR-V)	Core Feature in OpenCL 2.1

<b>Extension Name</b>	<b>Brief Description</b>	<b>Status</b>
<a href="#">cl_khr_image2d_from_buffer</a>	Create 2D Images from Buffers	Core Feature in OpenCL 2.0
<a href="#">cl_khr_initialize_memory</a>	Initialize Local and Private Memory on Allocation	Extension
<a href="#">cl_khr_int64_base_atomics</a>	Basic Atomic Operations on 64-bit Integers in Global and Local Memory	Extension
<a href="#">cl_khr_int64_extended_atomics</a>	Extended Atomic Operations on 64-bit Integers in Global and Local Memory	Extension
<a href="#">cl_khr_local_int32_base_atomics</a>	Basic Atomic Operations on 32-bit Integers in Local Memory	Extension
<a href="#">cl_khr_local_int32_extended_atomics</a>	Extended Atomic Operations on 32-bit Integers in Local Memory	Extension
<a href="#">cl_khr_mipmap_image</a>	Create and Use Images with Mipmaps	Extension
<a href="#">cl_khr_mipmap_image_writes</a>	Write to Images with Mipmaps	Extension
<a href="#">cl_khr_priority_hints</a>	Create Command Queues with Different Priorities	Extension
<a href="#">cl_khr_select_fprounding_mode</a>	Set the Current Kernel Rounding Mode	DEPRECATED
<a href="#">cl_khr_spir</a>	Standard Portable Intermediate Representation Programs	Extension, Superseded by IL Programs / SPIR-V
<a href="#">cl_khr_srgb_image_writes</a>	Write to sRGB Images	Extension
<a href="#">cl_khr_subgroup_named_barrier</a>	Barriers for Subsets of a Work Group	Extension
<a href="#">cl_khr_subgroups</a>	Sub-Groupings of Work Items	Core Feature in OpenCL 2.1
<a href="#">cl_khr_terminate_context</a>	Terminate an OpenCL Context	Extension
<a href="#">cl_khr_throttle_hints</a>	Create Command Queues with Different Throttle Policies	Extension