OpenCL SYCL API

Generated by Doxygen 1.8.7

Thu Jul 10 2014 10:56:50

Contents

Chapter 1

Main Page

This is a simple C++ sequential OpenCL SYCL C++ header file to experiment with the OpenCL CL provisional specification.For more information about OpenCL SYCL: http://www.khronos.org/opencl/sycl/

The aim of this file is mainly to define the interface of SYCL so that the specification documentation can be derived from it through tools like Doxygen or Sphinx. This explains why there are many functions and classes that are here only to do some forwarding in some inelegant way. This file is documentation driven and not implementation-style driven.

For more information on this project and to access to the source of this file, look at https://github.com/amd/triSYCL

The Doxygen version of the API in http://amd.github.io/triSYCL/Doxygen/SYCL/html and http://amd.github.io/triSYCL/Doxygen/SYCL/SYCL-API-refman.pdf

The Doxygen version of the implementation itself is in http://amd.github.io/triSYCL/Doxygen/triSYCL/html and http://amd.github.io/triSYCL/Doxygen/triSYCL/triSYCL-implementation-refman.pdf

Ronan.Keryell at AMD point com

Chapter 2

Todo List

Member __CL_ENABLE_EXCEPTIONS

Use a macro to check instead if the OpenCL header has been included before.

Namespace cl::sycl::access

This values should be normalized to allow separate compilation with different implementations?

Class cl::sycl::accessor< dataType, dimensions, mode, target >

Implement it for images according so section 3.3.4.5

Member cl::sycl::accessor< dataType, dimensions, mode, target >::dimensionality

in the specification: store the dimension for user request

Member cl::sycl::accessor< dataType, dimensions, mode, target >::element

in the specification: store the types for user request as STL

Member cl::sycl::accessor< dataType, dimensions, mode, target >::operator[] (id< dimensionality > Index) const

Implement the "const dataType &" version in the case the accessor is not for writing, as required by the specification

Member cl::sycl::accessor< dataType, dimensions, mode, target >::operator[] (size_t Index) const

This is not in the specification but looks like a cool common feature. Or solving it with an implicit constructor of id<1>?

Member cl::sycl::accessor< dataType, dimensions, mode, target >::operator[] (item< dimensionality > Index) const

Add in the specification because used by HPC-GPU slide 22

Class cl::sycl::buffer< T, dimensions >

there is a naming inconsistency in the specification between buffer and accessor on T versus datatype

Member cl::sycl::buffer< T, dimensions >::buffer (storage< T > &store, range< dimensions > r)

To be implemented

Member cl::sycl::buffer< T, dimensions >::buffer (const T ∗start_iterator, const T ∗end_iterator)

Add const to the SYCL specification

Member cl::sycl::buffer< T, dimensions >::buffer (buffer< T, dimensions > b, id< dimensions > base_index, range< dimensions > sub_range)

To be implemented

Update the specification to replace index by id

Member cl::sycl::buffer< T, dimensions >::buffer (cl_mem mem_object, queue from_queue, event available_event)

To be implemented

Improve the specification to allow CLHPP objects too

Member cl::sycl::buffer< T, dimensions >::element

Extension to SYCL specification: provide pieces of STL container interface?

Class cl::sycl::device

The implementation is quite minimal for now. :-)

Member cl::sycl::error_handler::default_handler

add this concept to the specification?

Member cl::sycl::error_handler::report_error (exception &error)=0

Add "virtual void" to the specification

Member cl::sycl::exception::get_buffer ()

Update specification to replace 0 by nullptr and add the templated buffer

to be implemented

Member cl::sycl::exception::get_cl_code ()

to be implemented

Member cl::sycl::exception::get_image ()

Update specification to replace 0 by nullptr and add the templated buffer

to be implemented

Member cl::sycl::exception::get_queue ()

Update specification to replace 0 by nullptr

Member cl::sycl::exception::get_sycl_code ()

to be implemented

use something else instead of cl_int to be usable without OpenCL

Class cl::sycl::gpu_selector

to be implemented

to be named device_selector::gpu instead in the specification?

Member cl::sycl::group< dims >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::group< dims >::get (int index)

add it to the specification?

is it supposed to be an int? A cl_int? a size_t?

Member cl::sycl::group< dims >::get_global_range ()

Update the specification to return a range<dims> instead of an id<>

Member cl::sycl::group< dims >::get_local_range ()

Update the specification to return a range<dims> instead of an id<>

Member cl::sycl::group< dims >::get_nr_range ()

Why the offset is not available here?

Also provide this access to the current nd_range

Member cl::sycl::group< dims >::group (const group &g)

in the specification, only provide a copy constructor. Any other constructors should be unspecified

Member cl::sycl::group< dims >::operator[] (int index)

add it to the specification?

is it supposed to be an int? A cl_int? a size_t?

Class cl::sycl::id< dims >

The definition of id and item seem completely broken in the current specification. The whole 3.4.1 is to be updated.

It would be nice to have [] working everywhere, provide both get_...() and get_...(int dim) equivalent to get_↩ ...()[int dim] Well it is already the case for item. So not needed for id? Indeed [] is mentioned in text of page 59 but not in class description.

Member cl::sycl::id< dims >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::id< dims >::get (int index)

is it supposed to be an int? A cl_int? a size_t?

Member cl::sycl::id< dims >::id (const range< dims > &r)

Is this necessary?

why in the specification id<int dims>(range<dims>global_size, range<dims> local_size) ?

Member cl::sycl::id< dims >::id (std::initializer_list< std::intptr_t > l)

Add this to the specification? Since it is said to be usable as a std::vector<>...

Member cl::sycl::id< dims >::id (std::intptr_t s)

Extension to the specification

Member cl::sycl::id< dims >::id ()

Add it to the specification?

Member cl::sycl::id< dims >::operator[] (int index)

explain in the specification (table 3.29, not only in the text) that [] works also for id, and why not range?

add also [] for range in the specification

is it supposed to be an int? A cl_int? a size_t?

Class cl::sycl::image< dimensions >

implement image

Class cl::sycl::item< dims >

Add to the specification: get_nd_range() to be coherent with providing get_local...() and get_global...() and what about the offset?

Member cl::sycl::item< dims >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::item< dims >::item (range< dims > global_size, range< dims > local_size)

what is the meaning of this constructor for a programmer?

Member cl::sycl::item< dims >::item (nd_range< dims > ndr)

a constructor from a nd_range too in the specification if the previous one has a meaning?

Member cl::sycl::kernel_lambda (Functor F)

This seems to have also the kernel_functor name in the specification

Class cl::sycl::nd_range< dims >

add copy constructors in the specification

Member cl::sycl::nd_range< dims >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::nd_range< dims >::get_offset ()

get_offset() is lacking in the specification

Member cl::sycl::parallel_for (Range r, Program p, ParallelForFunctor f)

deal with Program

Member cl::sycl::parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

Add an OpenMP implementation

Deal with incomplete work-groups

Implement with parallel_for_workgroup()/parallel_for_workitem()

Member cl::sycl::parallel_for (range< Dimensions > r, ParallelForFunctor f)

It is not clear if the ParallelForFunctor is called with an id<> or with an item. Let's use id<> when called with a range<> and item<> when called with a nd_range<>

Class cl::sycl::platform

triSYCL Implementation

Member cl::sycl::platform::get ()

Add cl.hpp version to the specification

Member cl::sycl::platform::get_info ()

It looks like in the specification the cl::detail:: is lacking to fit the cl.hpp version. Or is it to be redefined in SYCL too?

Member cl::sycl::platform::has_extension (const STRING_CLASS extension_name)

Should it be a param type instead of a STRING?

extend to any type of C++-string like object

Member cl::sycl::platform::platform (cl_platform_id platform id, const error_handler &handler=error_↩ handler::default_handler)

improve specification to accept also a cl.hpp object

Member cl::sycl::platform::platform (const error_handler &handler=error_handler::default_handler)

Add copy/move constructor to the implementation

Add const to the specification

Class cl::sycl::queue

The implementation is quite minimal for now. :-)

Class cl::sycl::range< dims >

use std::size_t dims instead of int dims in the specification?

add to the norm this default parameter value?

add to the norm some way to specify an offset?

Member cl::sycl::range< dims >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::range< dims >::get (int index)

explain in the specification (table 3.29, not only in the text) that [] works also for id, and why not range?

add also [] for range in the specification

is it supposed to be an int? A cl_int? a size_t?

Member cl::sycl::range< dims >::range (std::initializer_list< std::intptr_t > l)

This is not the same as the range(dim1,...) constructor from the specification

Member cl::sycl::single_task (std::function< void(void)> F)

remove from the SYCL specification and use a range-less parallel_for version with default construction of a 1-element range?

Member cl::sycl::storage< T >::element

Extension to SYCL specification: provide pieces of STL container interface?

Member cl::sycl::storage< T >::get_size ()=0

This is inconsistent in the specification with get_size() in buffer which returns the byte size. Is it to be renamed to get_count()?

Member STRING_CLASS

this should be more local, such as SYCL_STRING_CLASS or _SYCL_STRING_CLASS

use a typedef or a using instead of a macro?

implement __NO_STD_STRING

Table 3.2 in provisional specification is wrong: STRING_CLASS not at the right place

Member VECTOR_CLASS

this should be more local, such as SYCL_VECTOR_CLASS or _SYCL_VECTOR_CLASS

use a typedef or a using instead of a macro?

implement __NO_STD_VECTOR

Table 3.1 in provisional specification is wrong: VECTOR_CLASS not at the right place

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 6

Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 7

File Index

7.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 8

Module Documentation

8.1 Data access and storage in SYCL

Namespaces

- cl::sycl::access

  Describe the type of access by kernels.

Classes

- struct cl::sycl::accessor< dataType, dimensions, mode, target >

  The accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way. More...
- struct cl::sycl::storage< T >

  Abstract the way storage is managed to allow the programmer to control the storage management of buffers. More...
- struct cl::sycl::buffer< T, dimensions >

  A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on. More...

8.1.1 Detailed Description

8.1.2 Class Documentation

8.1.2.1 struct cl::sycl::accessor

template< typename dataType, size_t dimensions, access::mode mode, access::target target = access::global_buffer >struct cl↩ ::sycl::accessor< dataType, dimensions, mode, target >

The accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way.

Todo Implement it for images according so section 3.3.4.5

Public Types

- using element = dataType
- using **value_type** = dataType

Public Member Functions

- accessor (buffer< dataType, dimensions > &targetBuffer)

  Create an accessor to the given buffer.
- dataType & operator[] (id< dimensionality > Index) const

  Get the element specified by the given id.
- dataType & operator[] (size_t Index) const

  Get the element specified by the given index in the case we are mono-dimensional.
- dataType & operator[] (item< dimensionality > Index) const

  Get the element specified by the given item.

Static Public Attributes

- static const auto dimensionality = dimensions

8.1.2.1.1 Member Typedef Documentation

8.1.2.1.1.1 template<typename dataType , size_t dimensions, access::mode mode, access::target target = access::global_buffer> using cl::sycl::accessor< dataType, dimensions, mode, target >::element = dataType

Todo in the specification: store the types for user request as STL

8.1.2.1.2 Member Function Documentation

8.1.2.1.2.1 template<typename dataType , size_t dimensions, access::mode mode, access::target target = access::global_buffer> dataType& cl::sycl::accessor< dataType, dimensions, mode, target >::operator[] (id< dimensionality > *Index*) const [inline]

Get the element specified by the given id.

Todo Implement the "const dataType &" version in the case the accessor is not for writing, as required by the specification

8.1.2.1.2.2 template<typename dataType , size_t dimensions, access::mode mode, access::target target = access::global_buffer> dataType& cl::sycl::accessor< dataType, dimensions, mode, target >::operator[] (size_t *Index*) const [inline]

Get the element specified by the given index in the case we are mono-dimensional.

Todo This is not in the specification but looks like a cool common feature. Or solving it with an implicit constructor of id<1>?

8.1.2.1.2.3 template<typename dataType , size_t dimensions, access::mode mode, access::target target = access::global_buffer> dataType& cl::sycl::accessor< dataType, dimensions, mode, target >::operator[] (item< dimensionality > *Index*) const [inline]

Get the element specified by the given item.

Todo Add in the specification because used by HPC-GPU slide 22

8.1.2.1.3 Member Data Documentation

8.1.2.1.3.1 template<typename dataType , size_t dimensions, access::mode mode, access::target target = access::global_buffer> const auto cl::sycl::accessor< dataType, dimensions, mode, target >::dimensionality = dimensions [static]

Todo in the specification: store the dimension for user request

8.1.2.2 struct cl::sycl::storage

template<typename T>struct cl::sycl::storage< T >

Abstract the way storage is managed to allow the programmer to control the storage management of buffers.

Parameters

T	the type of the elements of the underlying data

The user is responsible for ensuring that their storage class implementation is thread-safe.

Public Types

- using element = T
- using **value_type** = T

Public Member Functions

- virtual size_t get_size ()=0

  Method called by SYCL system to get the number of elements of type T of the underlying data.
- virtual T ∗ get_host_data ()=0

  Method called by the SYCL system to know where that data is held in host memory.
- virtual const T ∗ get_initial_data ()=0

  Method called by the SYCL system at the point of construction to request the initial contents of the buffer.
- virtual T ∗ get_final_data ()=0

  Method called at the point of construction to request where the content of the buffer should be finally stored to.
- virtual void destroy ()=0

  Method called when the associated memory object is destroyed.
- virtual void in_use ()=0

  Method called when a command_group which accesses the data is added to a queue.
- virtual void completed ()=0

  Method called when the final enqueued command has completed.

8.1.2.2.1 Member Typedef Documentation

8.1.2.2.1.1 template<typename T> using cl::sycl::storage< T >::element = T

Todo Extension to SYCL specification: provide pieces of STL container interface?

8.1.2.2.2 Member Function Documentation

8.1.2.2.2.1 template<typename T> virtual void cl::sycl::storage< T >::destroy ( ) [pure virtual]

Method called when the associated memory object is destroyed.

This method is only called once, so if a memory object is copied multiple times, only when the last copy of the memory object is destroyed is the destroy method called.

Exceptions thrown by the destroy method will be caught and ignored.

8.1.2.2.2.2 template<typename T> virtual T∗ cl::sycl::storage< T >::get_final_data ( ) [pure virtual]

Method called at the point of construction to request where the content of the buffer should be finally stored to.

Returns

the address of where the buffer will be written to in host memory.

If the address is nullptr, then this phase is skipped.

If get_host_data() returns the same pointer as get_initial_data() and/or get_final_data() then the SYCL system should determine whether copying is actually necessary or not.

8.1.2.2.2.3 template<typename T> virtual T∗ cl::sycl::storage< T >::get_host_data ( ) [pure virtual]

Method called by the SYCL system to know where that data is held in host memory.

Returns

the address or nullptr if SYCL has to manage the temporary storage of the data.

8.1.2.2.2.4 template<typename T> virtual const T∗ cl::sycl::storage< T >::get_initial_data ( ) [pure virtual]

Method called by the SYCL system at the point of construction to request the initial contents of the buffer.

Returns

the address of the data to use or nullptr to skip this data initialization

8.1.2.2.2.5 template<typename T> virtual size_t cl::sycl::storage< T >::get_size ( ) [pure virtual]

Method called by SYCL system to get the number of elements of type T of the underlying data.

Todo This is inconsistent in the specification with get_size() in buffer which returns the byte size. Is it to be renamed to get_count()?

8.1.2.2.2.6 template<typename T> virtual void cl::sycl::storage< T >::in_use ( ) [pure virtual]

Method called when a command_group which accesses the data is added to a queue.

After completed is called, there may be further calls of in_use() if new work is enqueued that operates on the memory object.

8.1.2.3 struct cl::sycl::buffer

template<typename T, int dimensions>struct cl::sycl::buffer< T, dimensions >

A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on.

In the case we initialize it from a pointer, for now we just wrap the data with boost::multi_array_ref to provide the VLA semantics without any storage.

Todo there is a naming inconsistency in the specification between buffer and accessor on T versus datatype

Public Types

- using element = T
- using **value_type** = T

Public Member Functions

- buffer (const range< dimensions > &r)

  Create a new buffer with storage managed by SYCL.
- buffer (T ∗host_data, range< dimensions > r)

  Create a new buffer with associated host memory.
- buffer (const T ∗host_data, range< dimensions > r)

  Create a new read only buffer with associated host memory.
- buffer (storage< T > &store, range< dimensions > r)

  Create a new buffer from a storage abstraction provided by the user.
- buffer (const T ∗start_iterator, const T ∗end_iterator)

  Create a new allocated 1D buffer initialized from the given elements.
- buffer (buffer< T, dimensions > &b)

  Create a new buffer copy that shares the data with the origin buffer.
- buffer (buffer< T, dimensions > b, id< dimensions > base_index, range< dimensions > sub_range)

  Create a new sub-buffer without allocation to have separate accessors later.
- buffer (cl_mem mem_object, queue from_queue, event available_event)

  Create a buffer from an existing OpenCL memory object associated to a context after waiting for an event signaling the availability of the OpenCL data.
- template<access::mode mode, access::target target = access::global_buffer>
  accessor< T, dimensions, mode,
  target > get_access ()

  Get an accessor to the buffer with the required mode.

8.1.2.3.1 Member Typedef Documentation

8.1.2.3.1.1 template<typename T, int dimensions> using cl::sycl::buffer< T, dimensions >::element = T

Todo Extension to SYCL specification: provide pieces of STL container interface?

8.1.2.3.2 Constructor & Destructor Documentation

8.1.2.3.2.1 template<typename T, int dimensions> cl::sycl::buffer< T, dimensions >::buffer (const range< dimensions > & *r*) [inline]

Create a new buffer with storage managed by SYCL.

Parameters

r	defines the size

8.1.2.3.2.2 template<typename T, int dimensions> cl::sycl::buffer< T, dimensions >::buffer (T ∗ *host_data,* range< dimensions > *r*) [inline]

Create a new buffer with associated host memory.

Parameters

host_data	points to the storage and values used by the buffer
r	defines the size

8.1.2.3.2.3 template<typename T, int dimensions> cl::sycl::buffer< T, dimensions >::buffer (const T ∗ *host_data,* range< dimensions > *r*) [inline]

Create a new read only buffer with associated host memory.

Parameters

host_data	points to the storage and values used by the buffer
r	defines the size

8.1.2.3.2.4 template<typename T, int dimensions> cl::sycl::buffer< T, dimensions >::buffer (storage< T > & *store*, range< dimensions > *r*) [inline]

Create a new buffer from a storage abstraction provided by the user.

Parameters

store	is the storage back-end to use for the buffer
r	defines the size

The storage object has to exist during all the life of the buffer object.

Todo To be implemented

8.1.2.3.2.5 template<typename T, int dimensions> cl::sycl::buffer< T, dimensions >::buffer (const T * *start_iterator*, const T * *end_iterator*) [inline]

Create a new allocated 1D buffer initialized from the given elements.

Parameters

start_iterator	points to the first element to copy
end_iterator	points to just after the last element to copy

Todo Add const to the SYCL specification

8.1.2.3.2.6 template<typename T, int dimensions> cl::sycl::buffer< T, dimensions >::buffer (buffer< T, dimensions > & *b*) [inline]

Create a new buffer copy that shares the data with the origin buffer.

Parameters

b	is the buffer to copy from

The system use reference counting to deal with data lifetime

8.1.2.3.2.7 template<typename T, int dimensions> cl::sycl::buffer< T, dimensions >::buffer (buffer< T, dimensions > *b*, id< dimensions > *base_index*, range< dimensions > *sub_range*) [inline]

Create a new sub-buffer without allocation to have separate accessors later.

Parameters

b	is the buffer with the real data
base_index	specifies the origin of the sub-buffer inside the buffer b
sub_range	specifies the size of the sub-buffer

Todo To be implemented

Todo Update the specification to replace index by id

8.1.2.3.2.8 template<typename T, int dimensions> cl::sycl::buffer< T, dimensions >::buffer (cl_mem *mem_object*, queue *from_queue*, event *available_event*) [inline]

Create a buffer from an existing OpenCL memory object associated to a context after waiting for an event signaling the availability of the OpenCL data.

Parameters

mem_object	is the OpenCL memory object to use
from_queue	is the queue associated to the memory object
available_event	specifies the event to wait for if non null

Todo To be implemented

Todo Improve the specification to allow CLHPP objects too

8.1.2.3.3 Member Function Documentation

8.1.2.3.3.1 template<typename T, int dimensions> template<access::mode mode, access::target target = access::global_buffer> accessor<T, dimensions, mode, target> cl::sycl::buffer< T, dimensions >::get_access ( ) [inline]

Get an accessor to the buffer with the required mode.

Parameters

mode	is the requested access mode
target	is the type of object to be accessed

8.2 Expressing parallelism through kernels

Classes

- struct cl::sycl::range< dims >

  A SYCL range defines a multi-dimensional index range that can be used to launch parallel computation. More...
- struct cl::sycl::id< dims >

  Define a multi-dimensional index, used for example to locate a work item. More...
- struct cl::sycl::nd_range< dims >

  A ND-range, made by a global and local range, to specify work-group and work-item organization. More...
- struct cl::sycl::item< dims >

  A SYCL item stores information on a work-item within a work-group, with some more context such as the definition ranges. More...
- struct cl::sycl::group< dims >

  A group index used in a parallel_for_workitem to specify a work_group. More...

Functions

- template<typename KernelName , typename Functor >
  Functor cl::sycl::kernel_lambda (Functor F)

  kernel_lambda specify a kernel to be launch with a single_task or parallel_for
- void cl::sycl::single_task (std::function< void(void)> F)

  SYCL single_task launches a computation without parallelism at launch time.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for (range< Dimensions > r, ParallelForFunctor f)

  SYCL parallel_for launches a data parallel computation with parallelism specified at launch time by a range<>.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

  A variation of SYCL parallel_for to take into account a nd_range<>
- template<typename Range , typename Program , typename ParallelForFunctor >
  void cl::sycl::parallel_for (Range r, Program p, ParallelForFunctor f)

  SYCL parallel_for version that allows a Program object to be specified.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFunctor f)

  Loop on the work-groups.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_workitem (group< Dimensions > g, ParallelForFunctor f)

  Loop on the work-items inside a work-group.

8.2.1 Detailed Description

8.2.2 Class Documentation

8.2.2.1 struct cl::sycl::range

template<int dims = 1>struct cl::sycl::range< dims >

A SYCL range defines a multi-dimensional index range that can be used to launch parallel computation.

Todo use std::size_t dims instead of int dims in the specification?

Todo add to the norm this default parameter value?

Todo add to the norm some way to specify an offset?

Public Member Functions

- **range** (range< dims > &r)
- **range** (const range< dims > &r)
- range (std::initializer_list< std::intptr_t > l)

  Create a n-D range from a positive integer-like list.
- range (std::intptr_t x)

  To have implicit conversion from 1 integer.
- range (std::intptr_t x, std::intptr_t y)

  A 2-D constructor from 2 integers.
- range (std::intptr_t x, std::intptr_t y, std::intptr_t z)

  A 3-D constructor from 3 integers.
- int get (int index)

  Return the range size in the give dimension.

Static Public Attributes

- static const auto dimensionality = dims

8.2.2.1.1 Constructor & Destructor Documentation

8.2.2.1.1.1 template< int dims = 1> cl::sycl::range< dims >::range (std::initializer_list< std::intptr_t > *l*) [inline]

Create a n-D range from a positive integer-like list.

Todo This is not the same as the range(dim1,...) constructor from the specification

8.2.2.1.2 Member Function Documentation

8.2.2.1.2.1 template< int dims = 1> int cl::sycl::range< dims >::get (int *index*) [inline]

Return the range size in the give dimension.

Todo explain in the specification (table 3.29, not only in the text) that [] works also for id, and why not range?

Todo add also [] for range in the specification

Todo is it supposed to be an int? A cl_int? a size_t?

8.2.2.1.3 Member Data Documentation

8.2.2.1.3.1 template< int dims = 1> const auto cl::sycl::range< dims >::dimensionality = dims [static]

Todo add this Boost::multi_array or STL concept to the specification?

8.2.2.2 struct cl::sycl::id

template< int dims = 1> struct cl::sycl::id< dims >

Define a multi-dimensional index, used for example to locate a work item.

Todo The definition of id and item seem completely broken in the current specification. The whole 3.4.1 is to be updated.

Todo It would be nice to have [] working everywhere, provide both get_...() and get_...(int dim) equivalent to get_...()[int dim] Well it is already the case for item. So not needed for id? Indeed [] is mentioned in text of page 59 but not in class description.

Public Member Functions

- id ()

  Create a zero id.
- id (const id &init)

  Create an id with the same value of another one.
- id (const range< dims > &r)

  Create an id from a given range.
- id (std::initializer_list< std::intptr_t > l)

  Create a n-D range from a positive integer-like list.
- id (std::intptr_t s)

  To have implicit conversion from 1 integer.
- int get (int index)

  Return the id size in the given dimension.
- auto & operator[] (int index)

  Return the id size in the given dimension.

Static Public Attributes

- static const auto dimensionality = dims

8.2.2.2.1 Constructor & Destructor Documentation

8.2.2.2.1.1 template<int dims = 1> cl::sycl::id< dims >::id ( ) [inline]

Create a zero id.

Todo Add it to the specification?

8.2.2.2.1.2 template<int dims = 1> cl::sycl::id< dims >::id (const range< dims > & *r*) [inline]

Create an id from a given range.

Todo Is this necessary?

Todo why in the specification id<int dims>(range<dims>global_size, range<dims> local_size) ?

8.2.2.2.1.3 template<int dims = 1> cl::sycl::id< dims >::id (std::initializer_list< std::intptr_t > *l*) [inline]

Create a n-D range from a positive integer-like list.

Todo Add this to the specification? Since it is said to be usable as a std::vector<>...

8.2.2.2.1.4 template<int dims = 1> cl::sycl::id< dims >::id (std::intptr_t *s*) [inline]

To have implicit conversion from 1 integer.

Todo Extension to the specification

8.2.2.2.2 Member Function Documentation

8.2.2.2.2.1 template<int dims = 1> int cl::sycl::id< dims >::get (int *index*) [inline]

Return the id size in the given dimension.

Todo is it supposed to be an int? A cl_int? a size_t?

8.2.2.2.2.2 template<int dims = 1> auto& cl::sycl::id< dims >::operator[] (int *index*) [inline]

Return the id size in the given dimension.

Todo explain in the specification (table 3.29, not only in the text) that [] works also for id, and why not range?

Todo add also [] for range in the specification

Todo is it supposed to be an int? A cl_int? a size_t?

8.2.2.2.3 Member Data Documentation

8.2.2.2.3.1 template<int dims = 1> const auto cl::sycl::id< dims >::dimensionality = dims [static]

Todo add this Boost::multi_array or STL concept to the specification?

8.2.2.3 struct cl::sycl::nd_range

template<int dims = 1>struct cl::sycl::nd_range< dims >

A ND-range, made by a global and local range, to specify work-group and work-item organization.

The local offset is used to translate the iteration space origin if needed.

Todo add copy constructors in the specification

Public Member Functions

- nd_range (range< dims > global_size, range< dims > local_size, id< dims > offset=id< dims >())

  Construct a ND-range with all the details available in OpenCL.
- range< dims > get_global_range ()

  Get the global iteration space range.
- range< dims > get_local_range ()

  Get the local part of the iteration space range.
- range< dims > get_group_range ()

  Get the range of work-groups needed to run this ND-range.
- range< dims > get_offset ()

Static Public Attributes

- static const auto dimensionality = dims

8.2.2.3.1 Constructor & Destructor Documentation

8.2.2.3.1.1 template<int dims = 1> cl::sycl::nd_range< dims >::nd_range (range< dims > *global_size,* range< dims > *local_size,* id< dims > *offset* = id<dims>()) [inline]

Construct a ND-range with all the details available in OpenCL.

By default use a zero offset, that is iterations start at 0

8.2.2.3.2 Member Function Documentation

8.2.2.3.2.1 template<int dims = 1> range<dims> cl::sycl::nd_range< dims >::get_offset ( ) [inline]

Todo get_offset() is lacking in the specification

8.2.2.3.3 Member Data Documentation

8.2.2.3.3.1 template<int dims = 1> const auto cl::sycl::nd_range< dims >::dimensionality = dims [static]

Todo add this Boost::multi_array or STL concept to the specification?

8.2.2.4 struct cl::sycl::item

template<int dims = 1>struct cl::sycl::item< dims >

A SYCL item stores information on a work-item within a work-group, with some more context such as the definition ranges.

Todo Add to the specification: get_nd_range() to be coherent with providing get_local...() and get_global...() and what about the offset?

Public Member Functions

- item (range< dims > global_size, range< dims > local_size)

  Create an item from a local size and local size.
- item (nd_range< dims > ndr)
- int get_global (int dimension)

  Return the global coordinate in the given dimension.
- int get_local (int dimension)

  Return the local coordinate (that is in the work-group) in the given dimension.
- id< dims > get_global ()

  Get the whole global id coordinate.
- id< dims > get_local ()

  Get the whole local id coordinate (which is respective to the work-group)
- range< dims > get_global_range ()

  Get the global range where this item rely in.
- range< dims > get_local_range ()

  Get the local range (the dimension of the work-group) for this item.

Static Public Attributes

- static const auto dimensionality = dims

8.2.2.4.1 Constructor & Destructor Documentation

8.2.2.4.1.1 template<int dims = 1> cl::sycl::item< dims >::item (range< dims > *global_size*, range< dims > *local_size*) [inline]

Create an item from a local size and local size.

Todo what is the meaning of this constructor for a programmer?

8.2.2.4.1.2 template<int dims = 1> cl::sycl::item< dims >::item (nd_range< dims > *ndr*) [inline]

Todo a constructor from a nd_range too in the specification if the previous one has a meaning?

8.2.2.4.2 Member Data Documentation

8.2.2.4.2.1 template<int dims = 1> const auto **cl::sycl::item**< dims >::dimensionality = dims [static]

Todo add this Boost::multi_array or STL concept to the specification?

8.2.2.5 struct cl::sycl::group

template<int dims = 1>struct cl::sycl::group< dims >

A group index used in a parallel_for_workitem to specify a work_group.

Public Member Functions

- group (const group &g)
- id< dims > **get_group_id** ()
- range< dims > get_local_range ()

  Get the local range for this work_group.
- range< dims > get_global_range ()

  Get the local range for this work_group.
- nd_range< dims > get_nr_range ()
- int get (int index)

  Return the group coordinate in the given dimension.
- auto & operator[] (int index)

  Return the group coordinate in the given dimension.

Static Public Attributes

- static const auto dimensionality = dims

8.2.2.5.1 Constructor & Destructor Documentation

8.2.2.5.1.1 template<int dims = 1> **cl::sycl::group**< dims >::**group** (const **group**< dims > & *g*) [inline]

Todo in the specification, only provide a copy constructor. Any other constructors should be unspecified

8.2.2.5.2 Member Function Documentation

8.2.2.5.2.1 template<int dims = 1> int **cl::sycl::group**< dims >::get (int *index*) [inline]

Return the group coordinate in the given dimension.

Todo add it to the specification?

Todo is it supposed to be an int? A cl_int? a size_t?

8.2.2.5.2.2 template<int dims = 1> **range**<dims> **cl::sycl::group**< dims >::get_global_range ( ) [inline]

Get the local range for this work_group.

Todo Update the specification to return a range<dims> instead of an id<>

8.2.2.5.2.3 template<int dims = 1> range<dims> cl::sycl::group< dims >::get_local_range ( ) [inline]

Get the local range for this work_group.

Todo Update the specification to return a range<dims> instead of an id<>

8.2.2.5.2.4 template<int dims = 1> nd_range<dims> cl::sycl::group< dims >::get_nr_range ( ) [inline]

Todo Why the offset is not available here?

Todo Also provide this access to the current nd_range

8.2.2.5.2.5 template<int dims = 1> auto& cl::sycl::group< dims >::operator[] (int *index*) [inline]

Return the group coordinate in the given dimension.

Todo add it to the specification?

Todo is it supposed to be an int? A cl_int? a size_t?

8.2.2.5.3 Member Data Documentation

8.2.2.5.3.1 template<int dims = 1> const auto cl::sycl::group< dims >::dimensionality = dims [static]

Todo add this Boost::multi_array or STL concept to the specification?

8.2.3 Function Documentation

8.2.3.1 template<typename KernelName , typename Functor > Functor cl::sycl::kernel_lambda (Functor *F*)

kernel_lambda specify a kernel to be launch with a single_task or parallel_for

Todo This seems to have also the kernel_functor name in the specification

8.2.3.2 template<int Dimensions = 1, typename ParallelForFunctor > void cl::sycl::parallel_for (range< Dimensions > *r*, ParallelForFunctor *f*)

SYCL parallel_for launches a data parallel computation with parallelism specified at launch time by a range<>.

This implementation use OpenMP 3 if compiled with the right flag.

Todo It is not clear if the ParallelForFunctor is called with an id<> or with an item. Let's use id<> when called with a range<> and item<> when called with a nd_range<>

8.2.3.3 template<int Dimensions = 1, typename ParallelForFunctor > void cl::sycl::parallel_for (nd_range< Dimensions > *r*, ParallelForFunctor *f*)

A variation of SYCL parallel_for to take into account a nd_range<>

Todo Add an OpenMP implementation

Todo Deal with incomplete work-groups

Todo Implement with parallel_for_workgroup()/parallel_for_workitem()

8.2.3.4 template< typename Range , typename Program , typename ParallelForFunctor > void cl::sycl::parallel_for (Range *r,* Program *p,* ParallelForFunctor *f*)

SYCL parallel_for version that allows a Program object to be specified.

Todo deal with Program

8.2.3.5 void cl::sycl::single_task (std::function< void(void)> *F*)

SYCL single_task launches a computation without parallelism at launch time.

Right now the implementation does nothing else that forwarding the execution of the given functor

Todo remove from the SYCL specification and use a range-less parallel_for version with default construction of a 1-element range?

8.3 Error handling

Classes

- struct cl::sycl::exception

  Encapsulate a SYCL error information. More...
- struct cl::sycl::error_handler

  User supplied error handler to call a user-provided function when an error happens from a SYCL object that was constructed with this error handler. More...

8.3.1 Detailed Description

8.3.2 Class Documentation

8.3.2.1 struct cl::sycl::exception

Encapsulate a SYCL error information.

Public Member Functions

- cl_int get_cl_code ()

  Get the OpenCL error code.
- cl_int get_sycl_code ()

  Get the SYCL-specific error code.
- queue ∗ get_queue ()

  Get the queue that caused the error.
- template<typename T , int dimensions>
  buffer< T, dimensions > ∗ get_buffer ()

  Get the buffer that caused the error.
- template<int dimensions>
  image< dimensions > ∗ get_image ()

  Get the image that caused the error.

8.3.2.1.1 Member Function Documentation

8.3.2.1.1.1 template<typename T , int dimensions> buffer<T, dimensions>∗ cl::sycl::exception::get_buffer ( ) [inline]

Get the buffer that caused the error.

Returns

nullptr if not a buffer error

Todo Update specification to replace 0 by nullptr and add the templated buffer

Todo to be implemented

8.3.2.1.1.2 cl_int cl::sycl::exception::get_cl_code ( ) [inline]

Get the OpenCL error code.

Returns

0 if not an OpenCL error

Todo to be implemented

8.3.2.1.1.3 template<int dimensions> image<dimensions>∗ cl::sycl::exception::get_image ( ) [inline]

Get the image that caused the error.

Returns

nullptr if not a image error

Todo Update specification to replace 0 by nullptr and add the templated buffer

Todo to be implemented

8.3.2.1.1.4 queue∗ cl::sycl::exception::get_queue ( ) [inline]

Get the queue that caused the error.

Returns

nullptr if not a queue error

Todo Update specification to replace 0 by nullptr

8.3.2.1.1.5 cl_int cl::sycl::exception::get_sycl_code ( ) [inline]

Get the SYCL-specific error code.

Returns

0 if not a SYCL-specific error

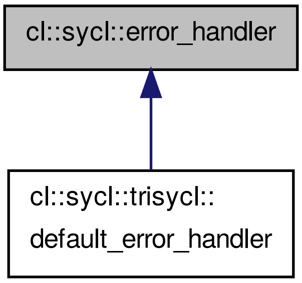
Todo to be implemented

Todo use something else instead of cl_int to be usable without OpenCL

8.3.2.2 struct cl::sycl::error_handler

User supplied error handler to call a user-provided function when an error happens from a SYCL object that was constructed with this error handler.

Inheritance diagram for cl::sycl::error_handler:



Public Member Functions

- virtual void report_error (exception &error)=0

  The method to define to be called in the case of an error.

Static Public Attributes

- static
  trisycl::default_error_handler default_handler

  Add a default_handler to be used by default.

8.3.2.2.1 Member Function Documentation

8.3.2.2.1.1 virtual void cl::sycl::error_handler::report_error (**exception** & *error*) `[pure virtual]`

The method to define to be called in the case of an error.

Todo Add "virtual void" to the specification

Implemented in cl::sycl::trisycl::default_error_handler.

8.3.2.2.2 Member Data Documentation

8.3.2.2.2.1 **trisycl::default_error_handler** cl::sycl::error_handler::default_handler `[static]`

Add a default_handler to be used by default.

Todo add this concept to the specification?

8.4 Platforms, contexts, devices and queues

Classes

- struct cl::sycl::device

  SYCL device. More...
- struct cl::sycl::device_selector

  The SYCL heuristics to select a device. More...
- struct cl::sycl::gpu_selector

  Select the best GPU, if any. More...
- struct cl::sycl::context

  SYCL context. More...
- struct cl::sycl::queue

  SYCL queue, similar to the OpenCL queue concept. More...
- struct cl::sycl::platform

  Abstract the OpenCL platform. More...
- struct cl::sycl::command_group

  SYCL command group gather all the commands needed to execute one or more kernels in a kind of atomic way. More...

8.4.1 Detailed Description

8.4.2 Class Documentation

8.4.2.1 struct cl::sycl::device

SYCL device.

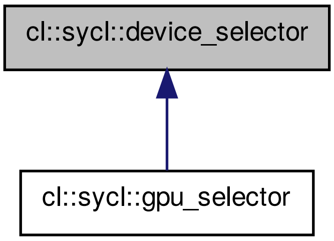
Todo The implementation is quite minimal for now. :-)

8.4.2.2 struct cl::sycl::device_selector

The SYCL heuristics to select a device.

The device with the highest score is selected

Inheritance diagram for cl::sycl::device_selector:



Public Member Functions

- virtual int **operator()** (device dev)=0

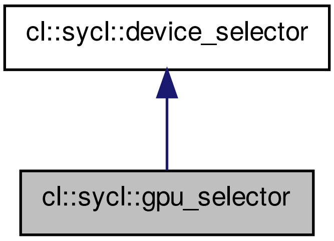
8.4.2.3 struct cl::sycl::gpu_selector

Select the best GPU, if any.

Todo to be implemented

Todo to be named device_selector::gpu instead in the specification?

Inheritance diagram for cl::sycl::gpu_selector:



Public Member Functions

- int **operator()** (device dev) override

8.4.2.4 struct cl::sycl::context

SYCL context.

The implementation is quite minimal for now. :-)

Public Member Functions

- **context** (gpu_selector s)
- **context** (device_selector &s)

8.4.2.5 struct cl::sycl::queue

SYCL queue, similar to the OpenCL queue concept.

Todo The implementation is quite minimal for now. :-)

Public Member Functions

- **queue** (context c)

8.4.2.6 struct cl::sycl::platform

Abstract the OpenCL platform.

Todo triSYCL Implementation

Public Member Functions

- platform (const error_handler &handler=error_handler::default_handler)

  Construct a default platform and provide an optional error_handler to deals with errors.
- platform (cl_platform_id platform id, const error_handler &handler=error_handler::default_handler)

  Create a SYCL platform from an existing OpenCL one and provide an optional error_handler to deals with errors.
- platform (cl_platform_id platform id, int &error_code)

  Create a SYCL platform from an existing OpenCL one and provide an integer place-holder to return the OpenCL error code, if any.
- ∼platform ()

  Destructor of the SYCL abstraction.
- cl_platform_id get ()

  Get the OpenCL platform_id underneath.
- template<cl_int name>
  cl::detail::param_traits
  < cl_platform_info, name >
  ::param_type get_info ()

  Get the OpenCL information about the requested parameter.
- bool is_host ()

  Test if this platform is a host platform.
- bool has_extension (const STRING_CLASS extension_name)

  Test if an extension is available on the platform.

Static Public Member Functions

- static VECTOR_CLASS< platform > get_platforms ()

  Get the list of all the platforms available to the application.
- static VECTOR_CLASS< device > get_devices (cl_device_type device_type=CL_DEVICE_TYPE_ALL)

  Get all the devices of a given type available to the application.

8.4.2.6.1 Constructor & Destructor Documentation

8.4.2.6.1.1 cl::sycl::platform::platform (const error_handler & *handler* = error_handler::default_handler) [inline]

Construct a default platform and provide an optional error_handler to deals with errors.

Todo Add copy/move constructor to the implementation

Todo Add const to the specification

8.4.2.6.1.2 cl::sycl::platform::platform (cl_platform_id **platform** *id,* const **error_handler** & *handler =* **error_handler::default_handler**) [inline]

Create a SYCL platform from an existing OpenCL one and provide an optional error_handler to deals with errors.

Todo improve specification to accept also a cl.hpp object

8.4.2.6.2 Member Function Documentation

8.4.2.6.2.1 cl_platform_id cl::sycl::platform::get ( ) [inline]

Get the OpenCL platform_id underneath.

Todo Add cl.hpp version to the specification

8.4.2.6.2.2 static VECTOR_CLASS<device> cl::sycl::platform::get_devices (cl_device_type *device_type =* CL_DEVICE_TYPE_ALL) [inline], [static]

Get all the devices of a given type available to the application.

By default returns all the devices.

8.4.2.6.2.3 template<cl_int name> cl::detail::param_traits<cl_platform_info, name>::param_type cl::sycl::platform::get_info ( ) [inline]

Get the OpenCL information about the requested parameter.

Todo It looks like in the specification the cl::detail:: is lacking to fit the cl.hpp version. Or is it to be redefined in SYCL too?

8.4.2.6.2.4 bool cl::sycl::platform::has_extension (const STRING_CLASS *extension_name*) [inline]

Test if an extension is available on the platform.

Todo Should it be a param type instead of a STRING?

Todo extend to any type of C++-string like object

8.4.2.7 struct cl::sycl::command_group

SYCL command group gather all the commands needed to execute one or more kernels in a kind of atomic way.

Since all the parameters are captured at command group creation, one can execute the content in an asynchronous way and delayed schedule.

For now just execute the command group directly.

Public Member Functions

- template<typename Functor >
  command_group (queue Q, Functor F)

Chapter 9

Namespace Documentation

9.1 cl Namespace Reference

SYCL dwells in the cl::sycl namespace.

9.1.1 Detailed Description

SYCL dwells in the cl::sycl namespace.

9.2 cl::sycl::access Namespace Reference

Describe the type of access by kernels.

Enumerations

- enum mode {
  read = 42, **write**, **atomic**, **read_write**,
  discard_read_write }

  This describes the type of the access mode to be used via accessor.
- enum target {
  global_buffer = 2014, **constant_buffer**, **local**, **image**,
  host_buffer, **host_image**, **image_array**, **cl_buffer**,
  cl_image }

  The target enumeration describes the type of object to be accessed via the accessor.

9.2.1 Detailed Description

Describe the type of access by kernels.

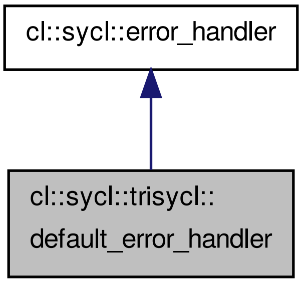
Todo This values should be normalized to allow separate compilation with different implementations?

Chapter 10

Class Documentation

10.1 cl::sycl::trisycl::default_error_handler Struct Reference

Inheritance diagram for cl::sycl::trisycl::default_error_handler:



Public Member Functions

- void report_error (exception &error) override

  The method to define to be called in the case of an error.

Additional Inherited Members

10.1.1 Member Function Documentation

10.1.1.1 void cl::sycl::trisycl::default_error_handler::report_error (exception & *error*) [inline], [override], [virtual]

The method to define to be called in the case of an error.

Todo Add "virtual void" to the specification

Implements cl::sycl::error_handler.

The documentation for this struct was generated from the following file:

- include/CL/sycl.hpp

10.2 cl::sycl::image< dimensions > Struct Template Reference

10.2.1 Detailed Description

template<int dimensions>struct cl::sycl::image< dimensions >

Todo implement image

The documentation for this struct was generated from the following file:

- include/CL/sycl.hpp

Chapter 11

File Documentation

11.1 include/CL/sycl.hpp File Reference

Classes

- struct cl::sycl::range< dims >

  A SYCL range defines a multi-dimensional index range that can be used to launch parallel computation. More...
- struct cl::sycl::id< dims >

  Define a multi-dimensional index, used for example to locate a work item. More...
- struct cl::sycl::nd_range< dims >

  A ND-range, made by a global and local range, to specify work-group and work-item organization. More...
- struct cl::sycl::item< dims >

  A SYCL item stores information on a work-item within a work-group, with some more context such as the definition ranges. More...
- struct cl::sycl::group< dims >

  A group index used in a parallel_for_workitem to specify a work_group. More...
- struct cl::sycl::buffer< T, dimensions >

  A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on. More...
- struct cl::sycl::image< dimensions >
- struct cl::sycl::exception

  Encapsulate a SYCL error information. More...
- struct cl::sycl::error_handler

  User supplied error handler to call a user-provided function when an error happens from a SYCL object that was constructed with this error handler. More...
- struct cl::sycl::trisycl::default_error_handler
- struct cl::sycl::device

  SYCL device. More...
- struct cl::sycl::device_selector

  The SYCL heuristics to select a device. More...
- struct cl::sycl::gpu_selector

  Select the best GPU, if any. More...
- struct cl::sycl::context

  SYCL context. More...
- struct cl::sycl::queue

  SYCL queue, similar to the OpenCL queue concept. More...
- struct cl::sycl::platform

  Abstract the OpenCL platform. More...

- struct cl::sycl::command_group

  SYCL command group gather all the commands needed to execute one or more kernels in a kind of atomic way. More...
- struct cl::sycl::accessor< dataType, dimensions, mode, target >

  The accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way. More...
- struct cl::sycl::storage< T >

  Abstract the way storage is managed to allow the programmer to control the storage management of buffers. More...
- struct cl::sycl::buffer< T, dimensions >

  A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on. More...

Namespaces

- cl

  SYCL dwells in the cl::sycl namespace.
- cl::sycl::access

  Describe the type of access by kernels.

Macros

- #define **TRISYCL_IMPL**(...)
- #define __CL_ENABLE_EXCEPTIONS

  Define TRISYCL_OPENCL to add OpenCL.
- #define VECTOR_CLASS std::vector

  The vector type to be used as SYCL vector.
- #define STRING_CLASS std::string

  The string type to be used as SYCL string.

Enumerations

- enum cl::sycl::access::mode {
  read = 42, **write**, **atomic**, **read_write**,
  discard_read_write }

  This describes the type of the access mode to be used via accessor.
- enum cl::sycl::access::target {
  global_buffer = 2014, **constant_buffer**, **local**, **image**,
  host_buffer, **host_image**, **image_array**, **cl_buffer**,
  cl_image }

  The target enumeration describes the type of object to be accessed via the accessor.

Functions

- template<typename KernelName , typename Functor >
  Functor cl::sycl::kernel_lambda (Functor F)

  kernel_lambda specify a kernel to be launch with a single_task or parallel_for
- void cl::sycl::single_task (std::function< void(void)> F)

  SYCL single_task launches a computation without parallelism at launch time.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for (range< Dimensions > r, ParallelForFunctor f)

  SYCL parallel_for launches a data parallel computation with parallelism specified at launch time by a range<>.

- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

  A variation of SYCL parallel_for to take into account a nd_range<>
- template<typename Range , typename Program , typename ParallelForFunctor >
  void cl::sycl::parallel_for (Range r, Program p, ParallelForFunctor f)

  SYCL parallel_for version that allows a Program object to be specified.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFunctor f)

  Loop on the work-groups.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_workitem (group< Dimensions > g, ParallelForFunctor f)

  Loop on the work-items inside a work-group.
- void **cl::sycl::barrier** (int barrier_type)

  The kernel synchronization barrier.

Variables

- int const **cl::sycl::CL_LOCAL_MEM_FENCE** = 123

11.1.1 Macro Definition Documentation

11.1.1.1 #define __CL_ENABLE_EXCEPTIONS

Define TRISYCL_OPENCL to add OpenCL.

triSYCL can indeed work without OpenCL if only host support is needed.

Right now it is set by Doxygen to generate the documentation.

Todo Use a macro to check instead if the OpenCL header has been included before.

But what is the right one? **OPENCL_CL_H? __OPENCL_C_VERSION**? CL_HPP_? Mostly CL_HPP_ to be able to use param_traits<> from cl.hpp...

11.1.1.2 #define STRING_CLASS std::string

The string type to be used as SYCL string.

Todo this should be more local, such as SYCL_STRING_CLASS or _SYCL_STRING_CLASS

Todo use a typedef or a using instead of a macro?

Todo implement __NO_STD_STRING

Todo Table 3.2 in provisional specification is wrong: STRING_CLASS not at the right place

11.1.1.3 #define VECTOR_CLASS std::vector

The vector type to be used as SYCL vector.

Todo this should be more local, such as SYCL_VECTOR_CLASS or _SYCL_VECTOR_CLASS

Todo use a typedef or a using instead of a macro?

Todo implement __NO_STD_VECTOR

Todo Table 3.1 in provisional specification is wrong: VECTOR_CLASS not at the right place

Index