

OpenCL (Open Computing Language) is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, GPUs, and other processors. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance compute servers, desktop computer systems, and handheld devices.

Specifications and more information about OpenCL and the OpenCL C++ Wrapper are available at www.khronos.org.

The OpenCL C++ Wrapper is designed to be built on top of the OpenCL 1.2 C API and is not a replacement. The C++ Wrapper API corresponds closely to the underlying C API and introduces no additional execution overhead.

- **[n.n.n]** refers to a section in the OpenCL C++ Wrapper API Specification
- **[n.n.n]** refers to a section in the OpenCL API Specification
- **type** refers to the types for indicated parameters
- **FunctionName** refers to non-member functions

cl::Platform [2.1]

Class cl::Platform is an interface to OpenCL cl_platform_id.

Platform();

Platform(const Platform &platform);

Platform(const cl_platform_id &platform);

cl_int getInfo(cl_platform_info name, STRING_CLASS *param) const;

name:

CL_PLATFORM_EXTENSIONS	STRING_CLASS
CL_PLATFORM_NAME	STRING_CLASS
CL_PLATFORM_PROFILE	STRING_CLASS
CL_PLATFORM_VENDOR	STRING_CLASS
CL_PLATFORM_VERSION	STRING_CLASS

Calls OpenCL function ::clGetPlatformInfo() [4.1]

template<cl_int name> type getInfo(cl_int *err = NULL) const;

name and type: See name for getInfo() above.

Calls OpenCL function ::clGetPlatformInfo() [4.1]

cl_int getDevices(cl_device_type type, VECTOR_CLASS<Device> *devices) const;

type: CL_DEVICE_TYPE_{ACCELERATOR, ALL, CPU}, CL_DEVICE_TYPE_{CUSTOM, DEFAULT, GPU}

Calls OpenCL function ::clGetDeviceIDs() [4.2]

static cl_int get(VECTOR_CLASS<cl::Platform> *platforms);

Calls OpenCL function ::clGetPlatformIDs() [4.1]

static cl_int get(Platform *platform);

Calls OpenCL function ::clGetPlatformIDs() [4.1]

static cl::Platform get(cl_int *errResult = NULL);

Calls OpenCL function ::clGetPlatformIDs() [4.1]

static cl::Platform getDefault(cl_int *errResult = NULL);

Calls OpenCL function ::clGetPlatformIDs() [4.1]

cl_int unloadCompiler();

Calls OpenCL function ::clUnloadPlatformCompiler() [5.6.6]

cl::Context [2.3]

Class cl::Context is an interface to OpenCL cl_context.

Context();

Context(const Context &context);

explicit Context(const cl_context &context);

Calls OpenCL function ::clCreateContext() [4.4]

Context(const VECTOR_CLASS<Device> &devices, cl_context_properties *properties = NULL, void (CL_CALLBACK *pfn_notify)(const char *errorInfo, const void *private_info_size, ::size_t cb, void *user_data) = NULL, void *user_data = NULL, cl_int *err = NULL);

properties:

NULL or CL_CONTEXT_PLATFORM, CL_CONTEXT_INTEROP_USER_SYNC

Calls OpenCL function ::clCreateContext() [4.4]

Context(const Device &device, cl_context_properties *properties = NULL, void (CL_CALLBACK *pfn_notify)(const char *errorInfo, const void *private_info_size, ::size_t cb, void *user_data) = NULL, void *user_data = NULL, cl_int *err = NULL);

properties: see properties for Context() above.

Calls OpenCL function ::clCreateContext() [4.4]

Context(cl_device_type type, cl_context_properties *properties = NULL, void (CL_CALLBACK *pfn_notify)(const char *errorInfo, const void *private_info_size, ::size_t cb, void *user_data) = NULL, void *user_data = NULL, cl_int *err = NULL);

properties: see properties for Context() above.

Calls OpenCL function ::clCreateContextFromType() [4.4]

static cl::Context getDefault(cl_int *err = NULL);

Calls OpenCL function ::clCreateContextFromType() [4.4]

cl::Device [2.2]

Class cl::Device is an interface to OpenCL cl_device_id.

Device();

Device(const Device &device);

Device(const cl_device_id &device);

static cl::Device getDefault(cl_int *err = NULL);

template<typename T> cl_int getInfo(cl_device_info name, T *param) const;

Calls OpenCL function ::clGetDeviceInfo() [4.2]

name: (where x is CL_DEVICE)

x_ADDRESS_BITS	cl_uint
x_AVAILABLE	cl_bool
x_BUILT_IN_KERNELS	STRING_CLASS
x_COMPILER_AVAILABLE	cl_bool
x_DOUBLE_FP_CONFIG	cl_device_fp_config
x_ENDIAN_LITTLE	cl_bool
x_ERROR_CORRECTION_SUPPORT	cl_bool
x_EXECUTION_CAPABILITIES	cl_device_exec_capabilities
x_EXTENSIONS	STRING_CLASS
x_GLOBAL_MEM_CACHE_SIZE	cl_ulong
x_GLOBAL_MEM_CACHE_TYPE	cl_device_mem_cache_type
x_GLOBAL_MEM_CACHESIZE	cl_uint
x_GLOBAL_MEM_SIZE	cl_ulong
x_HOST_UNIFIED_MEMORY	cl_bool
x_IMAGE_MAX_ARRAY_SIZE	size_t
x_IMAGE_MAX_BUFFER_SIZE	size_t
x_IMAGE_SUPPORT	cl_bool
x_IMAGE2D_MAX_{WIDTH, HEIGHT}	size_t
x_IMAGE3D_MAX_{WIDTH, HEIGHT, DEPTH}	size_t
x_LINKER_AVAILABLE	cl_bool
x_LOCAL_MEM_SIZE	cl_ulong
x_LOCAL_MEM_TYPE	cl_device_local_mem_type
x_MAX_CLOCK_FREQUENCY	cl_uint
x_MAX_COMPUTE_ARGS	cl_uint
x_MAX_COMPUTE_UNITS	cl_uint
x_MAX_CONSTANT_BUFFER_SIZE	cl_ulong
x_MAX_MEM_ALLOC_SIZE	cl_ulong
x_MAX_{READ, WRITE}_IMAGE_ARGS	cl_uint
x_MAX_PARAMETER_SIZE	::size_t
x_MAX_SAMPLERS	cl_uint
x_MAX_WORK_ITEM_DIMENSIONS	cl_uint
x_MAX_WORK_GROUP_SIZE	::size_t
x_MAX_WORK_ITEM_SIZES	VECTOR_CLASS::size_t
x_MEM_BASE_ADDR_ALIGN	cl_uint

template<typename T> cl_int getInfo(cl_context_info name, T *param) const;

name: (where x is CL_CONTEXT)

x_DEVICES	VECTOR_CLASS<cl::Device>
x_NUM_DEVICES	cl_uint
x_PROPERTIES	VECTOR_CLASS<cl_context_properties>
x_REFERENCE_COUNT	cl_uint

Calls OpenCL function ::clGetContextInfo() [4.4]

template<cl_int name> type getInfo(cl_int *err = NULL) const;

name and type: See name for getInfo() above.

Calls OpenCL function ::clGetContextInfo() [4.4]

cl_int getSupportedImageFormats(cl_mem_flags flags, cl_mem_object_type image_type, VECTOR_CLASS<ImageFormat> *formats) const;

flags:

CL_MEM_READ_WRITE, CL_MEM_{WRITE, READ}_ONLY,
CL_MEM_HOST_NO_ACCESS,
CL_MEM_HOST_{READ, WRITE}_ONLY,
CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

image_type:

CL_MEM_OBJECT_IMAGE1D, 2D, 3D, IMAGE1D_BUFFER,
CL_MEM_OBJECT_IMAGE1D_2D_ARRAY

Calls OpenCL function ::clGetSupportedImageFormats() [5.3.2]

cl_int setPrintfCallback(void (CL_CALLBACK *pfn_notify)(cl_context program, cl_uint printf_data_len, char *printf_data_ptr, void *user_data), void *user_data);

Calls OpenCL function ::clSetPrintfCallback()

x_NAME	STRING_CLASS
x_NATIVE_VECTOR_WIDTH_CHAR	cl_uint
x_NATIVE_VECTOR_WIDTH_INT	cl_uint
x_NATIVE_VECTOR_WIDTH_LONG	cl_uint
x_NATIVE_VECTOR_WIDTH_SHORT	cl_uint
x_NATIVE_VECTOR_WIDTH_DOUBLE	cl_uint
x_NATIVE_VECTOR_WIDTH_HALF	cl_uint
x_NATIVE_VECTOR_WIDTH_FLOAT	cl_uint
x_OPENCL_C_VERSION	STRING_CLASS
x_PARENT_DEVICE	cl_device_id
x_PARTITION_AFFINITY_DOMAIN	cl_device_affinity_domain
x_PARTITION_MAX_SUB_DEVICES	cl_uint
x_PARTITION_{PROPERTIES, TYPE}	VECTOR_CLASS<cl_device_partition_property>
x_PLATFORM	cl_platform_id
x_PREFERRED_INTEROP_USER_SYNC	cl_bool
x_PREFERRED_VECTOR_WIDTH_CHAR	cl_uint
x_PREFERRED_VECTOR_WIDTH_INT	cl_uint
x_PREFERRED_VECTOR_WIDTH_LONG	cl_uint
x_PREFERRED_VECTOR_WIDTH_SHORT	cl_uint
x_PREFERRED_VECTOR_WIDTH_DOUBLE	cl_uint
x_PREFERRED_VECTOR_WIDTH_HALF	cl_uint
x_PREFERRED_VECTOR_WIDTH_FLOAT	cl_uint
x_PRINTF_BUFFER_SIZE	size_t
x_PROFILE	STRING_CLASS
x_PROFILING_TIMER_RESOLUTION	::size_t
x_QUEUE_PROPERTIES	cl_command_queue_properties
x_REFERENCE_COUNT	cl_uint
x_SINGLE_FP_CONFIG	cl_device_fp_config
x_TYPE	cl_device_type
x_{VENDOR, VERSION}	STRING_CLASS
x_VENDOR_ID	cl_uint
CL_DRIVER_VERSION	STRING_CLASS

template<cl_int name> type getInfo(cl_int *errResult = NULL) const;

name and type: See name for getInfo() above.

Calls OpenCL function ::clGetDeviceInfo() [4.2]

cl_int createSubDevices(const cl_device_partition_property *properties, VECTOR_CLASS<Device> *devices);

Calls OpenCL function clCreateSubDevices() [4.3]

cl::Memory [3.1]

Class cl::Memory is an interface to OpenCL cl_mem.

cl::Memory



Memory();

Memory(const Memory &memory);

explicit Memory(const cl_mem &memory);

template<typename T> cl_int getInfo(cl_mem_info name, T *param) const;

name:

CL_MEM_ASSOCIATED_MEMOBJECT	cl_mem
CL_MEM_CONTEXT	cl::Context
CL_MEM_FLAGS	cl_mem_flags
CL_MEM_HOST_PTR	void*
CL_MEM_MAP_COUNT	cl_uint
CL_MEM_{OFFSET, SIZE}	::size_t
CL_MEM_REFERENCE_COUNT	cl_uint
CL_MEM_TYPE	cl_mem_object_type

Calls OpenCL function ::clGetMemObjectInfo() [5.4.5]

template<cl_int name> type getInfo(cl_int *err = NULL) const;

name and type: See name for getInfo() above.

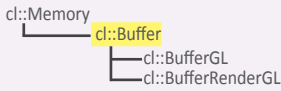
Calls OpenCL function ::clGetMemObjectInfo() [5.4.5]

cl_int setDestructorCallback(void (CL_CALLBACK *pfn_notify)(cl_mem memobj, void *user_data), void *user_data = NULL);

Calls OpenCL function ::clSetMemObjectDestructorCallback() [5.4.1]

cl::Buffer [3.2]

FunctionName in green refers to non-member functions.



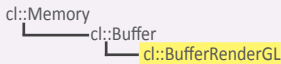
Buffer();
Buffer(const Buffer &buffer);
explicit Buffer(const cl_mem &buffer);
Calls OpenCL function ::clCreateBuffer() [5.2.1]
Buffer(const Context &context, cl_mem_flags flags, ::size_t size, void *host_ptr = NULL, cl_int *err = NULL);
flags: CL_MEM_READ_WRITE, CL_MEM_HOST_NO_ACCESS, CL_MEM_WRITE_ONLY, CL_MEM_HOST_READ_WRITE_ONLY, CL_MEM_USE_ALLOC_COPY, CL_MEM_HOST_PTR
Calls OpenCL function ::clCreateBuffer() [5.2.1]
Buffer(cl_mem_flags flags, ::size_t size, void *host_ptr = NULL, cl_int *err = NULL);

flags: See flags for Buffer() above.
Calls OpenCL function ::clCreateBuffer() [5.2.1]
 template<typename IteratorType>
Buffer(IteratorType startIterator, IteratorType endIterator, bool readOnly, bool useHostPtr = false, cl_int *err = NULL);
Calls OpenCL function ::clCreateBuffer() [5.2.1]

cl::Buffer createSubBuffer(cl_mem_flags flags, cl_buffer_create_type buffer_create_type, const void *buffer_create_info, cl_int *err = NULL);
flags: See flags for Buffer() above.
Calls OpenCL function ::clCreateSubBuffer() [5.2.1]
 template<typename IteratorType>inline cl_int **cl::copy**(IteratorType startIterator, IteratorType endIterator, cl::Buffer &buffer);
 template<typename IteratorType>inline cl_int **cl::copy**(cl::Buffer &buffer, IteratorType startIterator, IteratorType endIterator);

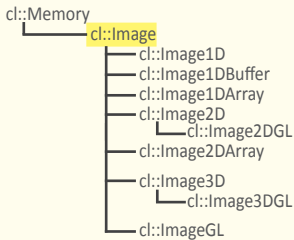
explicit BufferRenderGL(const cl_mem &buffer);
Calls OpenCL function ::clCreateFromGLRenderbuffer() [9.7.4]
BufferRenderGL(const Context &context, cl_mem_flags flags, GLuint bufobj, cl_int *err = NULL);
Calls OpenCL function ::clCreateFromGLRenderbuffer() [9.7.4]
 cl_int **getObjectInfo**(cl_gl_object_type *type, GLuint *gl_object_name);
Calls OpenCL function ::clGetGLObjectInfo() [9.7.5]

cl::BufferRenderGL [3.2.2]



BufferRenderGL();
BufferRenderGL(const BufferGL &buffer);

cl::Image [3.3]



Image();
Image(const Image &image);
explicit Image(const cl_mem &image);

template<typename T> cl_int **getImageInfo**(cl_image_info name, T *param) const;
name:
 CL_IMAGE_ARRAY_SIZE ::size_t
 CL_IMAGE_BUFFER cl_mem
 CL_IMAGE_DEPTH_ELEMENT_SIZE ::size_t
 CL_IMAGE_FORMAT cl_image_format
 CL_IMAGE_HEIGHT ::size_t
 CL_IMAGE_NUM_MIP_LEVELS_SAMPLES cl_uint
 CL_IMAGE_ROW_SLICE_PITCH ::size_t
 CL_IMAGE_WIDTH ::size_t
Calls OpenCL function ::clGetImageInfo() [5.3.6]
 template<cl_int name> **type getImageInfo**(cl_int *err = NULL) const;
name and type: See name for getImageInfo() above.
Calls OpenCL function ::clGetImageInfo() [5.3.6]

cl::Image1DArray [3.3.1]



Image1DArray();
Image1DArray(const Image1DArray &imageArray);

explicit Image1DArray(const cl_mem &imageArray);
Calls OpenCL function ::clCreateImage() [5.3.1]
Image1DArray(const Context &context, cl_mem_flags flags, ImageFormat format, ::size_t arraySize, ::size_t width, ::size_t rowPitch, void *host_ptr = NULL, cl_int *err = NULL);
flags: See flags for cl::Image1D.
Calls OpenCL function ::clCreateImage() [5.3.1]

cl::Image2DArray [3.3.2]



Image2DArray();
Image2DArray(const Image2DArray &imageArray);
explicit Image2DArray(const cl_mem &imageArray);
Calls OpenCL function ::clCreateImage() [5.3.1]

Image2DArray(const Context &context, cl_mem_flags flags, ImageFormat format, ::size_t arraySize, ::size_t width, ::size_t height, ::size_t rowPitch, ::size_t slicePitch, void *host_ptr = NULL, cl_int *err = NULL);
flags: See flags for cl::Image1D.
Calls OpenCL function ::clCreateImage() [5.3.1]

cl::Image2D [3.3.2]



Image2D();
Image2D(const Image2D &image2D);
explicit Image2D(const cl_mem &image2D);
Calls OpenCL function ::clCreateImage() [5.3.1]
Image2D(const Context &context, cl_mem_flags flags, ImageFormat format, ::size_t width, ::size_t height, ::size_t row_pitch = 0, void *host_ptr = NULL, cl_int *err = NULL);
flags: See flags for cl::Image1D.
Calls OpenCL function ::clCreateImage() [5.3.1]

cl::Image3D [3.3.3]

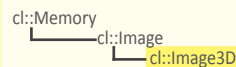


Image3D();
Image3D(const Image3D &image3D);
explicit Image3D(const cl_mem &image3D);
Calls OpenCL function ::clCreateImage() [5.3.1]
Image3D(const Context &context, cl_mem_flags flags, ImageFormat format, ::size_t width, ::size_t height, ::size_t depth, ::size_t row_pitch = 0, ::size_t slice_pitch = 0, void *host_ptr = NULL, cl_int *err = NULL);
flags: See flags for cl::Image1D.
Calls OpenCL function ::clCreateImage() [5.3.1]

cl::BufferGL [3.2.1]



BufferGL();
BufferGL(const BufferGL &buffer);
explicit BufferGL(const cl_mem &buffer);
Calls OpenCL function ::clCreateFromGLBuffer() [9.7.2]
BufferGL(const Context &context, cl_mem_flags flags, GLuint bufobj, cl_int *err = NULL);
Calls OpenCL function ::clCreateFromGLBuffer() [9.7.2]
 cl_int **getObjectInfo**(cl_gl_object_type *type, GLuint *gl_object_name);
Calls OpenCL function ::clGetGLObjectInfo() [9.7.5]

cl::ImageFormat

Struct ImageFormat is derived from OpenCL cl_image_format.
ImageFormat();

ImageFormat(cl_channel_order order, cl_channel_type type);

Built-in support:

CL_RGBA: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}

CL_BGRA: CL_UNORM_INT8

Optional support:

CL_R, CL_A: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}, CL_SNORM_INT{8,16}

CL_INTENSITY: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SNORM_INT{8,16}

CL_LUMINANCE: CL_UNORM_INT{8,16}, CL_HALF_FLOAT, CL_FLOAT, CL_SNORM_INT{8,16}

CL_RG, CL_RA: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}, CL_SNORM_INT{8,16}

CL_RGB: CL_UNORM_SHORT_{555,565}, CL_UNORM_INT_101010

CL_ARGB: CL_UNORM_INT8, CL_SIGNED_INT8, CL_UNSIGNED_INT8, CL_SNORM_INT8

CL_BGRA: CL_{SIGNED, UNSIGNED}_INT8, CL_SNORM_INT8

cl::ImageFormat::image_channel_order;

cl::ImageFormat::image_channel_data_type;

cl::Image1D [3.3.1]

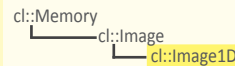


Image1D();
Image1D(const Image1D &image1D);
explicit Image1D(const cl_mem &image1D);
Calls OpenCL function ::clCreateImage() [5.3.1]

Image1D(const Context &context, cl_mem_flags flags, ImageFormat format, ::size_t width, void *host_ptr = NULL, cl_int *err = NULL);
flags: CL_MEM_READ_WRITE, CL_MEM_READ_WRITE_ONLY, CL_MEM_HOST_READ_WRITE_ONLY, CL_MEM_HOST_NO_ACCESS, CL_MEM_USE_ALLOC_COPY, CL_MEM_HOST_PTR
Calls OpenCL function ::clCreateImage() [5.3.1]

cl::Image1DBuffer [3.3.1]



Image1DBuffer();
Image1DBuffer(const Image1DBuffer &image1D);
explicit Image1DBuffer(const cl_mem &image1D);
Calls OpenCL function ::clCreateImage() [5.3.1]
Image1DBuffer(const Context &context, cl_mem_flags flags, ImageFormat format, ::size_t width, Buffer &buffer, void *host_ptr = NULL, cl_int *err = NULL);
flags: See flags for cl::Image1D.
Calls OpenCL function ::clCreateImage() [5.3.1]

Additional Image Classes on Next Page >

cl::ImageGL [3.3.4]



```

ImageGL();
ImageGL(const ImageGL &image);
explicit ImageGL(const cl_mem &image);
Calls OpenCL function ::clCreateFromGLTexture()

ImageGL(const Context &context, cl_mem_flags flags,
         GLenum target, GLint miplevel, GLuint texobj,
         cl_int *err = NULL);
Calls OpenCL function ::clCreateFromGLTexture() [9.7.3]
  
```

cl::Image2DGL



```

Image2DGL();
Image2DGL(const Image2DGL &image);
explicit Image2DGL(const cl_mem &image);

Image2DGL(const Context &context, cl_mem_flags flags,
          GLenum target, GLint miplevel, GLuint texobj,
          cl_int *err = NULL);
Calls OpenCL function ::clCreateFromGLTexture2D()
  
```

cl::Image3DGL



```

Image3DGL();
Image3DGL(const Image3DGL &image);
Image3DGL(const cl_mem &image);

Image3DGL(const Context &context, cl_mem_flags flags,
          GLenum target, GLint miplevel, GLuint texobj,
          cl_int *err = NULL);
Calls OpenCL function ::clCreateFromGLTexture3D()
  
```

cl::Program [3.5]

Class cl::Program is an interface to OpenCL cl_program.
FunctionName in green refers to non-member functions.

```

typedef VECTOR_CLASS<std::pair<const void*, ::size_t>>
  Binaries;

typedef VECTOR_CLASS<std::pair<const char*, ::size_t>>
  Sources;

Program();
Program(const Program &program);
explicit Program(const cl_program &program);

Program(const STRING_CLASS &source, cl_int *err =
  NULL);
Calls OpenCL function ::clCreateProgramWithSource() [5.6.1]
Calls OpenCL function ::clBuildProgram() [5.6.2]
  
```

```

Program(const STRING_CLASS &source,
        bool build, cl_int *err = NULL);
Calls OpenCL function ::clCreateProgramWithSource() [5.6.1]
Calls OpenCL function ::clBuildProgram() [5.6.2]
  
```

```

Program(const Context &context,
        const STRING_CLASS &source,
        bool build = false, cl_int *err = NULL);
Calls OpenCL function ::clCreateProgramWithSource() [5.6.1]
Calls OpenCL function ::clBuildProgram() [5.6.2]
  
```

```

Program(const Context &context,
        const Sources &sources, cl_int *err = NULL);
Calls OpenCL function ::clCreateProgramWithSource() [5.6.1]
  
```

```

Program(const Context &context,
        const VECTOR_CLASS<Device> &devices,
        const Binaries &binaries, VECTOR_CLASS<cl_int>
        *binaryStatus = NULL, cl_int *err = NULL);
Calls OpenCL function ::clCreateProgramWithBinary() [5.6.1]
  
```

```

Program(const Context &context,
        const VECTOR_CLASS<Device> &devices,
        const STRING_CLASS &kernelNames,
        cl_int *err = NULL);
Calls OpenCL function ::clCreateProgramWithBuiltInKernels() [5.6.1]
  
```

```

cl_int build(const VECTOR_CLASS<Device> &devices,
            const char *options = NULL, (CL_CALLBACK *pfn_notify)
            (cl_program, void *user_data) = NULL,
            void *data = NULL) const;
Calls OpenCL function ::clBuildProgram() [5.6.2]
  
```

cl::Sampler [3.4]

Class cl::Sampler is an interface to OpenCL cl_sampler.

```

Sampler();
Sampler(const Sampler &sampler);
Sampler(const cl_sampler &sampler);

Sampler(
  const Context &context, cl_bool normalized_coords,
  cl_addressing_mode addressing_mode,
  cl_filter_mode filter_mode, cl_int *err = NULL);
addressing_mode:
  CL_ADDRESS_CLAMP, CL_ADDRESS_CLAMP_TO_EDGE,
  CL_ADDRESS_NONE, CL_ADDRESS_MIRRORED_REPEAT,
  CL_ADDRESS_REPEAT
filter_mode: CL_FILTER_LINEAR, CL_FILTER_NEAREST
Calls OpenCL function ::clCreateSampler() [5.5]
  
```

```

cl_int build(const char *options = NULL,
            (CL_CALLBACK *pfn_notify) (cl_program,
            void *user_data) = NULL, void *data = NULL) const;
Calls OpenCL function ::clBuildProgram() [5.6.2]
  
```

```

cl_int compile(const char *options = NULL,
              void (CL_CALLBACK *notifyFptr)(cl_program,
              void *user_data) = NULL, void *data = NULL) const;
Calls OpenCL function ::clCompileProgram() [5.6.3]
  
```

```

inline cl::Program cl::linkProgram(
  Program input1, Program input2,
  const char *options = NULL,
  void (CL_CALLBACK *notifyFptr)
  (cl_program, void *user_data) = NULL,
  void *data = NULL, cl_int *err = NULL);
Calls OpenCL function ::clLinkProgram() [5.6.3]
  
```

```

inline cl::Program cl::linkProgram(
  VECTOR_CLASS<Program> inputPrograms,
  const char *options = NULL,
  void (CL_CALLBACK *notifyFptr)(cl_program,
  void *user_data) = NULL, void *data = NULL,
  cl_int *err = NULL);
Calls OpenCL function ::clLinkProgram() [5.6.3]
  
```

```

template <typename T> cl_int getInfo(
  cl_program_info name, T *param) const;
name:
  CL_PROGRAM_BINARIES          VECTOR_CLASS<char*>
  CL_PROGRAM_BINARY_SIZES     VECTOR_CLASS<::size_t>
  CL_PROGRAM_CONTEXT          cl::Context
  
```

```

template <typename T> cl_int getInfo(
  cl_sampler_info name, T *param);
name:
  CL_SAMPLER_ADDRESSING_MODE  cl_addressing_mode
  CL_SAMPLER_CONTEXT          cl::Context
  CL_SAMPLER_FILTER_MODE      cl_filter_mode
  CL_SAMPLER_NORMALIZED_COORDS cl_bool
  CL_SAMPLER_REFERENCE_COUNT  cl_uint
Calls OpenCL function ::clGetSampler() [5.5]
  
```

```

template<cl_int name> type getInfo(void);
name and type: See name for getInfo() above.
Calls OpenCL function ::clGetSampler() [5.5]
  
```

```

CL_PROGRAM_DEVICES          VECTOR_CLASS<cl_device_id>
CL_PROGRAM_KERNEL_NAMES     STRING_CLASS
CL_PROGRAM_NUM_DEVICES      cl_uint
CL_PROGRAM_NUM_KERNELS     ::size_t
CL_PROGRAM_REFERENCE_COUNT  cl_uint
CL_PROGRAM_SOURCE           STRING_CLASS
Calls OpenCL function ::clGetProgramInfo() [5.6.7]
  
```

```

template<cl_int name> type getInfo(
  cl_int *err = NULL) const;
name and type: See name for getInfo() above.
Calls OpenCL function ::clGetProgramInfo() [5.6.7]
  
```

```

template <> inline VECTOR_CLASS<char *>
  getInfo<cl_program_binaries>(cl_int *err) const;
Calls OpenCL function ::clGetProgramInfo() [5.6.7]
  
```

```

template <typename T> cl_int getBuildInfo(
  const Device &device, cl_program_build_info name,
  T *param) const;
name:
  CL_PROGRAM_BINARY_TYPE      cl_program_binary_type
  CL_PROGRAM_BUILD_{LOG, OPTIONS} STRING_CLASS
  CL_PROGRAM_BUILD_STATUS     cl_build_status
Calls OpenCL function ::clGetProgramBuildInfo() [5.6.7]
  
```

```

template<cl_int name> type getBuildInfo(
  const Device &device, cl_int *err = NULL) const;
name and type: See name for getBuildInfo() above.
Calls OpenCL function ::clGetProgramBuildInfo() [5.6.7]
  
```

```

cl_int createKernels(VECTOR_CLASS<Kernel> *kernels)
Calls OpenCL function ::clCreateKernelsInProgram() [5.7.1]
  
```

cl::Kernel [3.6]

Class cl::Kernel is an interface to OpenCL cl_kernel.

```

Kernel();
Kernel(const Kernel &kernel);
explicit Kernel(const cl_kernel &kernel);
inline Kernel(const Program &program, const char *name,
             cl_int *err = NULL);
Calls OpenCL function ::clCreateKernel() [5.7.1]

template <typename T> cl_int getInfo(
  cl_kernel_info name, T *param) const;
name:
  CL_KERNEL_ATTRIBUTES          STRING_CLASS
  CL_KERNEL_CONTEXT             cl::Context
  CL_KERNEL_FUNCTION_NAME      STRING_CLASS
  CL_KERNEL_NUM_ARGS           cl_uint
  CL_KERNEL_REFERENCE_COUNT    cl_uint
  CL_KERNEL_PROGRAM            cl::Program
Calls OpenCL function ::clGetKernelInfo() [5.7.2]
  
```

```

template<cl_int name> type getInfo(
  cl_int *err = NULL) const;
name and type: See name for getInfo() above.
Calls OpenCL function ::clGetKernelInfo() [5.7.2]
  
```

```

template <typename T> cl_int getArgInfo(cl_uint argIndex,
  cl_kernel_arg_info name, T *param) const;
name: (where x is CL_KERNEL_ARG)
  x_ADDRESS_QUALIFIER         cl_kernel_arg_address_qualifier
  x_ACCESS_QUALIFIER          cl_kernel_arg_access_qualifier
  x_ARG_NAME                   STRING_CLASS
  x_TYPE_QUALIFIER            cl_kernel_arg_type_qualifier
  x_ARG_TYPE_NAME             STRING_CLASS
Calls OpenCL function ::clGetKernelInfo() [5.7.2]
  
```

```

template<cl_int name> type getArgInfo(cl_uint argIndex,
  cl_int *err = NULL) const;
name and type: See name for getArgInfo() above.
Calls OpenCL function ::clGetKernelInfo() [5.7.2]
  
```

```

template <typename T> cl_int setArg(cl_uint index, T value);
Calls OpenCL function ::clSetKernelArg() [5.7.2]
  
```

```

cl_int setArg(cl_uint index, ::size_t size, void *argPtr);
Calls OpenCL function ::clSetKernelArg() [5.7.2]
  
```

For cl::Kernel::setArg, T is a compile-time argument that determines the type of a kernel argument being set. It can be one of the following:

- A cl::Memory object, e.g., a cl::Buffer, cl::Image3D, etc.
- A cl::Sampler object.
- A value of type cl::LocalSpaceArg, which corresponds to an argument of __local in the kernel object.
- A constant value that will be passed to the kernel.

The function cl::LocalSpaceArg cl::Local(::size_t) can be used to construct arguments specifying the size of a __local kernel argument. For example, cl::Local(100) would allocate sizeof(cl_char) * 100 of local memory.

PLEASE TRIM 3 INCHES OFF THIS SIDE

cl::Kernel (cont'd)

```
template <typename T> cl_int getWorkGroupInfo(
    const Device &device, cl_kernel_work_group_info
    name, T *param) const;
name: (where x is CL_KERNEL)
x_COMPILE_WORK_GROUP_SIZE           cl::size_t<3>
x_GLOBAL_WORK_SIZE                   cl::size_t<3>
x_LOCAL_MEM_SIZE                     cl_ulong
x_PREFERRED_WORK_GROUP_SIZE_MULTIPLE ::size_t
x_PRIVATE_MEM_SIZE                   cl_ulong
x_WORK_GROUP_SIZE                    ::size_t
Calls OpenCL function ::clGetKernelWorkGroupInfo() [5.7.2]
```

```
template<cl_int name> type getWorkGroupInfo(
    const Device &device, cl_int *err = NULL) const;
name and type: See name for getWorkGroupInfo() above.
Calls OpenCL function ::clGetKernelWorkGroupInfo() [5.7.2]
```

cl::Event [3.7]

Class cl::Event is an interface to OpenCL cl_event.

Event();**Event(const Event &event);****Event(const cl_event &event);**

```
template <typename T> cl_int getInfo(
    cl_event_info name, T *param) const;
```

name:

```
CL_EVENT_COMMAND_EXECUTION_STATUS  cl_int
CL_EVENT_COMMAND_QUEUE              cl::CommandQueue
CL_EVENT_COMMAND_TYPE                cl_command_type
CL_EVENT_CONTEXT                     cl::Context
CL_EVENT_REFERENCE_COUNT             cl_uint
Calls OpenCL function ::clGetEventInfo() [5.9]
```

```
template<cl_int name> type getInfo(
    cl_int *err = NULL) const;
```

name and type: See name for getInfo() above.

Calls OpenCL function ::clGetEventInfo() [5.9]

```
template <typename T> cl_int getProfilingInfo(
    cl_profiling_info name, T *param) const;
```

cl::UserEvent [3.8]**UserEvent();****UserEvent(const UserEvent &event);****UserEvent(Context &context, cl_int *err = NULL);**

Calls OpenCL function ::clCreateUserEvent() [5.9]

cl_int setStatus(cl_int status);

status: CL_COMPLETE or a negative integer error value.

Calls OpenCL function ::clSetUserEventStatus() [5.9]

name:

```
CL_PROFILING_COMMAND_END             cl_ulong
CL_PROFILING_COMMAND_QUEUED         cl_ulong
CL_PROFILING_COMMAND_START          cl_ulong
CL_PROFILING_COMMAND_SUBMIT         cl_ulong
Calls OpenCL function ::clGetEventProfilingInfo() [5.12]
```

```
template<cl_int name> type getProfilingInfo(
    cl_int *err = NULL) const;
```

name and type: See name for getProfilingInfo() above.

Calls OpenCL function ::clGetEventProfilingInfo() [5.12]

```
cl_int setCallback(cl_int type,
    void (CL_CALLBACK *pfn_notify)(cl_event event,
    cl_int command_exec_status,
    void *user_data), void *user_data = NULL);
type: CL_COMPLETE
Calls OpenCL function ::clSetEventCallback() [5.9]
```

cl_int wait(void) const;

Calls OpenCL function ::clWaitForEvents() [5.9]

```
static cl_int waitForEvents(
    const VECTOR_CLASS<Event> &events);
```

Calls OpenCL function ::clWaitForEvents() [5.9]

```
cl_int enqueueReadBuffer(
    const Buffer &buffer, cl_bool blocking_read,
    ::size_t offset, ::size_t size, void *ptr,
    const VECTOR_CLASS<Event> *events = NULL,
    Event *event = NULL) const;
```

Also available as an inline non-member function.

Calls OpenCL function ::clEnqueueReadBuffer() [5.2.2]

```
cl_int enqueueWriteBuffer(
    const Buffer &buffer,
    cl_bool blocking_write,
    ::size_t offset, ::size_t size, const void *ptr,
    const VECTOR_CLASS<Event> *events = NULL,
    Event *event = NULL) const;
```

Also available as an inline non-member function.

Calls OpenCL function ::clEnqueueWriteBuffer() [5.2.2]

```
cl_int enqueueReadBufferRect(
    const Buffer &buffer, cl_bool blocking_read,
    const size_t<3> &buffer_offset,
    const size_t<3> &host_offset,
    const size_t<3> &region, ::size_t buffer_row_pitch,
    ::size_t buffer_slice_pitch, ::size_t host_row_pitch,
    ::size_t host_slice_pitch, void *ptr,
    const VECTOR_CLASS<Event> *events = NULL,
    Event *event = NULL) const;
```

Calls OpenCL function ::clEnqueueReadBufferRect() [5.2.2]

```
cl_int enqueueWriteBufferRect(
    const Buffer &buffer, cl_bool blocking_write,
    const size_t<3> &buffer_offset,
    const size_t<3> &host_offset,
    const size_t<3> &region,
    ::size_t buffer_row_pitch,
    ::size_t buffer_slice_pitch,
    ::size_t host_row_pitch,
    ::size_t host_slice_pitch, void *ptr,
    const VECTOR_CLASS<Event> *events = NULL,
    Event *event = NULL) const;
```

Calls OpenCL function ::clEnqueueWriteBufferRect() [5.2.2]

```
cl_int enqueueCopyBuffer(
    const Buffer &src,
    const Buffer &dst,
    ::size_t src_offset,
    ::size_t dst_offset,
    ::size_t size,
    const VECTOR_CLASS<Event> *events = NULL,
    Event *event = NULL) const;
```

Also available as an inline non-member function.

Calls OpenCL function ::clEnqueueCopyBuffer() [5.2.2]

cl::CommandQueue [3.9]

Class cl::CommandQueue is an interface to OpenCL cl_command_queue.

CommandQueue();**CommandQueue(const CommandQueue &commandQueue);****CommandQueue(const cl_command_queue &commandQueue);**

```
CommandQueue(
    cl_command_queue_properties properties,
    cl_int *err = NULL);
```

```
properties: CL_QUEUE_PROFILING_ENABLE,
            CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE
Calls OpenCL function ::clCreateCommandQueue() [5.1]
```

```
CommandQueue(const Context &context,
    const Device &device,
    cl_command_queue_properties properties = 0,
    cl_int *err = NULL);
```

Calls OpenCL function ::clCreateCommandQueue() [5.1]

static CommandQueue getDefault(cl_int *err = NULL);

```
template <typename T> cl_int getInfo(
    cl_command_queue_info name, T *param) const;
```

name:

```
CL_QUEUE_CONTEXT                     cl::Context
CL_QUEUE_DEVICE                      cl::Device
CL_QUEUE_PROPERTIES                  cl_command_queue_properties
CL_QUEUE_REFERENCE_COUNT             cl_uint
Calls OpenCL function ::clGetCommandQueueInfo() [5.1]
```

```
template<cl_int name> type getInfo(
    cl_int *err = NULL) const;
```

name and type: See name for getInfo() above.

Calls OpenCL function ::clGetCommandQueueInfo() [5.1]

Continued on next page >

cl::CommandQueue (cont'd)

cl_int enqueueCopyBufferRect(

```
const Buffer &src,
const Buffer &dst,
const size_t<3> &src_origin,
const size_t<3> &dst_origin,
const size_t<3> &region,
::size_t src_row_pitch,
::size_t src_slice_pitch,
::size_t dst_row_pitch,
::size_t dst_slice_pitch,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL) const;
```

Calls OpenCL function ::clEnqueueCopyBufferRect() [5.2.2]

cl_int enqueueReadImage(

```
const Image &image,
cl_bool blocking_read,
const size_t<3> &origin,
const size_t<3> &region,
::size_t row_pitch,
::size_t slice_pitch,
void *ptr,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL) const;
```

Also available as an inline non-member function.

Calls OpenCL function ::clEnqueueReadImage() [5.3.3]

cl_int enqueueWriteImage(

```
const Image &image,
cl_bool blocking_write,
const size_t<3> &origin,
const size_t<3> &region,
::size_t row_pitch,
::size_t slice_pitch,
void *ptr,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL) const;
```

Also available as an inline non-member function.

Calls OpenCL function ::clEnqueueWriteImage() [5.3.3]

template<typename PatternType> cl_int

```
enqueueFillBuffer(
const Buffer &buffer, PatternType pattern,
::size_t offset, ::size_t size,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL) const;
```

Calls OpenCL function ::clEnqueueFillBuffer() [5.2.2]

cl_int enqueueFillImage(

```
const Image &image, U fillColor,
const size_t<3> &origin, const size_t<3> &region,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL) const;
```

U may be type cl_float4, cl_int4, or cl_uint4

Calls OpenCL function ::clEnqueueFillImage() [5.3.3]

cl_int enqueueCopyImage(

```
const Image &src_image,
const Image &dst_image, const size_t<3> &src_origin,
const size_t<3> &dst_origin, const size_t<3> &region,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL) const;
```

Also available as an inline non-member function.

Calls OpenCL function ::clEnqueueCopyImage() [5.3.3]

cl_int enqueueCopyImageToBuffer(

```
const Image &src, const Buffer &dst,
const size_t<3> &src_origin,
const size_t<3> &region, ::size_t dst_offset,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL) const;
```

Also available as an inline non-member function.

Calls OpenCL function ::clEnqueueCopyImageToBuffer() [5.3.4]

cl_int enqueueCopyBufferToImage(

```
const Buffer &src, const Image &dst, ::size_t src_offset,
const size_t<3> &dst_origin,
const size_t<3> &region,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL) const;
```

Also available as an inline non-member function.

Calls OpenCL function ::clEnqueueCopyBufferToImage() [5.3.4]

void *enqueueMapBuffer(

```
const Buffer &buffer,
cl_bool blocking_map, cl_map_flags flags,
::size_t offset, ::size_t size,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL, cl_int *err = NULL) const;
```

flags: See flags for enqueueMapImage() below.

Also available as an inline non-member function.

Calls OpenCL function ::clEnqueueMapBuffer() [5.2.3]

void *enqueueMapImage(

```
const Image &buffer,
cl_bool blocking_map, cl_map_flags flags,
const size_t<3> &origin, const size_t<3> &region,
::size_t *row_pitch, ::size_t *slice_pitch,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL, cl_int *err = NULL) const;
```

flags:

```
CL_MEM_READ_WRITE, CL_MEM_(WRITE, READ)_ONLY,
CL_MEM_HOST_NO_ACCESS,
CL_MEM_HOST_(READ, WRITE)_ONLY,
CL_MEM_(USE, ALLOC, COPY)_HOST_PTR
```

Calls OpenCL function ::clEnqueueMapImage() [5.3.5]

cl_int cl::enqueueUnmapMemObject(

```
const Memory &memory,
void *mapped_ptr,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL) const;
```

Also available as an inline non-member function.

Calls OpenCL function ::clEnqueueUnmapMemObject() [5.4.2]

cl_int enqueueNDRangeKernel(

```
const Kernel &kernel,
const NDRange &offset,
const NDRange &global,
const NDRange &local = NullRange,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL) const;
```

Calls OpenCL function ::clEnqueueNDRangeKernel() [5.8]

cl_int enqueueTask(

```
const Kernel &kernel,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL) const;
```

Calls OpenCL function ::clEnqueueTask() [5.8]

cl_int enqueueNativeKernel(

```
void (CL_CALLBACK *userFptr)(void *),
std::pair<void*, ::size_t> args,
const VECTOR_CLASS<Memory> *mem_objects =
NULL, const VECTOR_CLASS<const void*> *mem_locs =
NULL, const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL) const;
```

Calls OpenCL function ::clEnqueueNativeKernel() [5.8]

cl_int enqueueMigrateMemObjects(

```
const VECTOR_CLASS<Memory> &memObjects,
cl_mem_migration_flags flags,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL);
```

flags: See flags for enqueueMapImage()

Calls OpenCL function ::clEnqueueMigrateMemObjects() [5.4.4]

cl_int enqueueMarkerWithWaitList(

```
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL);
```

Calls OpenCL function ::clEnqueueMarkerWithWaitList() [5.10]

cl_int clEnqueueBarrierWithWaitList(

```
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL);
```

Calls OpenCL function ::clEnqueueMarkerWithWaitList() [5.10]

cl_int flush(void) const;

Also available as an inline non-member function.

Calls OpenCL function ::clFlush() [5.13]

cl_int finish(void) const;

Also available as an inline non-member function.

Calls OpenCL function ::clFinish() [5.13]

cl_int enqueueAcquireGLObjects(

```
const VECTOR_CLASS<Memory> *memObjects = NULL,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL) const;
```

Calls OpenCL function ::clEnqueueAcquireGLObjects() [9.7.6]

cl_int enqueueReleaseGLObjects(

```
const VECTOR_CLASS<Memory> *memObjects = NULL,
const VECTOR_CLASS<Event> *events = NULL,
Event *event = NULL) const;
```

Calls OpenCL function ::clEnqueueReleaseGLObjects() [9.7.6]

The following functions are also available as non-member functions, where they operate on the default command queue:

enqueueReadBuffer()	enqueueWriteBuffer(),
enqueueCopyBuffer()	enqueueReadImage(),
enqueueWriteImage()	enqueueCopyImage(),
enqueueCopyImageToBuffer()	enqueueMapBuffer(),
enqueueCopyBufferToImage()	flush()
enqueueUnmapMemObject()	finish()

PLEASE TRIM 3 INCHES OFF THIS SIDE

Functors

cl::EnqueueArgs

struct cl::EnqueueArgs is an interface for describing dispatch information.

- EnqueueArgs(NDRange *global*);
- EnqueueArgs(NDRange *global*, NDRange *local*);
- EnqueueArgs(NDRange *offset*, NDRange *global*, NDRange *local*);
- EnqueueArgs(CommandQueue *queue*, NDRange *global*);
- EnqueueArgs(CommandQueue *queue*, NDRange *global*, NDRange *local*);
- EnqueueArgs(CommandQueue *queue*, NDRange *offset*, NDRange *global*, NDRange *local*);

cl::KernelFunctorGlobal

class template<class... Args> KernelFunctorGlobal is an interface exporting a functor interface to OpenCL cl::Kernel.

- KernelFunctorGlobal(Kernel *kernel*, cl_int **err* = NULL);
- KernelFunctorGlobal(const Program& *program*, const STRING_CLASS *name*, cl_int **err* = NULL);
- Event operator() (const EnqueueArgs &*args*, Args... *actuals*);
- Event operator() (const EnqueueArgs &*args*, const Event &*waitEvent*, Args... *actuals*);

cl::make_kernel

struct template<class... Args> cl::make_kernel : public KernelFunctorGlobal<Args...> is an extended interface to OpenCL kernels.

make_kernel(const Program& *program*, const STRING_CLASS *name*, cl_int **err* = NULL);

template<class Args...> make_kernel(const Kernel *kernel*, cl_int **err* = NULL);

cl::NDRange [3.9]

- NDRange(::size_t *size0*);
- NDRange(::size_t *size0*, ::size_t *size1*);
- NDRange(::size_t *size0*, ::size_t *size1*, ::size_t *size2*);
- operator const ::size_t *() const;
- ::size_t dimensions(void) const;

EXCEPTIONS [4]

To enable the use of exceptions, define the preprocessor macro __CL_ENABLE_EXCEPTIONS. Once enabled, an error originally reported via a return value will be reported by throwing the exception class cl::Error. By default the method cl::Error::what() will return a const pointer to a string naming the C API call that reported the error.

Preprocessor macro names:

__BUILD_PROGRAM_ERR
__COMPILE_PROGRAM_ERR
__COPY_ERR
__CREATE_BUFFER_ERR
__CREATE_COMMAND_QUEUE_ERR
__CREATE_CONTEXT_FROM_TYPE_ERR
__CREATE_GL_BUFFER_ERR
__CREATE_GL_TEXTURE_ERR
__CREATE_IMAGE_ERR
__CREATE_KERNEL_ERR
__CREATE_KERNELS_IN_PROGRAM_ERR
__CREATE_PROGRAM_WITH_BUILT_IN_KERNELS_ERR
__CREATE_PROGRAM_WITH_SOURCE_ERR
__CREATE_PROGRAM_WITH_BINARY_ERR
__CREATE_SUBBUFFER_ERR
__CREATE_USER_EVENT_ERR
__ENQUEUE_COPY_BUFFER_TO_IMAGE_ERR

__ENQUEUE_COPY_BUFFER_ERR
__ENQUEUE_COPY_BUFFER_RECT_ERR
__ENQUEUE_COPY_IMAGE_ERR
__ENQUEUE_COPY_IMAGE_TO_BUFFER_ERR
__ENQUEUE_FILL_BUFFER_ERR
__ENQUEUE_FILL_IMAGE_ERR
__ENQUEUE_MAP_BUFFER_ERR
__ENQUEUE_MAP_IMAGE_ERR
__ENQUEUE_MIGRATE_MEM_OBJECTS_ERR
__ENQUEUE_NATIVE_KERNEL
__ENQUEUE_NDRANGE_KERNEL_ERR
__ENQUEUE_READ_BUFFER_ERR
__ENQUEUE_READ_BUFFER_RECT_ERR
__ENQUEUE_READ_IMAGE_ERR
__ENQUEUE_TASK_ERR
__ENQUEUE_UNMAP_MEM_OBJECT_ERR
__ENQUEUE_WRITE_BUFFER_ERR
__ENQUEUE_WRITE_BUFFER_RECT_ERR
__ENQUEUE_WRITE_IMAGE_ERR
__FLUSH_ERR
__FINISH_ERR
__GET_COMMAND_QUEUE_INFO_ERR
__GET_CONTEXT_INFO_ERR
__GET_DEVICE_INFO_ERR

__GET_DEVICE_IDS_ERR
__GET_EVENT_INFO_ERR
__GET_EVENT_PROFILE_INFO_ERR
__GET_IMAGE_INFO_ERR
__GET_KERNEL_ARG_INFO_ERR
__GET_KERNEL_INFO_ERR
__GET_KERNEL_WORK_GROUP_INFO_ERR
__GET_MEM_OBJECT_INFO_ERR
__GET_PLATFORM_INFO_ERR
__GET_PROGRAM_INFO_ERR
__GET_PROGRAM_BUILD_INFO_ERR
__GET_SAMPLER_INFO_ERR
__GET_SUPPORTED_IMAGE_FORMATS_ERR
__IMAGE_DIMENSION_ERR
__RELEASE_ERR
__RETAIN_ERR
__SET_COMMAND_QUEUE_PROPERTY_ERR
__SET_EVENT_CALLBACK_ERR
__SET_KERNEL_ARGS_ERR
__SET_MEM_OBJECT_DESTRUCTOR_CALLBACK_ERR
__SET_PRINTF_CALLBACK_ERR
__SET_USER_EVENT_STATUS_ERR
__UNLOAD_COMPILER_ERR
__WAIT_FOR_EVENTS_ERR

Supported Data Types

The optional double scalar and vector types are supported if CL_DEVICE_DOUBLE_FP_CONFIG is not zero.

OpenCL C++ Data Types

OpenCL C++ Data Type and Description	
cl::Error	exception object, derived from std::exception
cl::size_t<>	interface for static-sized arrays of size_t

ptrdiff_t	--	32- or 64-bit signed integer
intptr_t	--	32- or 64-bit signed integer
uintptr_t	--	32- or 64-bit unsigned integer
void	void	void

image3d_t	OPTIONAL	3D image handle
image2d_array_t	OPTIONAL	2D image array
image1d_t	OPTIONAL	1D image handle
image1d_buffer_t	OPTIONAL	1D image buffer
image1d_array_t	OPTIONAL	1D image array
sampler_t	OPTIONAL	sampler handle
event_t		event handle

Built-in OpenCL Vector Data Types [6.1.2]

OpenCL Type	API Type	Description
char _n	cl_char _n	8-bit signed
uchar _n	cl_uchar _n	8-bit unsigned
short _n	cl_short _n	16-bit signed
ushort _n	cl_ushort _n	16-bit unsigned
int _n	cl_int _n	32-bit signed
uint _n	cl_uint _n	32-bit unsigned
long _n	cl_long _n	64-bit signed
ulong _n	cl_ulong _n	64-bit unsigned
float _n	cl_float _n	32-bit float
double _n	OPTIONAL cl_double _n	64-bit float

OpenCL Reserved Data Types [6.1.4]

OpenCL Type	Description
bool _n	boolean vector
half _n	16-bit, vector
quad, quad _n	128-bit float, vector
complex half, complex half _n imaginary half, imaginary half _n	16-bit complex, vector
complex float, complex float _n imaginary float, imaginary float _n	32-bit complex, vector
complex double, complex double _n imaginary double, imaginary double _n	64-bit complex, vector
complex quad, complex quad _n imaginary quad, imaginary quad _n	128-bit complex, vector
float _n x _m	<i>n</i> * <i>m</i> matrix of 32-bit floats
double _n x _m	<i>n</i> * <i>m</i> matrix of 64-bit floats

Other Built-in OpenCL Data Types [6.1.3]

The optional types listed here other than event_t are only defined if CL_DEVICE_IMAGE_SUPPORT is CL_TRUE.

OpenCL Type	Description
image2d_t	OPTIONAL 2D image handle

Built-in OpenCL Scalar Data Types [6.1.1]

OpenCL Type	API Type	Description
bool	--	true (1) or false (0)
char	cl_char	8-bit signed
unsigned char, uchar	cl_uchar	8-bit unsigned
short	cl_short	16-bit signed
unsigned short, ushort	cl_ushort	16-bit unsigned
int	cl_int	32-bit signed
unsigned int, uint	cl_uint	32-bit unsigned
long	cl_long	64-bit signed
unsigned long, ulong	cl_ulong	64-bit unsigned
float	cl_float	32-bit float
double	OPTIONAL cl_double	64-bit. IEEE 754
half	cl_half	16-bit float (storage only)
size_t	--	32- or 64-bit unsigned integer



The Khronos Group is an industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics and dynamic media on a wide variety of platforms and devices. See www.khronos.org to learn more about the Khronos Group.

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