The OpenCL™ Extension Specification

Khronos® OpenCL Working Group

Version v3.0.16, Thu, 04 Apr 2024 12:00:0 +0000: from git branch: main commit: 16880f9276828e66bb97017f10f34f97423d1bcf

Table of Contents

Chapter 1. Extensions Overview

Extensions are optional features which may be supported by OpenCL implementations. Extensions are not required to be supported by a conformant OpenCL implementation, but are expected to be widely available, and in some cases may define functionality that is likely to be required in a future revision of the OpenCL specification.

In the past, this document contained full specification language for Khronos-approved `khr` extensions, described in terms of changes to the core OpenCL Specification. This extension language has now been integrated into the OpenCL 3.0 Specification, and can be read in context there.

The remaining parts of this document describe general issues in *using* extensions, such as API Naming Conventions for Optional Extensions; OpenCL C Compiler Directives for Optional Extensions; and Getting OpenCL API Extension Function Pointers.

In addition, there is a section on Extensions to the OpenCL SPIR-V Environment.

Finally, the Quick Reference appendix summarizes khr extensions and links to them in the OpenCL API Specification. In some cases, extensions are mostly or entirely to the OpenCL C language rather than to the OpenCL API. Such extensions can be reached by following the links in the API Specification extension appendices.

1.1. Naming Convention for Optional Extensions

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

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

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

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

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

enum_name>_<*vendor_name*>.

1.2. Compiler Directives for Optional Extensions

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

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

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

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

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

The initial state of the compiler is as if the directive

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

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

Every extension which affects the OpenCL language semantics, syntax or adds built-in functions to

the language must create a preprocessor `#define` that matches the extension name string. This `#define` would be available in the language if and only if the extension is supported on a given implementation.

Example:

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

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

1.3. Getting OpenCL API Extension Function Pointers

The function

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

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

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

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

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

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

```
#ifndef extension_name
#define extension_name 1

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

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

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

#endif // _extension_name_
```

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

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

```
#ifndef cl_khr_gl_sharing
#define cl_khr_gl_sharing 1

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

// function pointer typedefs must use the
// following naming convention
typedef cl_int
        (CL_API_CALL *clGetGLContextInfoKHR_fn)(
            const cl_context_properties * /* properties */,
            cl_gl_context_info /* param_name */,
            size_t /* param_value_size */,
            void * /* param_value */,
            size_t * /*param_value_size_ret*/);

#endif // cl_khr_gl_sharing
```

Index

C

Appendix A: Extensions Promoted to Core Features

A.1. For OpenCL 1.1:

- The functionality previously described by `cl_khr_byte_addressable_store` is now part of the core feature set.
- The functionality previously described by `cl_khr_global_int32_base_atomics`, `cl_khr_global_int32_extended_atomics`, `cl_khr_local_int32_base_atomics`, and `cl_khr_local_int32_extended_atomics` is now part of the core feature set.

A.2. For OpenCL 1.2:

- The functionality previously described by `cl_khr_fp64` is now an optional core feature.

A.3. For OpenCL 2.0:

- The functionality described by `cl_khr_3d_image_writes` is part of the core feature set.
- The functionality described by `cl_khr_create_command_queue` is part of the core feature set.
- The functionality described by `cl_khr_depth_images` is now part of the core feature set.
- The functionality described by `cl_khr_image2d_from_buffer` is now part of the core feature set.

A.4. For OpenCL 2.1:

- The functionality described by `cl_khr_il_program` is now part of the core feature set.
- The API functionality described by `cl_khr_subgroups` is now part of the core API feature set, but the built-in functions described by `cl_khr_subgroups` must still be accessed as an extension to the OpenCL 2.0 C Language specification.

A.5. For OpenCL 3.0:

- The API functionality described by `cl_khr_extended_versioning` is now part of the core API feature set, with minor modifications.
- The built-in functions described by `cl_khr_subgroups` are now supported in OpenCL C 3.0 when the {opencl_c_subgroups} feature is supported.

Appendix B: Deprecated Extensions

B.1. For OpenCL 1.1:

- The `cl_khr_select_fprounding_mode` extension has been deprecated. Its use is no longer recommended.

Appendix C: Quick Reference

Each extension in this table includes a link to the corresponding appendix in the OpenCL 3.0 API Specification, which provides a fuller description and references to the actual extension specification language in the API and C Language Specifications.

Extension Name and Link	Brief Description	Status
cl_khr_3d_image_writes	Write to 3D images	Core Feature in OpenCL 2.0
cl_khr_async_work_group_copy_fence	Asynchronous Copy Fences	Extension
cl_khr_byte_addressable_store	Read and write from 8-bit and 16-bit pointers	Core Feature in OpenCL 1.1
cl_khr_command_buffer	Record and Replay Commands	Provisional Extension
cl_khr_command_buffer_multi_device	Allow a command-buffer to contain commands targeting different devices	Provisional Extension
cl_khr_command_buffer_mutable_dispatch	Modify kernel execution commands between enqueues of a command-buffer	Provisional Extension
cl_khr_create_command_queue	API to Create Command-Queues with Properties	Core Feature in OpenCL 2.0
cl_khr_d3d10_sharing	Share Direct3D 10 Buffers and Textures with OpenCL	Extension
cl_khr_d3d11_sharing	Share Direct3D 11 Buffers and Textures with OpenCL	Extension
cl_khr_depth_images	Single Channel Depth Images	Core Feature in OpenCL 2.0
cl_khr_device_enqueue_local_arg_types	Pass Non-Void Local Pointers to Child Kernels	Extension
cl_khr_device_uuid	Unique Device and Driver Identifier Queries	Extension
cl_khr_dx9_media_sharing	Share DirectX 9 Media Surfaces with OpenCL	Extension
cl_khr_egl_event	Share EGL Sync Objects with OpenCL	Extension
cl_khr_egl_image	Share EGL Images with OpenCL	Extension
cl_khr_extended_async_copies	2D and 3D Async Copies	Extension
cl_khr_extended_bit_ops	Bit Insert, Extract, and Reverse Operations	Extension

Extension Name and Link	Brief Description	Status
cl_khr_extended_versioning	Extend versioning of platform, devices, extensions, etc.	Core Feature in OpenCL 3.0 (with minor changes)
cl_khr_external_memory	Common Functionality for External Memory Sharing	Extension
cl_khr_external_memory_dma_buf	dma_buf External Memory Handles	Extension
cl_khr_external_memory_dx	Direct3D 11 and 12 External Memory Handles	Provisional Extension
cl_khr_external_memory_opaque_fd	Opaque File Descriptor External Memory Handles	Extension
cl_khr_external_memory_win32	NT Handle External Memory Handles	Extension
cl_khr_expect_assume	Kernel Optimization Hints	Extension
cl_khr_external_semaphore	Common Functionality for External Semaphore Sharing	Extension
cl_khr_external_semaphore_dx_fence	Direct3D 12 External Semaphore Handles	Provisional Extension
cl_khr_external_semaphore_opaque_fd	Opaque File Descriptor External Semaphore Handles	Extension
cl_khr_external_semaphore_sync_fd	Sync FD External Semaphore Handles	Extension
cl_khr_external_semaphore_win32	NT Handle External Semaphore Handles	Provisional Extension
cl_khr_fp16	Operations on 16-bit Floating-Point Values	Extension
cl_khr_fp64	Operations on 64-bit Floating-Point Values	Optional Core Feature in OpenCL 1.2
cl_khr_gl_depth_images	Share OpenGL Depth Images with OpenCL	Extension
cl_khr_gl_event	Share OpenGL Fence Sync Objects with OpenCL	Extension
cl_khr_gl_msaa_sharing	Share OpenGL MSAA Textures with OpenCL	Extension
cl_khr_gl_sharing	Sharing OpenGL Buffers and Textures with OpenCL	Extension

Extension Name and Link	Brief Description	Status
cl_khr_global_int32_base_atomics	Basic Atomic Operations on 32-bit Integers in Global Memory	Core Feature in OpenCL 1.1
cl_khr_global_int32_extended_atomics	Extended Atomic Operations on 32-bit Integers in Global Memory	Core Feature in OpenCL 1.1
cl_khr_icd	Installable Client Drivers	Extension
cl_khr_il_program	Support for Intermediate Language (IL) Programs (SPIR-V)	Core Feature in OpenCL 2.1
cl_khr_image2d_from_buffer	Create 2D Images from Buffers	Core Feature in OpenCL 2.0
cl_khr_initialize_memory	Initialize Local and Private Memory on Allocation	Extension
cl_khr_int64_base_atomics	Basic Atomic Operations on 64-bit Integers in Global and Local Memory	Extension
cl_khr_int64_extended_atomics	Extended Atomic Operations on 64-bit Integers in Global and Local Memory	Extension
cl_khr_local_int32_base_atomics	Basic Atomic Operations on 32-bit Integers in Local Memory	Core Feature in OpenCL 1.1
cl_khr_local_int32_extended_atomics	Extended Atomic Operations on 32-bit Integers in Local Memory	Core Feature in OpenCL 1.1
cl_khr_integer_dot_product	Integer dot product operations	Extension
cl_khr_kernel_clock	Sample Clock Values Within a Kernel	Extension
cl_khr_mipmap_image	Create and Use Images with Mipmaps	Extension
cl_khr_pci_bus_info	Query PCI Bus Information for an OpenCL Device	Extension
cl_khr_priority_hints	Create Command-Queues with Different Priorities	Extension
cl_khr_select_fprounding_mode	Set the Current Kernel Rounding Mode	DEPRECATED
cl_khr_semaphore	Semaphore Synchronization Primitives	Extension
cl_khr_spir	Standard Portable Intermediate Representation Programs	Extension, Superseded by IL Programs / SPIR-V

Extension Name and Link	Brief Description	Status
cl_khr_spirv_extended_debug_info	Allows Use of the SPIR-V OpenCL.DebugInfo.100 Extended Instruction Set	Extension
cl_khr_spirv_linkonce_odr	Allows Use of the SPIR-V SPV_KHR_linkonce_odr Extension	Extension
cl_khr_spirv_no_integer_wrap_decoration	Allows Use of the SPIR-V SPV_KHR_no_integer_wrap_decoration Extension	Extension
cl_khr_spirv_extended_debug_info	Allows Use of the SPIR-V OpenCL.DebugInfo.100 Extended Instruction Set	Extension
cl_khr_spirv_linkonce_odr	Allows Use of the SPIR-V SPV_KHR_linkonce_odr Extension	Extension
cl_khr_spirv_no_integer_wrap_decoration	Allows Use of the SPIR-V SPV_KHR_no_integer_wrap_decoration Extension	Extension
cl_khr_srgb_image_writes	Write to sRGB Images	Extension
cl_khr_subgroups	Sub-Groupings of Work Items	Core Feature in OpenCL 2.1 (with minor changes)
cl_khr_subgroup_ballot	Exchange Ballots Among Sub-Groupings of Work Items	Extension
cl_khr_subgroup_clustered_reduce	Clustered Reductions for Sub-Groupings of Work Items	Extension
cl_khr_subgroup_extended_types	Additional Type Support for Sub-group Functions	Extension
cl_khr_subgroup_named_barrier	Barriers for Subsets of a Work-group	Extension
cl_khr_subgroup_non_uniform_arithmetic	Sub-group Arithmetic Functions in Non-Uniform Control Flow	Extension
cl_khr_subgroup_non_uniform_vote	Hold Votes Among Sub-Groupings of Work Items	Extension
cl_khr_subgroup_rotate	Rotation Among Sub-Groupings of Work Items	Extension
cl_khr_subgroup_shuffle	General-Purpose Shuffles Among Sub-Groupings of Work Items	Extension
cl_khr_subgroup_shuffle_relative	Relative Shuffles Among Sub-Groupings of Work Items	Extension
cl_khr_suggested_local_work_size	Query a Suggested Local Work Size	Extension

Extension Name and Link	Brief Description	Status
cl_khr_terminate_context	Terminate an OpenCL Context	Extension
cl_khr_throttle_hints	Create Command-Queues with Different Throttle Policies	Extension
cl_khr_work_group_uniform_arithmetic	Work-group Uniform Arithmetic	Extension