The **OpenCL Extension** Specification

*Version: 2.1*

*Document Revision: 17*

Khronos OpenCL Working Group

*Editor: Allen Hux*
9. OPTIONAL EXTENSIONS ................................................................. 6

9.1 Compiler Directives for Optional Extensions ..................................... 7

9.2 Getting OpenCL API Extension Function Pointers ................................ 9

9.3 Creating CL context from a GL context or share group ........................ 11
  9.3.1 Overview ................................................................. 11
  9.3.2 New Procedures and Functions ............................................. 11
  9.3.3 New Tokens ............................................................... 11
  9.3.4 Additions to Chapter 4 of the OpenCL 2.1 Specification ................. 12
  9.3.5 Additions to section 9.7 of the OpenCL 2.1 Extension Specification ... 14
  9.3.6 Issues ................................................................. 17

9.4 Sharing Memory Objects with OpenGL / OpenGL ES Buffer, Texture and Renderbuffer Objects ................................................................. 20
  9.4.1 Lifetime of Shared Objects ................................................. 20
  9.4.2 CL Buffer Objects → GL Buffer Objects ................................ 21
  9.4.3 CL Image Objects → GL Textures ...................................... 22
    9.4.3.1 List of OpenGL and corresponding OpenCL Image Formats .......... 24
  9.4.4 CL Image Objects → GL Renderbuffers ................................. 25
  9.4.5 Querying GL object information from a CL memory object ............ 27
  9.4.6 Sharing memory objects that map to GL objects between GL and CL contexts ................................................................. 29
    9.4.6.1 Synchronizing OpenCL and OpenGL Access to Shared Objects ... 31

9.5 Creating CL event objects from GL sync objects .................................. 33
  9.5.1 Overview ................................................................. 33
  9.5.2 New Procedures and Functions ............................................. 33
  9.5.3 New Tokens ............................................................... 33
  9.5.4 Additions to Chapter 5 of the OpenCL 2.1 Specification ................. 33
  9.5.5 Additions to Chapter 9 of the OpenCL 2.1 Specification ................. 35
  9.5.6 Issues ................................................................. 36

9.6 Sharing Memory Objects with Direct3D 10 ...................................... 38
  9.6.1 Overview ................................................................. 38
  9.6.2 Header File ............................................................... 38
  9.6.3 New Procedures and Functions ............................................. 38
  9.6.4 New Tokens ............................................................... 39
  9.6.5 Additions to Chapter 4 of the OpenCL 2.1 Specification ................. 40
  9.6.6 Additions to Chapter 5 of the OpenCL 2.1 Specification ................. 41
  9.6.7 Sharing Memory Objects with Direct3D 10 Resources .................. 42
    9.6.7.1 Querying OpenCL Devices Corresponding to Direct3D 10 Devices .... 42
    9.6.7.2 Lifetime of Shared Objects ............................................. 44
    9.6.7.3 Sharing Direct3D 10 Buffer Resources as OpenCL Buffer Objects ... 45
    9.6.7.4 Sharing Direct3D 10 Texture and Resources as OpenCL Image Objects ... 46
    9.6.7.5 Querying Direct3D properties of memory objects created from Direct3D 10 resources ...................................................... 46
    9.6.7.6 Sharing memory objects created from Direct3D 10 resources between Direct3D 10 and OpenCL contexts .................................. 49
  9.6.8 Issues ................................................................. 53

9.7 DX9 Media Surface Sharing .......................................................... 55
  9.7.1 Overview ................................................................. 55
  9.7.2 Header File ............................................................... 55
  9.7.3 New Procedures and Functions ............................................. 55

Last Revision Date: November 5, 2015
9.7.4 New Tokens ................................................................. 56
9.7.5 Additions to Chapter 4 of the OpenCL 2.1 Specification ................. 57
9.7.6 Additions to Chapter 5 of the OpenCL 2.1 Specification .................. 58
9.7.7 Sharing Media Surfaces with OpenCL ................................... 59
  9.7.7.1 Querying OpenCL Devices corresponding to Media Adapters .......... 59
  9.7.7.2 Creating Media Resources as OpenCL Image Objects .................. 61
  9.7.7.3 Querying Media Surface Properties of Memory Objects created from Media Surfaces 63
  9.7.7.4 Sharing Memory Objects created from Media Surfaces between a Media Adapter and OpenCL .......................... 63
  9.7.7.5 Surface formats for Media Surface Sharing .......................... 67

9.8 Sharing Memory Objects with Direct3D 11 .................................. 69
  9.8.1 Overview .................................................................. 69
  9.8.2 Header File ............................................................ 69
  9.8.3 New Procedures and Functions ........................................... 69
  9.8.4 New Tokens ............................................................ 70
  9.8.5 Additions to Chapter 4 of the OpenCL 2.1 Specification .................. 71
  9.8.6 Additions to Chapter 5 of the OpenCL 2.1 Specification .................. 72
  9.8.7 Sharing Memory Objects with Direct3D 11 Resources .................. 73
     9.8.7.1 Querying OpenCL Devices Corresponding to Direct3D 11 Devices .... 73
     9.8.7.2 Lifetime of Shared Objects ........................................... 75
     9.8.7.3 Sharing Direct3D 11 Buffer Resources as OpenCL Buffer Objects ........ 76
     9.8.7.4 Sharing Direct3D 11 Texture and Resources as OpenCL Image Objects .... 77
     9.8.7.5 Querying Direct3D properties of memory objects created from Direct3D 11 resources ...... 80
     9.8.7.6 Sharing memory objects created from Direct3D 11 resources between Direct3D 11 and OpenCL contexts ... 80

9.9 Sharing OpenGL and OpenGL ES Depth and Depth-Stencil Images ............. 85
  9.9.1 Additions to Chapter 5 of the OpenCL 2.1 Specification .................. 85
  9.9.2 Additions to Chapter 9.7 of the OpenCL 2.1 Extension Specification ........ 86

9.10 Sharing of CL / GL MSAA Textures ............................................. 87
  9.10.1 Additions to Chapter 9.7 of the OpenCL 2.1 Extension Specification ........ 87
  9.10.2 Additions to Chapter 5 of the OpenCL 2.1 Specification .................. 88

9.11 Local and Private Memory Initialization ........................................ 89
  9.11.1 Additions to Chapter 4 of the OpenCL 2.1 Specification .................. 89
  9.11.2 Additions to Chapter 6 of the OpenCL 2.1 Specification .................. 89

9.12 Terminating OpenCL contexts .................................................. 91
  9.12.1 Additions to Chapter 4 of the OpenCL 2.1 Specification .................. 91

9.13 SPIR 1.2 Binaries .............................................................. 94
  9.13.1 Additions to Chapter 4 of the OpenCL 2.1 Specification .................. 94
  9.13.2 Additions to Chapter 5 of the OpenCL 2.1 Specification .................. 94

9.14 OpenCL Installable Client Driver (ICD) ...................................... 96
  9.14.1 Overview .................................................................. 96
  9.14.2 Inferring Vendors from Function Call Arguments ........................ 96
  9.14.3 ICD Data .............................................................. 97
  9.14.4 ICD Loader Vendor Enumeration on Windows ............................ 97
  9.14.5 ICD Loader Vendor Enumeration on Linux ............................... 97
  9.14.6 ICD Loader Vendor Enumeration on Android ............................. 97
  9.14.7 Adding a Vendor Library .............................................. 98
  9.14.8 New Procedures and Functions ........................................... 98
  9.14.9 New Tokens ............................................................ 98
  9.14.10 Additions to Chapter 4 of the OpenCL 2.1 Specification .................. 99
  9.14.11 Additions to Chapter 9 of the OpenCL 2.1 Extension Specification ........ 100
9.15 Mipmaps .................................................................................................................. 102
  9.15.1 Additions to Chapter 5 of the OpenCL 2.1 Specification ........................................ 102
  9.15.1.1 Additions to section 5.3 – Image Objects .............................................................. 102
  9.15.1.2 Additions to section 5.7 – Sampler Objects ........................................................ 103
  9.15.2 Additions to section 9.7 – Sharing Memory Objects with OpenGL / OpenGL ES Texture Objects .......................................................... 103

9.16 Creating CL image objects from EGL images ............................................................ 105
  9.16.1 Overview ............................................................................................................. 105
  9.16.2 New Procedures and Functions ............................................................................ 105
  9.16.3 New Tokens ....................................................................................................... 105
  9.16.4 Additions to Chapter 5 of the OpenCL 2.1 Specification .......................................... 106
  9.16.5 Issues ................................................................................................................ 110

9.17 Creating CL event objects from EGL sync objects ....................................................... 113
  9.17.1 Overview ............................................................................................................. 113
  9.17.2 New Procedures and Functions ............................................................................ 113
  9.17.3 New Tokens ....................................................................................................... 113
  9.17.4 Additions to Chapter 5 of the OpenCL 2.1 Specification .......................................... 113
  9.17.5 Additions to Chapter 9 of the OpenCL 2.1 Specification .......................................... 115
  9.17.6 Issues ................................................................................................................ 116

9.18 Priority Hints ............................................................................................................ 117
  9.18.1 Host-side API modifications ................................................................................ 117

9.19 Throttle Hints ............................................................................................................ 118
  9.19.1 Host-side API modifications ................................................................................ 118

APPENDIX A – CHANGES ............................................................................................... 119
A.1 Summary of changes from OpenCL 2.0 ...................................................................... 119

INDEX - APIS .................................................................................................................... 120
This specification is protected by copyright laws and contains material proprietary to the Khronos Group, Inc. 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. You may use this specification for implementing the functionality therein, without altering or removing any trademark, copyright or other notice from the specification, but the receipt or possession of this specification does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part.

Khronos Group grants express permission to any current Promoter, Contributor or Adopter member of Khronos to copy and redistribute UNMODIFIED versions of this specification in any fashion, provided that NO CHARGE is made for the specification and the latest available update of the specification for any version of the API is used whenever possible. Such distributed specification may be re-formatted AS LONG AS the contents of the specification are not changed in any way. The specification may be incorporated into a product that is sold as long as such product includes significant independent work developed by the seller. A link to the current version of this specification on the Khronos Group web-site should be included whenever possible with specification distributions.

Khronos Group makes no, and expressly disclaims any, representations or warranties, express or implied, regarding this specification, including, without limitation, any implied warranties of merchantability or fitness for a particular purpose or non-infringement of any intellectual property. Khronos Group makes no, and expressly disclaims any, warranties, express or implied, regarding the correctness, accuracy, completeness, timeliness, and reliability of the specification. 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.

Khronos, StreamInput, WebGL, COLLADA, OpenKODE, OpenVG, OpenWF, OpenSL ES, OpenMAX, OpenMAX AL, OpenMAX IL and OpenMX DL are trademarks and WebCL is a certification mark of the Khronos Group Inc. 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.
9. Optional Extensions

This document describes the list of optional features supported by OpenCL 2.1. Optional extensions may be supported by some OpenCL devices. Optional extensions are not required to be supported by a conformant OpenCL implementation, but are expected to be widely available; they define functionality that is likely to move into the required feature set in a future revision of the OpenCL specification. A brief description of how OpenCL extensions are defined is provided below.

For OpenCL extensions approved by the OpenCL working group, the following naming conventions are used:

- 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 CL_PLATFORM_EXTENSIONS string defined in table 4.1 or CL_DEVICE_EXTENSIONS string described in table 4.3.

- All API functions defined by the extension will have names of the form cl<FunctionName>KHR.

- All enumerants defined by the extension will have names of the form CL_<enum_name>_KHR.

OpenCL extensions approved by the OpenCL working group can be promoted to required core features in later revisions of OpenCL. When this occurs, the extension specifications are merged into the core specification. Functions and enumerants that are part of such promoted extensions will have the 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.

For vendor extensions, the following naming conventions are used:

- 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 CL_PLATFORM_EXTENSIONS string described in table 4.1 or CL_DEVICE_EXTENSIONS string described in table 4.3.

- All API functions defined by the vendor extension will have names of the form cl<FunctionName><vendor_name>.

---

1 This document describes section 9 of the OpenCL 2.1 specification. Any reference to section 1.x – 8.x or tables 1.x – 8.x in this document refer to sections and tables described in the OpenCL 2.1 API and OpenCL C specifications.
All enumerants defined by the vendor extension will have names of the form 
CL_<enum_name>_<vendor_name>.

9.1 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.

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

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 \texttt{#define} that matches the extension name string. This \texttt{#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 "\texttt{cl_khr_3d_image_writes}" should also add a preprocessor \texttt{#define} called \texttt{cl_khr_3d_image_writes}. A kernel can now use this preprocessor \texttt{#define} to do something like:

\begin{verbatim}
  \ifdef cl_khr_3d_image_writes
  \hspace{1cm} // do something using the extension
  \else
  \hspace{1cm} // do something else or \texttt{#error}!
  \endif
\end{verbatim}
9.2 Getting OpenCL API Extension Function Pointers

The function

```c
void* clGetExtensionFunctionAddressForPlatform2(
    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.

`clGetExtensionFunctionAddressForPlatform` may not be queried for core (non-extension) functions in OpenCL. For functions that are queryable with `clGetExtensionFunctionAddressForPlatform`, implementations may choose to also 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:

```c
#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
```

2 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.
typedef CL_API_ENTRY return type
   (CL_API_CALL *clextenstion_func_nameTAG_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
9.3 Creating CL context from a GL context or share group

9.3.1 Overview

This section describes the `cl_khr_gl_sharing` extension. The OpenCL specification in section 9.7 defines how to share data with texture and buffer objects in a parallel OpenGL implementation, but does not define how the association between an OpenCL context and an OpenGL context or share group is established. This extension defines optional attributes to OpenCL context creation routines which associate a GL context or share group object with a newly created OpenCL context.

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

9.3.2 New Procedures and Functions

```c
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);
```

9.3.3 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 -1000
```

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

```
CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR 0x2006
CL_DEVICES_FOR_GL_CONTEXT_KHR 0x2007
```

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

```
CL_GL_CONTEXT_KHR 0x2008
CL_EGL_DISPLAY_KHR 0x2009
```
9.3.4 Additions to Chapter 4 of the OpenCL 2.1 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 as described in section 9.7. 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 in section 9.7 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>
<thead>
<tr>
<th>Attribute Name</th>
<th>Allowed Values (Default value is in bold)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_GL_CONTEXT_KHR</td>
<td>0, OpenGL context handle</td>
<td>OpenGL context to associated the OpenCL context with</td>
</tr>
<tr>
<td>CL_CGL_SHAREGROUP_KHR</td>
<td>0, CGL share group handle</td>
<td>CGL share group to associate the OpenCL context with</td>
</tr>
<tr>
<td>CL_EGL_DISPLAY_KHR</td>
<td>EGL_NO_DISPLAY, EGLDisplay handle</td>
<td>EGLDisplay an OpenGL context was created with respect to</td>
</tr>
<tr>
<td>CL_GLX_DISPLAY_KHR</td>
<td>None, X handle</td>
<td>X Display an OpenGL context was created with respect to</td>
</tr>
<tr>
<td>CL_WGL_HDC_KHR</td>
<td>0, HDC handle</td>
<td>HDC an OpenGL context was created with respect to</td>
</tr>
</tbody>
</table>

**Table 4.5: Context creation attributes**

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, as described in section 9.7.

`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`.

### 9.3.5 Additions to section 9.7 of the OpenCL 2.1 Extension Specification

Add new section 9.7.7:

"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

```c
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 GL context information to query, and must be one of the values shown in `table 9.ctxprop`.

`param_value` is a pointer to memory where the result of the query is returned as described in `table 9.ctxprop`. 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 `table 9.ctxprop`.

`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>
<thead>
<tr>
<th><code>param_name</code></th>
<th><code>Return Type</code></th>
<th>Information returned in <code>param_value</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR</td>
<td>cl_device_id</td>
<td>Return the CL device currently associated with the specified OpenGL context.</td>
</tr>
<tr>
<td>CL_DEVICES_FOR_GL_CONTEXT_KHR</td>
<td>cl_device_id[]</td>
<td>List of all CL devices which may be associated with the specified OpenGL context.</td>
</tr>
</tbody>
</table>

**Table 9. ctxprop**: GL context information that can be queried with `clGetGLContextInfoKHR`

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

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

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

A context was specified for a WGL-based OpenGL implementation by setting the attributes CL_GL_CONTEXT_KHR and CL_WGL_HDC_KHR.

and any of the following conditions hold:

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

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

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

- A context or share group object was specified for one of CGL, EGL, GLX, or WGL and the OpenGL implementation does not support that window-system binding API.
- More than one of the attributes CL_CGL_SHAREGROUP_KHR, CL_EGL_DISPLAY_KHR, CL_GLX_DISPLAY_KHR, and CL_WGL_HDC_KHR is set to a non-default value.
- Both of the attributes CL_CGL_SHAREGROUP_KHR and CL_GL_CONTEXT_KHR are set to non-default values.
- Any of the devices specified in the <devices> argument cannot support OpenCL objects which share the data store of an OpenGL object, as described in section 9.7.

`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 `table 9.ctxprop`, or if the size in bytes specified by `param_value_size` is less than the size of the return type shown in `table 9.ctxprop`, and `param_value` is not a NULL value, CL_OUT_OF_RESOURCES if there is a failure to allocate resources required by the OpenCL implementation on the device, or CL_OUT_OF_HOST_MEMORY if there is a failure to allocate resources required by the OpenCL implementation on the host."
9.3.6 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: As described in section 9.7, 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 required by the section 9.7 interfaces, 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 \texttt{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 \texttt{CL_INVALID_OPERATION} error.

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

RESOLVED: The \texttt{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 section 9.7 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 \texttt{CL_DEVICES_FOR_GL_CONTEXT_KHR} query?

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

9) Miscellaneous issues during syncing of version 12 with the OpenCL 1.0 revision 47 spec language and the minor changes made including this extension as section 9.11 of that spec:

- Rev47 spec numbers table 9.ctxprop as "9.7" but this depends on the core spec revision.
- Rev47 spec uses 'cl_context' as the return type for \texttt{clGetGLContextInfoKHR} param names, but cl_device_id / cl_device_id[] are the proper types.
Rev47 spec omits the paragraph describing CL_SUCCESS return from 
clGetGLContextInfoKHR.
9.4 Sharing Memory Objects with OpenGL / OpenGL ES Buffer, Texture and Renderbuffer Objects

This section discusses OpenCL functions that allow applications to use OpenGL buffer, texture and renderbuffer objects as OpenCL memory objects. This allows efficient sharing of data between OpenCL and OpenGL. The OpenCL API may be used to execute kernels that read and/or write memory objects that are also OpenGL objects.

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.

OpenCL memory objects may be created from OpenGL objects if and only if the OpenCL context has been created from an OpenGL share group object or context. OpenGL share groups and contexts are created using platform specific APIs such as EGL, CGL, WGL, and GLX. On MacOS X, an OpenCL context may be created from an OpenGL share group object using the OpenCL platform extension cl_apple_gl_sharing. On other platforms including Microsoft Windows, Linux/Unix and others, an OpenCL context may be created from an OpenGL context using the Khronos platform extension cl_khr_gl_sharing. Refer to the platform documentation for your OpenCL implementation, or visit the Khronos Registry at http://www.khronos.org/registry/cl/ for more information.

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

9.4.1 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.
9.4.2 CL Buffer Objects → GL Buffer Objects

The function

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

creates an OpenCL buffer object from an OpenGL buffer object.

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

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

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

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

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

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

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

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

### 9.4.3 CL Image Objects → GL Textures

The function

```c
cl_mem clCreateFromGLTexture(cl_context context,
                               cl_mem_flags flags,
                               GLenum texture_target,
                               GLint mipmaplevel,
                               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\textsuperscript{3}. \textit{texture\_target} is used only to define the image type of \textit{texture}. No reference to a bound GL texture object is made or implied by this parameter.

\textit{miplevel} is the mipmap level to be used\textsuperscript{4}. If \textit{texture\_target} is GL_TEXTURE_BUFFER, \textit{miplevel} must be 0.

\textit{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 \textit{texture} format and dimensions defined by OpenGL for the specified \textit{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.

\textit{errcode\_ret} will return an appropriate error code as described below. If \textit{errcode\_ret} is NULL, no error code is returned.

\textbf{clCreateFromGLTexture} returns a valid non-zero OpenCL image object and \textit{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 \textit{errcode\_ret}:

\begin{itemize}
  \item CL\_INVALID\_CONTEXT if \textit{context} is not a valid context or was not created from a GL context.
  \item CL\_INVALID\_VALUE if values specified in \textit{flags} are not valid or if value specified in \textit{texture\_target} is not one of the values specified in the description of \textit{texture\_target}.
  \item CL\_INVALID\_MIP\_LEVEL if \textit{miplevel} is less than the value of \textit{level\_base} (for OpenGL implementations) or zero (for OpenGL ES implementations); or greater than the value of \textit{q} (for both OpenGL and OpenGL ES). \textit{level\_base} and \textit{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.
  \item CL\_INVALID\_MIP\_LEVEL if \textit{miplevel} is greater than zero and the OpenGL implementation does not support creating from non-zero mipmap levels.
  \item CL\_INVALID\_GL\_OBJECT if \textit{texture} is not a GL texture object whose type matches \textit{texture\_target}, if the specified \textit{miplevel} of \textit{texture} is not defined, or if the width or height of the specified \textit{miplevel} is zero or if the GL texture object is incomplete.
  \item CL\_INVALID\_IMAGE\_FORMAT\__DESCRIPTOR if the OpenGL texture internal format does not map to a supported OpenCL image format.
\end{itemize}

\textsuperscript{3} Requires OpenGL 3.1. Alternatively, GL\_TEXTURE\_RECTANGLE\_ARB may be specified if the OpenGL extension GL\_ARB\_texture\_rectangle is supported.

\textsuperscript{4} Implementations may return CL\_INVALID\_OPERATION for \textit{miplevel} values $> 0$. 
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.

### 9.4.3.1 List of OpenGL and corresponding OpenCL Image Formats

*Table 9.4* describes the list of GL texture internal formats and the corresponding CL image formats. If a GL texture object with an internal format from *table 9.4* is successfully created by OpenGL, then there is guaranteed to be a mapping to one of the corresponding CL image format(s) in that table. Texture objects created with other OpenGL internal formats may (but are not guaranteed to) have a mapping to a CL 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>
<thead>
<tr>
<th>GL internal format</th>
<th>CL image format (channel order, channel data type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_RGBA8</td>
<td>CL_RGBA, CL_UNORM_INT8 or CL_BGRA, CL_UNORM_INT8</td>
</tr>
<tr>
<td>GL_SRGB8_ALPHA8</td>
<td>CL_sRGBA, CL_UNORM_INT8</td>
</tr>
<tr>
<td>GL_RGBA, GL_UNSIGNED_INT_8_8_8_8_REV</td>
<td>CL_RGBA, CL_UNORM_INT8</td>
</tr>
<tr>
<td>GL_BGRA, GL_UNSIGNED_INT_8_8_8_REV</td>
<td>CL_BGRA, CL_UNORM_INT8</td>
</tr>
<tr>
<td>GL_RGBA8I, GL_RGBA8I_EXT</td>
<td>CL_RGBA, CL_SIGNED_INT8</td>
</tr>
<tr>
<td>GL_RGBA16I, GL_RGBA16I_EXT</td>
<td>CL_RGBA, CL_SIGNED_INT16</td>
</tr>
<tr>
<td>GL_RGBA32I, GL_RGBA32I_EXT</td>
<td>CL_RGBA, CL_SIGNED_INT32</td>
</tr>
<tr>
<td>GL_RGBA8UI, GL_RGBA8UI_EXT</td>
<td>CL_RGBA, CL_UNSIGNED_INT8</td>
</tr>
<tr>
<td>GL_RGBA16UI, GL_RGBA16UI_EXT</td>
<td>CL_RGBA, CL_UNSIGNED_INT16</td>
</tr>
<tr>
<td>GL_RGBA32UI, GL_RGBA32UI_EXT</td>
<td>CL_RGBA, CL_UNSIGNED_INT32</td>
</tr>
</tbody>
</table>
Table 9.4  Mapping of GL internal format to CL image format

<table>
<thead>
<tr>
<th>GL_RGBA8_SNORM</th>
<th>CL_RGBA, CL_SNORM_INT8</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_RGBA16</td>
<td>CL_RGBA, CL_UNORM_INT16</td>
</tr>
<tr>
<td>GL_RGBA16_SNORM</td>
<td>CL_RGBA, CL_SNORM_INT16</td>
</tr>
<tr>
<td>GL_RGBA16F, GL_RGBA16F_ARB</td>
<td>CL_RGBA, CL_HALF_FLOAT</td>
</tr>
<tr>
<td>GL_RGBA32F, GL_RGBA32F_ARB</td>
<td>CL_RGBA, CL_FLOAT</td>
</tr>
<tr>
<td>GL_R8</td>
<td>CL_R, CL_UNORM_INT8</td>
</tr>
<tr>
<td>GL_R8_SNORM</td>
<td>CL_R, CL_SNORM_INT8</td>
</tr>
<tr>
<td>GL_R16</td>
<td>CL_R, CL_UNORM_INT16</td>
</tr>
<tr>
<td>GL_R16_SNORM</td>
<td>CL_R, CL_SNORM_INT16</td>
</tr>
<tr>
<td>GL_R16F</td>
<td>CL_R, CL_HALF_FLOAT</td>
</tr>
<tr>
<td>GL_R32F</td>
<td>CL_R, CL_FLOAT</td>
</tr>
<tr>
<td>GL_R8I</td>
<td>CL_R, CL_SIGNED_INT8</td>
</tr>
<tr>
<td>GL_R16I</td>
<td>CL_R, CL_SIGNED_INT16</td>
</tr>
<tr>
<td>GL_R32I</td>
<td>CL_R, CL_SIGNED_INT32</td>
</tr>
<tr>
<td>GL_R8UI</td>
<td>CL_R, CL_UNSIGNED_INT8</td>
</tr>
<tr>
<td>GL_R16UI</td>
<td>CL_R, CL_UNSIGNED_INT16</td>
</tr>
<tr>
<td>GL_R32UI</td>
<td>CL_R, CL_UNSIGNED_INT32</td>
</tr>
<tr>
<td>GL_RG8</td>
<td>CL_RG, CL_UNORM_INT8</td>
</tr>
<tr>
<td>GL_RG8_SNORM</td>
<td>CL_RG, CL_SNORM_INT8</td>
</tr>
<tr>
<td>GL_RG16</td>
<td>CL_RG, CL_UNORM_INT16</td>
</tr>
<tr>
<td>GL_RG16_SNORM</td>
<td>CL_RG, CL_SNORM_INT16</td>
</tr>
<tr>
<td>GL_RG16F</td>
<td>CL_RG, CL_HALF_FLOAT</td>
</tr>
<tr>
<td>GL_RG32F</td>
<td>CL_RG, CL_FLOAT</td>
</tr>
<tr>
<td>GL_RG8I</td>
<td>CL_RG, CL_SIGNED_INT8</td>
</tr>
<tr>
<td>GL_RG16I</td>
<td>CL_RG, CL_SIGNED_INT16</td>
</tr>
<tr>
<td>GL_RG32I</td>
<td>CL_RG, CL_SIGNED_INT32</td>
</tr>
<tr>
<td>GL_RG8UI</td>
<td>CL_RG, CL_UNSIGNED_INT8</td>
</tr>
<tr>
<td>GL_RG16UI</td>
<td>CL_RG, CL_UNSIGNED_INT16</td>
</tr>
<tr>
<td>GL_RG32UI</td>
<td>CL_RG, CL_UNSIGNED_INT32</td>
</tr>
</tbody>
</table>

9.4.4  CL Image Objects → GL 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.
Table 9.4 describes the list of GL renderbuffer internal formats and the corresponding CL image formats. If a GL renderbuffer object with an internal format from table 9.4 is successfully created by OpenGL, then there is guaranteed to be a mapping to one of the corresponding CL image format(s) in that table. Renderbuffer objects created with other OpenGL internal formats may (but are not guaranteed to) have a mapping to a CL 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.

9.4.5 Querying GL object information from a CL 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.

```c
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
- 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 **table 9.5** below.

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

*param_value_size* is used to specify the size in bytes of memory pointed to by *param_value*. This size must be >= size of return type as described in **table 9.5** 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>
<thead>
<tr>
<th>cl_gl_texture_info</th>
<th>Return Type</th>
<th>Info. returned in <em>param_value</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_GL_TEXTURE_TARGET</td>
<td>GLenum</td>
<td>The <em>texture_target</em> argument specified in <strong>clCreateFromGLTexture</strong>.</td>
</tr>
<tr>
<td>CL_GL_MIPMAP_LEVEL</td>
<td>GLint</td>
<td>The <em>mipmap</em> argument specified in <strong>clCreateFromGLTexture</strong>.</td>
</tr>
</tbody>
</table>

**Table 9.5**  List of supported *param_names* by **clGetGLTextureInfo**

**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 < size of return type as described in **table 9.5** 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.

9.4.6 Sharing memory objects that map to GL objects between GL and CL contexts

The function

```c
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

```c
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 read / write 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.

9.4.6.1 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.


9.5 Creating CL event objects from GL sync objects

9.5.1 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.

9.5.2 New Procedures and Functions

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

9.5.3 New Tokens

Returned by clGetEventInfo when param_name is CL_EVENT_COMMAND_TYPE:

| CL_COMMAND_GL_FENCE_SYNC_OBJECT_KHR | 0x200D |

9.5.4 Additions to Chapter 5 of the OpenCL 2.1 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 (see section 9.7.6.1)."

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

\[
\text{cl\_event clCreateEventFromGLsyncKHR (cl\_context context, GLsync sync, cl\_int *errcode\_ret)}
\]

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.

9.5.5 Additions to Chapter 9 of the OpenCL 2.1 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 section 9.7.6.1 (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.

### 9.5.6 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.
9.6 Sharing Memory Objects with Direct3D 10

9.6.1 Overview

This section describes the cl_khr_d3d10_sharing extension. The goal of this extension is to provide interoperability between OpenCL and Direct3D 10. This is designed to function analogously to the OpenGL interoperability as defined in sections 9.7 and 9.8.

9.6.2 Header File

As currently proposed the interfaces for this extension would be provided in cl_d3d10.h.

9.6.3 New Procedures and Functions

```c
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)

### 9.6.4 New Tokens

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

- **CL_D3D10_DEVICE_KHR** 0x4010
- **CL_D3D10_DXGI_ADAPTER_KHR** 0x4011

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

- **CL_PREFERRED_DEVICES_FOR_D3D10_KHR** 0x4012
- **CL_ALL_DEVICES_FOR_D3D10_KHR** 0x4013

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

- **CL_CONTEXT_D3D10_DEVICE_KHR** 0x4014

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

- **CL_CONTEXT_D3D10_PREFER_SHARED_RESOURCES_KHR** 0x402C

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

- **CL_MEM_D3D10_RESOURCE_KHR** 0x4015

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

- **CL_IMAGE_D3D10_SUBRESOURCE_KHR** 0x4016
Returned in the *param_value* parameter of *clGetEventInfo* when *param_name* is CL_EVENT_COMMAND_TYPE:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_COMMAND_ACQUIRE_D3D10_OBJECTS_KHR</td>
<td>0x4017</td>
</tr>
<tr>
<td>CL_COMMAND_RELEASE_D3D10_OBJECTS_KHR</td>
<td>0x4018</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_INVALID_D3D10_DEVICE_KHR</td>
<td>-1002</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_INVALID_D3D10_RESOURCE_KHR</td>
<td>-1003</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_D3D10_Resource_ALREADY_ACQUIRED_KHR</td>
<td>-1004</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_D3D10_RESOURCE_NOT_ACQUIRED_KHR</td>
<td>-1005</td>
</tr>
</tbody>
</table>

### 9.6.5 Additions to Chapter 4 of the OpenCL 2.1 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*:

<table>
<thead>
<tr>
<th>cl_context_properties enum</th>
<th>Property value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_CONTEXT_D3D10_DEVICE_KHR</td>
<td>ID3D10Device *</td>
<td>Specifies the ID3D10Device * to use for Direct3D 10 interoperability.</td>
</tr>
</tbody>
</table>
The default value is NULL.

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

- **CL_INVALID_D3D10_DEVICE_KHR** if the value of the property
  **CL_CONTEXT_D3D10_DEVICE_KHR** is non-NULL and does not specify a valid
  Direct3D 10 device with which the *cl_device_ids* against which this context
  is to be created may interoperate.

- **CL_INVALID_OPERATION** if Direct3D 10 interoperability is specified by setting
  **CL_INVALID_D3D10_DEVICE_KHR** to a non-NULL value, and interoperability with
  another graphics API is also specified.”

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

Add the following row to **table 4.6**:

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

### 9.6.6 Additions to Chapter 5 of the OpenCL 2.1 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 5.13** to include the following entry.

<table>
<thead>
<tr>
<th><strong>cl_mem_info</strong></th>
<th><strong>Return type</strong></th>
<th><strong>Info. returned in param_value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>cl_image_info</th>
<th>Return type</th>
<th>Info. returned in <code>param_value</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_MEM_D3D10_SUBRESOURCE_KHR</td>
<td>ID3D10Resource *</td>
<td>If <code>image</code> was created using <code>clCreateFromD3D10Texture2DKHR</code>, or <code>clCreateFromD3D10Texture3DKHR</code>, returns the <code>subresource</code> argument specified when <code>image</code> was created.</td>
</tr>
</tbody>
</table>

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`

### 9.6.7 Sharing Memory Objects with Direct3D 10 Resources

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

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

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

```c
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 table 9.9.1.

*d3d_object* specifies the object whose corresponding OpenCL devices are being queried. The type of *d3d_object* must be as specified in table 9.9.1.

*d3d_device_set* specifies the set of devices to return, and must be one of the values shown in table 9.9.2.

*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 \textit{d3d\_object} were found.

<table>
<thead>
<tr>
<th>\textbf{cl_d3d_device_source_khr}</th>
<th>\textbf{Type of \textit{d3d_object}}</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_D3D10_DEVICE_KHR</td>
<td>ID3D10Device *</td>
</tr>
<tr>
<td>CL_D3D10_DXGI_ADAPTER_KHR</td>
<td>IDXGIApapter *</td>
</tr>
</tbody>
</table>

\textbf{Table 9.9.1} Types used to specify the object whose corresponding OpenCL devices are being queried by \textit{clGetDeviceIDsFromD3D10KHR}

<table>
<thead>
<tr>
<th>\textbf{cl_d3d_device_set_khr}</th>
<th>\textbf{Devices returned in devices}</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_PREFERRED_DEVICES_FOR_D3D10_KHR</td>
<td>The OpenCL devices associated with the specified Direct3D object.</td>
</tr>
<tr>
<td>CL_ALL_DEVICES_FOR_D3D10_KHR</td>
<td>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.</td>
</tr>
</tbody>
</table>

\textbf{Table 9.9.2} Sets of devices queriable using \textit{clGetDeviceIDsFromD3D10KHR}

\textbf{9.6.7.2 Lifetime of Shared Objects}

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

The successful creation of a \textit{cl\_context} against a Direct3D 10 device specified via the context create parameter \textbf{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.
9.6.7.3 Sharing Direct3D 10 Buffer Resources as OpenCL Buffer Objects

The function

```c
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.
9.6.7.4 Sharing Direct3D 10 Texture and Resources as OpenCL Image Objects

The function

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

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

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

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

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

*subresource* is the subresource of *resource* to share.

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

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

- CL_INVALID_CONTEXT if *context* is not a valid context.
- CL_INVALID_VALUE if values specified in *flags* are not valid or if *subresource* is not a valid subresource index for *resource*.
- CL_INVALID_D3D10_RESOURCE_KHR if *resource* is not a Direct3D 10 texture resource, if *resource* was created with the D3D10_USAGE flag D3D10_USAGE_IMMUTABLE, if *resource* is a multisampled texture, if a cl_mem from subresource *subresource* of *resource* has already been created using *clCreateFromD3D10Texture2DKHR*, or if *context* was not created against the same Direct3D 10 device from which *resource* was created.
- CL_INVALID_IMAGE_FORMAT_DESCRIPTOR if the Direct3D 10 texture format of *resource* is not listed in table 9.3 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 table 9.9.3.

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.

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

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

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

subresource is the subresource of resource to share.

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

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

- CL_INVALID_CONTEXT if context is not a valid context.
- CL_INVALID_VALUE if values specified in flags are not valid or if subresource is not a valid subresource index for resource.
- CL_INVALID_D3D10_RESOURCE_KHR if resource is not a Direct3D 10 texture resource, if resource was created with the D3D10_USAGE flag
D3D10_USAGE_IMMUTABLE, if resource is a multisampled texture, if a cl_mem from subresource subresource of resource has already been created using clCreateFromD3D10Texture3DKHR, or if context was not created against the same Direct3D 10 device from which resource was created.

- CL_INVALID_IMAGE_FORMAT_DESCRIPTOR if the Direct3D 10 texture format of resource is not listed in table 9.9.3 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 table 9.9.3.

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>
<thead>
<tr>
<th>DXGI format</th>
<th>CL image format (channel order, channel data type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DXGI_FORMAT_R32G32B32A32_FLOAT</td>
<td>CL_RGBA, CL_FLOAT</td>
</tr>
<tr>
<td>DXGI_FORMAT_R32G32B32A32_UINT</td>
<td>CL_RGBA, CL_UNSIGNED_INT32</td>
</tr>
<tr>
<td>DXGI_FORMAT_R32G32B32A32_SINT</td>
<td>CL_RGBA, CL_SIGNED_INT32</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16B16A16_FLOAT</td>
<td>CL_RGBA, CL_HALF_FLOAT</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16B16A16_UNORM</td>
<td>CL_RGBA, CL_UNORM_INT16</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16B16A16_SNORM</td>
<td>CL_RGBA, CL_SIGNED_INT16</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16B16A16_SINT</td>
<td>CL_RGBA, CL_SIGNED_INT16</td>
</tr>
<tr>
<td>DXGI_FORMAT_B8G8R8A8_UNORM</td>
<td>CL_BGRA, CL_UNORM_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8B8A8_UNORM</td>
<td>CL_RGBA, CL_UNORM_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8B8A8_UINT</td>
<td>CL_RGBA, CL_UNSIGNED_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8B8A8_SNORM</td>
<td>CL_RGBA, CL_SIGNED_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8B8A8_SINT</td>
<td>CL_RGBA, CL_SIGNED_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R32G32_FLOAT</td>
<td>CL_RG, CL_FLOAT</td>
</tr>
<tr>
<td>DXGI_FORMAT_R32G32_UINT</td>
<td>CL_RG, CL_UNSIGNED_INT32</td>
</tr>
<tr>
<td>DXGI_FORMAT_R32G32_SINT</td>
<td>CL_RG, CL_SIGNED_INT32</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16_FLOAT</td>
<td>CL_RG, CL_HALF_FLOAT</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16_UNORM</td>
<td>CL_RG, CL_UNORM_INT16</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16_UINT</td>
<td>CL_RG, CL_UNSIGNED_INT16</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16_SNORM</td>
<td>CL_RG, CL_SIGNED_INT16</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16_SINT</td>
<td>CL_RG, CL_SIGNED_INT16</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8_UNORM</td>
<td>CL_RG, CL_UNORM_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8_UINT</td>
<td>CL_RG, CL_UNSIGNED_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8_SNORM</td>
<td>CL_RG, CL_SNORM_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8_SINT</td>
<td>CL_RG, CL_SIGNED_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R32_FLOAT</td>
<td>CL_R, CL_HALF_FLOAT</td>
</tr>
<tr>
<td>DXGI_FORMAT_R32_UINT</td>
<td>CL_R, CL_UNSIGNED_INT32</td>
</tr>
<tr>
<td>DXGI_FORMAT_R32_SINT</td>
<td>CL_R, CL_SIGNED_INT32</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16_FLOAT</td>
<td>CL_R, CL_HALF_FLOAT</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16_UINT</td>
<td>CL_R, CL_UNSIGNED_INT16</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16_SNORM</td>
<td>CL_R, CL_SNORM_INT16</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16_SINT</td>
<td>CL_R, CL_SIGNED_INT16</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8_UNORM</td>
<td>CL_R, CL_UNORM_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8_UINT</td>
<td>CL_R, CL_UNSIGNED_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8_SNORM</td>
<td>CL_R, CL_SNORM_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8_SINT</td>
<td>CL_R, CL_SIGNED_INT8</td>
</tr>
</tbody>
</table>

Table 9.9.3  List of Direct3D 10 and corresponding OpenCL image formats

9.6.7.5 Querying Direct3D properties of memory objects created from Direct3D 10 resources

Properties of Direct3D 10 objects may be queried using `clGetMemObjectInfo` and `clGetImageInfo` with `param_name` `CL_MEM_D3D10_RESOURCE_KHR` and `CL_IMAGE_D3D10 SUBRESOURCE_KHR` respectively as described in sections 5.4.3 and 5.3.6.

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

The function

```c
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.

cache_index 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.

device 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. device 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

```c
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 events specified in `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.

## 9.6.8 Issues

1) Should this extension be KHR or EXT?

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

RESOLVED: KHR.

2) Requiring SharedHandle on ID3D10Resource

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

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

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

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

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

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

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

3) Texture1D support

There is not a matching CL type, so do we want to support this and map to buffer or Texture2D? If so the command might correspond to the 2D / 3D versions:

```
cl_mem clCreateFromD3D10Texture1D (cl_context context,
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.
9.7 DX9 Media Surface Sharing

9.7.1 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.

9.7.2 Header File

As currently proposed the interfaces for this extension would be provided in cl_dx9_media_sharing.h.

9.7.3 New Procedures and Functions

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

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

cl_int  clEnqueueAcquireDX9MediaSurfacesKHR (cl_command_queue command_queue,
                                             cl_uint num_surfaces,
                                             cl_dx9_media_surface_type_khr *media_surfaces,
                                             cl_uint num_requests,
                                             cl_event *events,
                                             cl_uint flags)
```

Last Revision Date: November 5, 2015
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)

9.7.4 New Tokens

Accepted by the media_adapter_type parameter of clGetDeviceIDsFromDX9MediaAdapterKHR:

CL_ADAPTER_D3D9_KHR 0x2020
CL_ADAPTER_D3D9EX_KHR 0x2021
CL_ADAPTER_DXVA_KHR 0x2022

Accepted by the media_adapter_set parameter of clGetDeviceIDsFromDX9MediaAdapterKHR:

CL_PREFERRED_DEVICES_FOR_DX9_MEDIA_ADAPTER_KHR 0x2023
CL_ALL_DEVICES_FOR_DX9_MEDIA_ADAPTER_KHR 0x2024

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

CL_CONTEXT_ADAPTER_D3D9_KHR 0x2025
CL_CONTEXT_ADAPTER_D3D9EX_KHR 0x2026
CL_CONTEXT_ADAPTER_DXVA_KHR 0x2027

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

CL_MEM_DX9_MEDIA_ADAPTER_TYPE_KHR 0x2028
CL_MEM_DX9_MEDIA_SURFACE_INFO_KHR 0x2029

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

CL_IMAGE_DX9_MEDIA_PLANE_KHR 0x202A
Returned in the `param_value` parameter of `clGetEventInfo` when `param_name` is `CL_EVENT_COMMAND_TYPE`:

- `CL_COMMAND_ACQUIRE_DX9_MEDIA_SURFACES_KHR` 0x202B
- `CL_COMMAND_RELEASE_DX9_MEDIA_SURFACES_KHR` 0x202C

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` -1010

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` -1011

Returned by `clEnqueueAcquireDX9MediaSurfacesKHR` when any of `mem_objects` are currently acquired by OpenCL

- `CL_DX9_MEDIA_SURFACE_ALREADY_ACQUIRED_KHR` -1012

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

- `CL_DX9_MEDIA_SURFACE_NOT_ACQUIRED_KHR` -1013

### 9.7.5 Additions to Chapter 4 of the OpenCL 2.1 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:

<table>
<thead>
<tr>
<th>cl_context_properties enum</th>
<th>Property value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 9.7.6 Additions to Chapter 5 of the OpenCL 2.1 Specification

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**.

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 5.13 to include the following entry.

<table>
<thead>
<tr>
<th>cl_mem_info</th>
<th>Return type</th>
<th>Info. returned in param_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_MEM_DX9_MEDIA_ADAPTER_TYPE_KHR</td>
<td>cl_dx9_media_adapter_type_khr</td>
<td>Returns the <code>cl_dx9_media_adapter_type_khr</code> argument value specified when <code>memobj</code> is created using <strong>clCreateFromDX9MediaSurfaceKHR</strong>.</td>
</tr>
<tr>
<td>CL_MEM_DX9_MEDIA_SURFACE_INFO_KHR</td>
<td>cl_dx9_surface_info_khr</td>
<td>Returns the <code>cl_dx9_surface_info_khr</code> argument value specified when <code>memobj</code> is created using <strong>clCreateFromDX9MediaSurfaceKHR</strong>.</td>
</tr>
</tbody>
</table>

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 
cCreateFromDX9MediaSurfaceKHR.

Extend table 5.9 to include the following entry.

<table>
<thead>
<tr>
<th>cl_image_info</th>
<th>Return type</th>
<th>Info. returned in param_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_IMAGE_DX9_MEDIA_PLANE_KHR</td>
<td>cl_uint</td>
<td>Returns the plane argument value specified when memobj is created using clCreateFromDX9MediaSurfaceKHR.</td>
</tr>
</tbody>
</table>

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

### 9.7.7 Sharing Media Surfaces with OpenCL

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

#### 9.7.7.1 Querying OpenCL Devices corresponding to Media Adapters

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

The function

```c
cl_int  cGetDeviceIDsFromDX9MediaAdapterKHR (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 table 9.10.1.

<table>
<thead>
<tr>
<th>cl_dx9_media_adapter_type_khr</th>
<th>Type of media adapters</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_ADAPTER_D3D9_KHR</td>
<td>IDirect3DDevice9 *</td>
</tr>
<tr>
<td>CL_ADAPTER_D3D9EX_KHR</td>
<td>IDirect3DDevice9Ex *</td>
</tr>
<tr>
<td>CL_ADAPTER_DXVA_KHR</td>
<td>IDXVAHD_Device *</td>
</tr>
</tbody>
</table>

Table 9.10.1 List of cl_dx9_media_adapter_type_khr values

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 describes in table 9.10.1.

media_adapter_set specifies the set of adapters to return and must be one of the values described in table 9.10.2.

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.

cGetDeviceIDsFromDX9MediaAdapterKHR 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.

### 9.7.7.2 Creating Media Resources as OpenCL Image Objects

The function

```c
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 table 9.10.1. 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:

\[
\text{typedef struct \_cl\_dx9\_surface\_info\_khr}
\{
  IDirect3DSurface9 *resource;
  HANDLE shared_handle;
\} cl\_dx9\_surface\_info\_khr;
\]

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 tables 9.10.3 and 9.10.4.
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 tables 9.10.3 and 9.10.4.

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.

9.7.7.3 Querying Media Surface Properties of Memory Objects created from Media Surfaces

Properties of media surface objects may be queried using clGetMemObjectInfo and clGetImageInfo with param_name CL_MEM_DX9_MEDIA_ADAPTER_TYPE_KHR, CL_MEM_DX9_MEDIA_SURFACE_INFO_KHR and CL_IMAGE_DX9_MEDIA_PLANE_KHR as described in sections 5.4.3 and 5.3.6.

9.7.7.4 Sharing Memory Objects created from Media Surfaces between a Media Adapter and OpenCL

The function

```
cl_int clEnqueueAcquireDX9MediaSurfacesKHR (  
    cl_command_queue command_queue,  
    cl_uint num_objects,  
    const cl_mem *mem_objects,  
    cl_uint num_events_in_wait_list,  
    const cl_event *event_wait_list,  
    cl_event *event)  
```

is used to acquire OpenCL memory objects that have been created from a media surface. The media surfaces are acquired by the OpenCL context associated with command_queue and can therefore be used by all command-queues associated with the OpenCL context.

OpenCL memory objects created from media surfaces must be acquired before they can be used by any OpenCL commands queued to a command-queue. If an OpenCL memory object created from a media surface is used while it is not currently acquired by OpenCL, the call attempting to
use that OpenCL memory object will return
CL_DX9_MEDIA_SURFACE_NOT_ACQUIRED_KHR.

If CL_CONTEXT_INTEROP_USER_SYNC is not specified as CL_TRUE during context creation, clEnqueueAcquireDX9MediaSurfacesKHR provides the synchronization guarantee that any media adapter API calls involving the interop device(s) used in the OpenCL context made before clEnqueueAcquireDX9MediaSurfacesKHR is called will complete executing before event reports completion and before the execution of any subsequent OpenCL work issued in command_queue begins. If the context was created with properties specifying CL_CONTEXT_INTEROP_USER_SYNC as CL_TRUE, the user is responsible for guaranteeing that any media adapter API calls involving the interop device(s) used in the OpenCL context made before clEnqueueAcquireDX9MediaSurfacesKHR is called have completed before calling clEnqueueAcquireDX9MediaSurfacesKHR.

command_queue is a valid command-queue.

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

mem_objects is a pointer to a list of OpenCL memory objects that were created from media surfaces.

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

event returns an event object that identifies this 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

```c
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 events specified in 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 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.
9.7.7.5 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 9.10.3 and table 9.10.4.

<table>
<thead>
<tr>
<th>FOUR CC code</th>
<th>CL image format (channel order, channel data type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOURCC(‘N’,’V’,’1’,’2’), Plane 0</td>
<td>CL_R, CL_UNORM_INT8</td>
</tr>
<tr>
<td>FOURCC(‘N’,’V’,’1’,’2’), Plane 1</td>
<td>CL_RG, CL_UNORM_INT8</td>
</tr>
<tr>
<td>FOURCC(‘Y’,’V’,’1’,’2’), Plane 0</td>
<td>CL_R, CL_UNORM_INT8</td>
</tr>
<tr>
<td>FOURCC(‘Y’,’V’,’1’,’2’), Plane 1</td>
<td>CL_R, CL_UNORM_INT8</td>
</tr>
<tr>
<td>FOURCC(‘Y’,’V’,’1’,’2’), Plane 2</td>
<td>CL_R, CL_UNORM_INT8</td>
</tr>
</tbody>
</table>

Table 9.10.3 YUV FourCC codes and corresponding OpenCL image format

In table 9.10.3, 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>
<thead>
<tr>
<th>D3D format</th>
<th>CL image format (channel order, channel data type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3DFMT_R32F</td>
<td>CL_R, CL_FLOAT</td>
</tr>
<tr>
<td>D3DFMT_R16F</td>
<td>CL_R, CL_HALF_FLOAT</td>
</tr>
<tr>
<td>D3DFMT_L16</td>
<td>CL_R, CL_UNORM_INT16</td>
</tr>
<tr>
<td>D3DFMT_A8</td>
<td>CL_A, CL_UNORM_INT8</td>
</tr>
<tr>
<td>D3DFMT_L8</td>
<td>CL_R, CL_UNORM_INT8</td>
</tr>
<tr>
<td>D3DFMT_G32R32F</td>
<td>CL_RG, CL_FLOAT</td>
</tr>
<tr>
<td>D3DFMT_G16R16F</td>
<td>CL_RG, CL_HALF_FLOAT</td>
</tr>
<tr>
<td>D3DFMT_G16R16</td>
<td>CL_RG, CL_UNORM_INT16</td>
</tr>
<tr>
<td>D3DFMT_A8L8</td>
<td>CL_RG, CL_UNORM_INT8</td>
</tr>
<tr>
<td>D3DFMT_A32B32G32R32F</td>
<td>CL_RGBA, CL_FLOAT</td>
</tr>
</tbody>
</table>

5 Note that D3D9 format names seem to imply that the order of the color channels are switched relative to OpenCL but this is not the case. For example, layout of channels for each pixel for D3DFMT_A32FB32FG32FR32F is the same as CL_RGBA, CL_FLOAT.
<table>
<thead>
<tr>
<th>Direct3D Format</th>
<th>OpenCL Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3DFMT_A16B16G16R16F</td>
<td>CL_RGBA, CL_HALF_FLOAT</td>
</tr>
<tr>
<td>D3DFMT_A16B16G16R16</td>
<td>CL_RGBA, CL_UNORM_INT16</td>
</tr>
<tr>
<td>D3DFMT_A8B8G8R8</td>
<td>CL_RGBA, CL_UNORM_INT8</td>
</tr>
<tr>
<td>D3DFMT_X8B8G8R8</td>
<td>CL_RGBA, CL_UNORM_INT8</td>
</tr>
<tr>
<td>D3DFMT_A8R8G8B8</td>
<td>CL_BGRA, CL_UNORM_INT8</td>
</tr>
<tr>
<td>D3DFMT_X8R8G8B8</td>
<td>CL_BGRA, CL_UNORM_INT8</td>
</tr>
</tbody>
</table>

**Table 9.10.4** List of Direct3D and corresponding OpenCL image formats
9.8 Sharing Memory Objects with Direct3D 11

9.8.1 Overview

This section describes the cl_khr_d3d11_sharing extension. The goal of this extension is to provide interoperability between OpenCL and Direct3D 11. This is designed to function analogously to the OpenGL interoperability as defined in sections 9.7 and 9.8.

9.8.2 Header File

As currently proposed the interfaces for this extension would be provided in cl_d3d11.h.

9.8.3 New Procedures and Functions

```c
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)
```

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

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

```c
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)

9.8.4 New Tokens

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

CL_D3D11_DEVICE_KHR 0x4019
CL_D3D11_DXGI_ADAPTER_KHR 0x401A

Accepted as a set of Direct3D 11 devices in the d3d_device_set parameter of clGetDeviceIDsFromD3D11KHR:

CL_PREFERRED_DEVICES_FOR_D3D11_KHR 0x401B
CL_ALL_DEVICES_FOR_D3D11_KHR 0x401C

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

CL_CONTEXT_D3D11DEVICE_KHR 0x401D

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

CL_CONTEXT_D3D11_PREFER_SHARED_RESOURCES_KHR 0x402D

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

CL_MEM_D3D11RESOURCE_KHR 0x401E

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

CL_IMAGE_D3D11_SUBRESOURCE_KHR 0x401F
Returned in the `param_value` parameter of `clGetEventInfo` when `param_name` is `CL_EVENT_COMMAND_TYPE`:

- `CL_COMMAND_ACQUIRE_D3D11_OBJECTS_KHR` 0x4020
- `CL_COMMAND_RELEASE_D3D11_OBJECTS_KHR` 0x4021

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` -1006

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` -1007

Returned by `clEnqueueAcquireD3D11ObjectsKHR` when any of `mem_objects` are currently acquired by OpenCL

- `CL_D3D11_RESOURCE_ALREADY_ACQUIRED_KHR` -1008

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

- `CL_D3D11_RESOURCE_NOT_ACQUIRED_KHR` -1009

### 9.8.5 Additions to Chapter 4 of the OpenCL 2.1 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`:

<table>
<thead>
<tr>
<th><code>cl_context_properties enum</code></th>
<th><code>Property value</code></th>
<th><code>Description</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CL_CONTEXT_D3D11_DEVICE_KHR</code></td>
<td><code>ID3D11Device *</code></td>
<td>Specifies the ID3D11Device * to use for Direct3D 11 interoperability.</td>
</tr>
</tbody>
</table>
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:

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

9.8.6 Additions to Chapter 5 of the OpenCL 2.1 Specification

Add to the list of errors for `clGetMemObjectInfo`:

- CL_INVALID_D3D11RESOURCE_KHR if `param_name` is CL_MEM_D3D11RESOURCE_KHR and `memobj` was not created by the function `clCreateFromD3D11BufferKHR, clCreateFromD3D11Texture2DKHR, or clCreateFromD3D11Texture3DKHR."`

Extend table 5.13 to include the following entry.

<table>
<thead>
<tr>
<th>cl_mem_info</th>
<th>Return type</th>
<th>Info. returned in param_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_MEM_D3D11_</td>
<td>ID3D11Resource *</td>
<td>If <code>memobj</code> was created using</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>cl_image_info</th>
<th>Return type</th>
<th>Info, returned in <code>param_value</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_MEM_D3D11_SUBRESOURCE_KHR</td>
<td>UINT</td>
<td>If <em>image</em> was created using <code>clCreateFromD3D11Texture2DKHR</code>, or <code>clCreateFromD3D11Texture3DKHR</code>, returns the <code>subresource</code> argument specified when <em>image</em> was created.</td>
</tr>
</tbody>
</table>

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

### 9.8.7 Sharing Memory Objects with Direct3D 11 Resources

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

#### 9.8.7.1 Querying OpenCL Devices Corresponding to Direct3D 11 Devices

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

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

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

(platform refers to the platform ID returned by clGetPlatformIDs.

d3d_device_source specifies the type of d3d_object, and must be one of the values shown in table 9.11.1.

d3d_object specifies the object whose corresponding OpenCL devices are being queried. The type of d3d_object must be as specified in table 9.11.1.

d3d_device_set specifies the set of devices to return, and must be one of the values shown in table 9.11.2.

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>
<thead>
<tr>
<th>cl_d3d_device_source_khr</th>
<th>Type of d3d_object</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_D3D11_DEVICE_KHR</td>
<td>ID3D11Device *</td>
</tr>
<tr>
<td>CL_D3D11_DXGI_ADAPTER_KHR</td>
<td>IDXGIAdapter *</td>
</tr>
</tbody>
</table>

Table 9.11.1  Types used to specify the object whose corresponding OpenCL devices are being queried by clGetDeviceIDsFromD3D11KHR

<table>
<thead>
<tr>
<th>cl_d3d_device_set_khr</th>
<th>Devices returned in devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_PREFERRED_DEVICES_FOR_D3D11_KHR</td>
<td>The preferred OpenCL devices associated with the specified Direct3D object.</td>
</tr>
<tr>
<td>CL_ALL_DEVICES_FOR_D3D11_KHR</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 9.11.2  Sets of devices queriable using clGetDeviceIDsFromD3D11KHR

9.8.7.2 Lifetime of Shared Objects

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

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

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

The function

\[
\text{cl_mem clCreateFromD3D11BufferKHR (cl_context context,}
\]
\[
\text{cl_mem_flags flags,}
\]
\[
\text{ID3D11Buffer *resource,}
\]
\[
\text{cl_int *errcode_ret)}
\]

creates an OpenCL buffer object from a Direct3D 11 buffer.

\textit{context} is a valid OpenCL context created from a Direct3D 11 device.

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

\textit{resource} is a pointer to the Direct3D 11 buffer to share.

\textit{errcode_ret} will return an appropriate error code. If \textit{errcode_ret} is NULL, no error code is returned.

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

\begin{itemize}
  \item \texttt{CL_INVALID_CONTEXT} if \textit{context} is not a valid context.
  \item \texttt{CL_INVALID_VALUE} if values specified in \textit{flags} are not valid.
  \item \texttt{CL_INVALID_D3D11_RESOURCE_KHR} if \textit{resource} is not a Direct3D 11 buffer resource, if \textit{resource} was created with the D3D11\_USAGE flag D3D11\_USAGE\_IMMUTABLE, if a \texttt{cl_mem} from \textit{resource} has already been created using \texttt{clCreateFromD3D11BufferKHR}, or if \textit{context} was not created against the same Direct3D 11 device from which \textit{resource} was created.
  \item \texttt{CL_OUT_OF_HOST_MEMORY} if there is a failure to allocate resources required by the OpenCL implementation on the host.
\end{itemize}

The size of the returned OpenCL buffer object is the same as the size of \textit{resource}. This call will increment the internal Direct3D reference count on \textit{resource}. The internal Direct3D reference count on \textit{resource} will be decremented when the OpenCL reference count on the returned OpenCL memory object drops to zero.
9.8.7.4 Sharing Direct3D 11 Texture and Resources as OpenCL Image Objects

The function

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

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

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

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

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

`subresource` is the subresource of `resource` to share.

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

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

- CL_INVALID_CONTEXT if `context` is not a valid context.
- CL_INVALID_VALUE if values specified in `flags` are not valid or if `subresource` is not a valid subresource index for `resource`.
- CL_INVALID_D3D11_RESOURCE_KHR if `resource` is not a Direct3D 11 texture resource, if `resource` was created with the D3D11_USAGE flag D3D11_USAGE_IMMUTABLE, if `resource` is a multisampled texture, if a cl_mem from subresource `subresource` of `resource` has already been created using `clCreateFromD3D11Texture2DKHR`, or if `context` was not created against the same Direct3D 10 device from which `resource` was created.
- CL_INVALID_IMAGE_FORMAT_DESCRIPTOR if the Direct3D 11 texture format of `resource` is not listed in table 9.11.3 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 table 9.11.3.

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

\[
\text{cl_mem clCreateFromD3D11Texture3DKHR (cl_context context, cl_mem_flags flags, ID3D11Texture3D *resource, UINT subresource, cl_int *errcode_ret)}
\]

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_D3D11RESOURCE_KHR if resource is not a Direct3D 11 texture resource, if resource was created with the D3D11_USAGE flag.
D3D11_USAGE_IMMUTABLE, if \textit{resource} is a multisampled texture, if a \texttt{cl_mem} from subresource \textit{subresource} of \textit{resource} has already been created using \texttt{clCreateFromD3D11Texture3DKHR}, or if \textit{context} was not created against the same Direct3D 11 device from which \textit{resource} was created.

- \texttt{CL_INVALID_IMAGE_FORMAT_DESCRIPTOR} if the Direct3D 11 texture format of \textit{resource} is not listed in \textit{table 9.11.3} or if the Direct3D 11 texture format of \textit{resource} does not map to a supported OpenCL image format.

- \texttt{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 \textit{subresource} of \textit{resource}. The channel type and order of the returned OpenCL 3D image object is determined by the format of \textit{resource} by \textit{table 9.9.3}.

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

<table>
<thead>
<tr>
<th>DXGI format</th>
<th>CL image format</th>
</tr>
</thead>
<tbody>
<tr>
<td>DXGI_FORMAT_R32G32B32A32_FLOAT</td>
<td>CL_RGBA, CL_FLOAT</td>
</tr>
<tr>
<td>DXGI_FORMAT_R32G32B32A32_UINT</td>
<td>CL_RGBA, CL_UNSIGNED_INT32</td>
</tr>
<tr>
<td>DXGI_FORMAT_R32G32B32A32_SINT</td>
<td>CL_RGBA, CL_SIGNED_INT32</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16B16A16_FLOAT</td>
<td>CL_RGBA, CL_HALF_FLOAT</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16B16A16_UINT</td>
<td>CL_RGBA, CL_UNSIGNED_INT16</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16B16A16_SNORM</td>
<td>CL_RGBA, CL_SNORM_INT16</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16B16A16_SINT</td>
<td>CL_RGBA, CL_SIGNED_INT16</td>
</tr>
<tr>
<td>DXGI_FORMAT_B8G8R8A8_UNORM</td>
<td>CL_BGRA, CL_UNORM_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8B8A8_UNORM</td>
<td>CL_RGBA, CL_UNORM_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8B8A8_UINT</td>
<td>CL_RGBA, CL_UNSIGNED_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8B8A8_SNORM</td>
<td>CL_RGBA, CL_SNORM_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8B8A8_SINT</td>
<td>CL_RGBA, CL_SIGNED_INT8</td>
</tr>
<tr>
<td>DXGI_FORMAT_R32G32_FLOAT</td>
<td>CL_RG, CL_FLOAT</td>
</tr>
<tr>
<td>DXGI_FORMAT_R32G32_UINT</td>
<td>CL_RG, CL_UNSIGNED_INT32</td>
</tr>
<tr>
<td>DXGI_FORMAT_R32G32_SINT</td>
<td>CL_RG, CL_SIGNED_INT32</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16_FLOAT</td>
<td>CL_RG, CL_HALF_FLOAT</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16_UNORM</td>
<td>CL_RG, CL_UNSIGNED_INT16</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16_UINT</td>
<td>CL_RG, CL_UNSIGNED_INT16</td>
</tr>
<tr>
<td>DXGI_FORMAT_R16G16_SNORM</td>
<td>CL_RG, CL_SNORM_INT16</td>
</tr>
</tbody>
</table>
Table 9.11.3  List of Direct3D 11 and corresponding OpenCL image formats

<table>
<thead>
<tr>
<th>DirectX Format</th>
<th>OpenCL Type 1</th>
<th>OpenCL Type 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>DXGI_FORMAT_R16G16_SINT</td>
<td>CL_RG, CL_SIGNED_INT16</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8_UNORM</td>
<td>CL_RG, CL_UNORM_INT8</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8_UINT</td>
<td>CL_RG, CL_UNSIGNED_INT8</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8_SNORM</td>
<td>CL_RG, CL_SNORM_INT8</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R8G8_SINT</td>
<td>CL_RG, CL_SIGNED_INT8</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R32_FLOAT</td>
<td>CL_R, CL_FLOAT</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R32_UINT</td>
<td>CL_R, CL_UNSIGNED_INT32</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R32_SINT</td>
<td>CL_R, CL_SIGNED_INT32</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R16_FLOAT</td>
<td>CL_R, CL_HALF_FLOAT</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R16_UNORM</td>
<td>CL_R, CL_UNORM_INT16</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R16_UINT</td>
<td>CL_R, CL_UNSIGNED_INT16</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R16_SNORM</td>
<td>CL_R, CL_SNORM_INT16</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R16_SINT</td>
<td>CL_R, CL_SIGNED_INT16</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R8_UNORM</td>
<td>CL_R, CL_UNORM_INT8</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R8_UINT</td>
<td>CL_R, CL_UNSIGNED_INT8</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R8_SNORM</td>
<td>CL_R, CL_SNORM_INT8</td>
<td></td>
</tr>
<tr>
<td>DXGI_FORMAT_R8_SINT</td>
<td>CL_R, CL_SIGNED_INT8</td>
<td></td>
</tr>
</tbody>
</table>

9.8.7.5 Querying Direct3D properties of memory objects created from Direct3D 11 resources

Properties of Direct3D 11 objects may be queried using clGetMemObjectInfo and clGetImageInfo with param_name CL_MEM_D3D11RESOURCE_KHR and CL_IMAGE_D3D11_SUBRESOURCE_KHR respectively as described in sections 5.4.3 and 5.3.6.

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

The function

```c
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_D3D11RESOURCE_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

```c
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 not 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` 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.
9.9 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 extension extends CL / GL sharing (i.e. the cl_khr_gl_sharing extension) defined in section 9.7 to allow a CL image to be created from a GL depth or depth-stencil texture.

9.9.1 Additions to Chapter 5 of the OpenCL 2.1 Specification

The cl_khr_gl_depth_images extension extends CL / GL sharing by allowing a CL depth image to be created from a GL 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 2.1 specification.

<table>
<thead>
<tr>
<th>Enum values that can be specified in channel_order</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_DEPTH_STENCIL. This format can only be used if channel data type = CL_UNORM_INT24 or CL_FLOAT.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image Channel Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_UNORM_INT24</td>
<td>Each channel component is a normalized unsigned 24-bit integer value</td>
</tr>
<tr>
<td>CL_FLOAT</td>
<td>Each channel component is a single precision floating-point value</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>num_channels</th>
<th>channel_order</th>
<th>channel_data_type</th>
<th>read / write</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CL_DEPTH_STENCIL</td>
<td>CL_UNORM_INT24, CL_FLOAT</td>
<td>read only</td>
</tr>
</tbody>
</table>

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 = `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.

### 9.9.2 Additions to Chapter 9.7 of the OpenCL 2.1 Extension Specification

The following new image formats are added to table 9.4 in section 9.6.3.1 of the OpenCL 2.1 extension specification. If a GL texture object with an internal format from table 9.4 is successfully created by OpenGL, then there is guaranteed to be a mapping to one of the corresponding CL image format(s) in that table.

<table>
<thead>
<tr>
<th>GL internal format</th>
<th>CL image format (channel order, channel data type)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GL_DEPTH_COMPONENT32F</code></td>
<td><code>CL_DEPTH, CL_FLOAT</code></td>
</tr>
<tr>
<td><code>GL_DEPTH_COMPONENT16</code></td>
<td><code>CL_DEPTH, CL_UNORM_INT16</code></td>
</tr>
<tr>
<td><code>GL_DEPTH24_STENCIL8</code></td>
<td><code>CL_DEPTH_STENCIL, CL_UNORM_INT24</code></td>
</tr>
<tr>
<td><code>GL_DEPTH32F_STENCIL8</code></td>
<td><code>CL_DEPTH_STENCIL, CL_FLOAT</code></td>
</tr>
</tbody>
</table>
9.10  Sharing of CL / GL MSAA Textures

This extension extends the CL / GL sharing (i.e. the cl_khr_gl_sharing_extension) defined in section 9.7 to allow a CL image to be created from a GL 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.

9.10.1 Additions to Chapter 9.7 of the OpenCL 2.1 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 CL image objects can only be read from a kernel. Multi-sample CL 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 table 9.5:

<table>
<thead>
<tr>
<th>cl_gl_texture_info</th>
<th>Return Type</th>
<th>Info. returned in param_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_GL_NUM_SAMPLES</td>
<td>GLsizei</td>
<td>The samples argument passed to glTexImage2DMultisample or glTexImage3DMultisample.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If image is not a MSAA texture, 1 is returned.</td>
</tr>
</tbody>
</table>
9.10.2 Additions to Chapter 5 of the OpenCL 2.1 Specification

The formats described in tables 5.8.a and 5.8.b of the OpenCL 2.1 specification and the additional formats added to this table described in section 9.12.1 also support CL images created from a GL 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.
9.11 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`.

9.11.1 Additions to Chapter 4 of the OpenCL 2.1 Specification

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

<table>
<thead>
<tr>
<th>cl_context_properties enum</th>
<th>Property value</th>
<th>Description</th>
</tr>
</thead>
</table>
| CL_CONTEXT_MEMORY_ INITIALIZE_KHR | cl_context_memory_initialize_khr | Describes which memory types for the context must be initialized. This is a bit-field, where the following values are currently supported:
  - CL_CONTEXT_MEMORY_INITIALIZE_LOCAL_KHR – Initialize local memory to zeros.
  - CL_CONTEXT_MEMORY_INITIALIZE_PRIVATE_KHR – Initialize private memory to zeros. |

9.11.2 Additions to Chapter 6 of the OpenCL 2.1 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.
9.12 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`.

9.12.1 Additions to Chapter 4 of the OpenCL 2.1 Specification

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

<table>
<thead>
<tr>
<th>cl_device_info</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_DEVICE_TERMINATE_CAPABILITY_KHR</td>
<td>cl_device_terminate_capability_khr</td>
<td>Describes the termination capability of the OpenCL device. This is a bitfield where a value of CL_DEVICE_TERMINATE_CAPABILITY_CONTEXT_KHR indicates that context termination is supported.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>cl_context_properties enum</th>
<th>Property value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_CONTEXT_TERMINATE_KHR</td>
<td>cl_bool</td>
<td>Specifies whether the context can be terminated. The default value is CL_FALSE.</td>
</tr>
</tbody>
</table>

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.

A new function is added. The 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).
9.13 SPIR 1.2 Binaries

This extension adds support to create an OpenCL program object from a Standard Portable Intermediate Representation (SPIR) instance. SPIR is a vendor neutral non-source representation for OpenCL C programs that has since been superceded by the SPIR-V standard.

The extension name is `cl_khr_spir`.

9.13.1 Additions to Chapter 4 of the OpenCL 2.1 Specification

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

<table>
<thead>
<tr>
<th>cl_device_info</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_DEVICE_SPIR_VERSIONS</td>
<td>char[]</td>
<td>A space separated list of SPIR versions supported by the device. For example returning “1.2 2.0” in this query implies that SPIR version 1.2 and 2.0 are supported by the implementation.</td>
</tr>
</tbody>
</table>

9.13.2 Additions to Chapter 5 of the OpenCL 2.1 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 5.18` (clGetProgramBuildInfo) to add a potential value CL_PROGRAM_BINARY_TYPE_INTERMEDIATE:

<table>
<thead>
<tr>
<th>cl_program_build_info</th>
<th>Return Type</th>
<th>Info. returned in <code>param_value</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_PROGRAM_BINARY_TYPE</td>
<td>cl_program_binary_type</td>
<td>CL_PROGRAM_BINARY_TYPE_INTERMEDIATE – An intermediate (non-source) representation for the program is loaded as a binary. The program must be further processed with clCompileProgram or clBuildProgram. If processed with clCompileProgram, the result will be a binary of type</td>
</tr>
</tbody>
</table>
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:

“The 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:

“The 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`.”
9.14 OpenCL Installable Client Driver (ICD)

9.14.1 Overview

This is 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. If this extension is supported by an implementation, the string cl_khr_icd will be present in the CL_PLATFORM_EXTENSIONS string described in table 4.1.

9.14.2 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:

```c
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.

A link to source code which defines the entries in the function table structure _cl_icd_dispatch is available in the Sample Code section of this document. 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 are ignored, with the exception of clGetExtensionFunctionAddressForPlatform which is described below.
9.14.3 ICD Data

A Vendor ICD is defined by two pieces of data:

- The Vendor ICD library specifies a library which contains the OpenCL entrypoints 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. See Additions to Chapter 9 for details on the mechanism through which this is accomplished. The vendor suffix string is optional.

9.14.4 ICD Loader Vendor Enumeration on Windows

To enumerate Vendor ICDs on Windows, the ICD Loader scans the values in the registry key HKEY_LOCAL_MACHINE\SOFTWARE\Khronos\OpenCL\Vendors. For each value in this key which has DWORD data set to 0, the ICD Loader opens the dynamic link library specified by the name of the value using LoadLibraryA.

For example, if the registry contains the following value

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

then the ICD will open the library "c:\vendor a\vndra_ocl.dll".

9.14.5 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".

9.14.6 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".

9.14.7 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 clGetExtensionFunctionAddressForPlatform. 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.

9.14.8 New Procedures and Functions

\[
\text{cl_int clIcdGetPlatformIDsKHR}(\text{cl_uint num_entries}, \\
\text{cl_platform_id *platforms,} \\
\text{cl_uint *num_platforms});
\]

9.14.9 New Tokens

Accepted as param_name to the function clGetPlatformInfo

\[
\text{CL_PLATFORM_ICD_SUFFIX_KHR} \quad 0x0920
\]

Returned by clGetPlatformIDs when no platforms are found

\[
\text{CL_PLATFORM_NOT_FOUND_KHR} \quad -1001
\]
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:

\[
\text{cl_int } \text{clIcdGetPlatformIDsKHR (cl_uint num_entries,}
\]
\[
\text{cl_platform_id } \ast \text{platforms,}
\]
\[
\text{cl_uint } \ast \text{num_platforms);}\]

\text{num_entries} is the number of cl_platform_id entries that can be added to \text{platforms}. If \text{platforms} is not NULL, then \text{num_entries} must be greater than zero.

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

\text{num_platforms} returns the number of OpenCL platforms available. If \text{num_platforms} is NULL, then this argument is ignored.

\text{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 \text{num_entries} is equal to zero and \text{platforms} is not NULL or if both \text{num_platforms} and \text{platforms} are NULL."

Add the following to table 4.1:

<table>
<thead>
<tr>
<th>cl_platform_info enum</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_PLATFORM_ICD_SUFFIX_KHR</td>
<td>char[]</td>
<td>The function name suffix used to identify extension functions to be directed to this platform by the ICD Loader.</td>
</tr>
</tbody>
</table>
9.14.11 Additions to Chapter 9 of the OpenCL 2.1 Extension Specification

Add the following paragraph to the end of Section 9.2:

"For functions supported by the ICD Loader, `clGetExtensionFunctionAddressForPlatform` will return the function pointer of the ICD Loader implementation. For extension functions which the ICD Loader is unaware of, the function `clGetExtensionFunctionAddressForPlatform` will determine the vendor implementation to return based on the string passed in. The ICD Loader will return the result from querying `clGetExtensionFunctionAddressForPlatform` on the vendor ICD enumerated by the ICD Loader whose ICD suffix is a suffix of the function name being queried. If no such vendor exists or the suffix of the function is KHR or EXT then `clGetExtensionFunctionAddressForPlatform` will return NULL."


The official source for the ICD loader is available at the Khronos website. The complete `cl_icd_dispatch` structure is defined in the header `icd_dispatch.h` which is available as a part of the source code.

9.14.13 Issues

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

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

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.
9.15 Mipmaps

This extension adds support for mipmaps. This proposal is implemented as two optional extensions. The cl_khr_mipmap_image extension implements support to create a mip-mapped image, enqueue commands to read/write/copy/map 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.

9.15.1 Additions to Chapter 5 of the OpenCL 2.1 Specification

9.15.1.1 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 > 1 in 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.

### 9.15.1.2 Additions to section 5.7 – Sampler Objects

Add the following sampler properties to `table 5.15` that can be specified when a sampler object is created using `clCreateSamplerWithProperties`.

<table>
<thead>
<tr>
<th>cl_sampler_properties enum</th>
<th>Property Value</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_SAMPLER_MIP_FILTER_MODE_KHR</td>
<td>cl_filter_mode</td>
<td>CL_FILTER_NEAREST_KHR</td>
</tr>
<tr>
<td>CL_SAMPLER_LOD_MIN_KHR</td>
<td>cl_float</td>
<td>0.0f</td>
</tr>
<tr>
<td>CL_SAMPLER_LOD_MAX_KHR</td>
<td>cl_float</td>
<td>MAXFLOAT</td>
</tr>
</tbody>
</table>

**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.

### 9.15.2 Additions to section 9.7 – Sharing Memory Objects with OpenGL / OpenGL ES Texture Objects

If the `cl_khr_mipmap_image` extension is supported by the OpenCL device, the `cl_khr_gl_sharing` extension adds support for creating a mip-mapped CL image from a
mip-mapped GL texture.

To create a mip-mapped CL image from a mip-mapped GL texture, the \textit{miplevel} argument to \texttt{clCreateFromGLTexture} should be a negative value. If \textit{miplevel} is a negative value then a CL mipmapped image object is created from a mipmapped GL texture object instead of a CL image object for a specific mplevel of a GL texture.

\textbf{NOTE:} For a detailed description of how the level of detail is computed, please refer to \textit{section 3.9.7} of the OpenGL 3.0 specification.
9.16 Creating CL image objects from EGL images

9.16.1 Overview

This section describes the cl_khr_egl_image extension. This extension provides a mechanism for creating derived resources, such as OpenCL image objects, from EGLImages.

9.16.2 New Procedures and Functions

cl_mem clCreateFromEGLImageKHR (cl_context context, CLEglDisplayKHR display, CLEglImageKHR image, cl_mem_flags flags, const cl_egl_image_properties_khr *properties, cl_int *errcode_ret);

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

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

9.16.3 New Tokens

New error codes:

    CL_EGLRESOURCENOTACQUIRED_KHR     -1092
    CL_INVALID_EGL_OBJECT_KHR         -1093

New command types:
9.16.4 Additions to Chapter 5 of the OpenCL 2.1 Specification

In section 5.2.4, add the following text after the paragraph defining clCreateImage:

`clCreateFromEGLImageKHR` creates an EGLImage target of type cl_mem from the
EGLImage source provided as `image`.

`display` should be of type EGLDisplay, cast into the type CLeglDisplayKHR.

`image` should be of type EGLImageKHR, cast into the type CLeglImageKHR. Assuming no
errors are generated in this function, the resulting image object will be an EGLImage target of
the specified EGLImage `image`. The resulting cl_mem is an image object which may be used
normally by all OpenCL operations. This maps to an image2d_t type in OpenCL kernel code.

`flags` is a bit-field that is used to specify usage information about the memory object being
created.

The possible values for `flags` are: CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and
CL_MEM_READ_WRITE.

For OpenCL 1.2 `flags` also accepts: CL_MEM_HOST_WRITE_ONLY,
CL_MEM_HOST_READ_ONLY or CL_MEM_HOST_NO_ACCESS.

This extension only requires support for CL_MEM_READ_ONLY, and for OpenCL 1.2
CL_MEM_HOST_NO_ACCESS. For OpenCL 1.1, a CL_INVALID_OPERATION will be returned
for images which do not support host mapping.

If the value passed in `flags` is not supported by the OpenCL implementation it will return
CL_INVALID_VALUE. The accepted `flags` may be dependent upon the texture format used.

`properties` specifies a list of property names and their corresponding values. Each property name
is immediately followed by the corresponding desired value. The list is terminated with 0. No
properties are currently supported with this version of the extension. `properties` can be NULL.

Errors

- 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.

**Lifetime of Shared Objects**

An OpenCL memory object created from an EGL image remains valid according to the lifetime behaviour 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 (EGL sibling) the image remains valid.

**9.12.7.1 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 synchronisation 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.

9.12.7 Sharing memory objects created from EGL resources between EGLDisplays and OpenCL contexts

The function

```c
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.

9.16.5 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 behaviour 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 seperated then the actual dimensionality of the seperated 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.
9.17 Creating CL event objects from EGL sync objects

9.17.1 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.

9.17.2 New Procedures and Functions

```c
cl_event clCreateEventFromEGLSyncKHR (cl_context context,
                                      CLEglSyncKHR sync,
                                      CLEglDisplayKHR display,
                                      cl_int *errcode_ret);
```

9.17.3 New Tokens

Returned by clCreateEventFromEGLSyncKHR if `sync` is not a valid EGLSyncKHR handle created with respect to EGLDisplay `display`:

```
CL_INVALID_EGL_OBJECT_KHR -1093
```

Returned by clGetEventInfo when `param_name` is CL_EVENT_COMMAND_TYPE:

```
CL_COMMAND_EGL_FENCE_SYNC_OBJECT_KHR 0x202F
```

9.17.4 Additions to Chapter 5 of the OpenCL 2.1 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.

### 9.17.5 Additions to Chapter 9 of the OpenCL 2.1 Specification

Replace the second paragraph of section 9.7.6.1 (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 synchronisation* 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."
### 9.17.6 Issues

Most issues are shared with `cl_khr_gl_event` and are resolved as described in that extension.

1) Should we support implicit synchronization?

RESOLVED: No, as this may be very difficult since the synchronization would not be with EGL, it would be with currently bound EGL client APIs. It would be necessary to know which client APIs might be bound, to validate that they're associated with the EGLDisplay associated with the OpenCL context, and to reach into each such context.

2) Do we need to have typedefs to use EGL handles in OpenCL?

RESOLVED Using typedefs for EGL handles.

3) Should we restrict which CL APIs can be used with this cl_event?

RESOLVED Use is limited to `clEnqueueAcquire***` calls only.

4) What is the desired behaviour for this extension when EGLSyncKHR is of a type other than `EGL_SYNC_FENCE_KHR`?

RESOLVED This extension only requires support for `EGL_SYNC_FENCE_KHR`. Support of other types is an implementation choice, and will result in `CL_INVALID_EGL_OBJECT_KHR` if unsupported.
9.18 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 describe the scheduling behavior guaranteed.

9.18.1 Host-side API modifications

The function clCreateCommandQueueWithProperties (Section 5.1) is extended to support a priority value as part of the properties argument.

Add to table 5.1:

<table>
<thead>
<tr>
<th>Queue Properties</th>
<th>Property Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_QUEUE_PRIORITY_KHR</td>
<td>cl_uint</td>
<td>Specifies the priority value, which may be one of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CL_QUEUE_PRIORITY_HIGH_KHR – indicates highest priority.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CL_QUEUE_PRIORITY_MED_KHR – indicates medium priority.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CL_QUEUE_PRIORITY_LOW_KHR – indicates lowest priority.</td>
</tr>
</tbody>
</table>

The priority property applies to OpenCL command queues that belong to the same OpenCL context.

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.
9.19 Throttle Hints

This section describes the **cl_khr_throttle_hints** extension. This extension adds throttle hints for OpenCL, but does not specify the throttling behaviour or minimum guarantees. It is expected that the user guide associated with each implementation which supports this extension describe the throttling behaviour guaranteed.

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**).

### 9.19.1 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.

Add to **table 5.1**:

<table>
<thead>
<tr>
<th>Queue Properties</th>
<th>Property Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CL_QUEUE_THROTTLE_KHR</strong></td>
<td>cl_uint</td>
<td>Specifies the throttle value, which may be one of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>CL_QUEUE_THROTTLE_HIGH_KHR</strong> – indicates highest energy consumption.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>CL_QUEUE_THROTTLE_MED_KHR</strong> – indicates normal behavior.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>CL_QUEUE_THROTTLE_LOW_KHR</strong> – indicates least energy consumption.</td>
</tr>
</tbody>
</table>

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_PRIORITY_KHR** property is specified and the queue is a **CL_QUEUE_ON_DEVICE**.
Appendix A – Changes

A.1 Summary of changes from OpenCL 2.0

The following features are added to 2.0:

- OpenCL 2.1 KHR extension `cl_khr_priority_hints` has been added.
- OpenCL 2.1 KHR extension `cl_khr_throttle_hints` has been added.

The following features are deprecated (see glossary) in OpenCL 2.1:

- OpenCL 2.0 KHR extension `cl_khr_il_program` has been deprecated. The feature is now core.
- OpenCL 2.0 KHR extension `cl_khr_subgroups` has been deprecated. The feature is now core.
Index - APIs

clCreateEventFromEGLSyncKHR, 114
clCreateEventFromGLsyncKHR, 34
clCreateFromDX9MediaSurfaceKHR, 61
clCreateFromD3D10BufferKHR, 45
clCreateFromD3D10Texture2DKHR, 46
clCreateFromD3D10Texture3DKHR, 47
clCreateFromD3D11BufferKHR, 76
clCreateFromD3D11Texture2DKHR, 77
clCreateFromD3D11Texture3DKHR, 78
clCreateFromEGLImageKHR, 105
clCreateFromGLBuffer, 21
clCreateFromGLRenderbuffer, 25
clCreateFromGLTexture, 22
clEnqueueAcquire
    DX9MediaSurfacesKHR, 63
clEnqueueAcquireD3D10ObjectsKHR, 49, 64, 65
clEnqueueAcquireD3D11ObjectsKHR, 80
clEnqueueAcquireEGLObjectsKHR, 105
clEnqueueAcquireGLObjects, 29
clEnqueueRelease
    DX9MediaSurfacesKHR, 65
clEnqueueReleaseD3D10ObjectsKHR, 51
clEnqueueReleaseD3D11ObjectsKHR, 82
clEnqueueReleaseEGLObjectsKHR, 105
clEnqueueReleaseGLObjects, 30
clGetDeviceIDsFromD3D10KHR, 43
clGetDeviceIDsFromD3D11KHR, 74
clGetDeviceIDsFromDX9MediaAdapterKHR, 59
clGetExtensionFunctionAddressForPlatform, 9
clGetGLObjectInfo, 27
clGetGLTextureInfo, 28
clIcdGetPlatformIDsKHR, 98, 99
clTerminateContextKHR, 92