triSYCL implementation of OpenCL SYCL

Generated by Doxygen 1.8.11

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/sycl/

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

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

Ronan at keryell dot FR

Chapter 2

Todo List

File address_space.hpp

Add the alias ..._ptr<T> = ...<T *>

Namespace cl::sycl::access

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

Class cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >

Implement it for images according so section 3.3.4.5

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor (buffer< DataType, Dimensions, Allocator > &target_buffer)

add this lacking constructor to specification

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor (buffer< DataType, Dimensions, Allocator > &target_buffer, handler &command_group_handler)

Add template allocator type in all the accessor constructors in the specification or just use a more opaque Buffer type?

fix specification where access mode should be target instead

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::begin () const

Add these functions to the specification

The fact that the lambda capture make a const copy of the accessor is not yet elegantly managed... The issue is that begin()/end() dispatch is made according to the accessor constness and not from the array member constness...

try to solve it by using some enable_if on array constness?

The issue is that the end may not be known if it is implemented by a raw OpenCL cl_mem... So only provide on the device the iterators related to the start? Actually the accessor needs to know a part of the shape to have the multidimentional addressing. So this only require a size_t more...

Factor out these in a template helper

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::dimensionality

in the specification: store the dimension for user request

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator* () const

Add in the specification?

Add the concept of 0-dim buffer and accessor for scalar and use an implicit conversion to value_type reference to access the value with the accessor?

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator* ()

Add in the specification

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[] (nd_item< dimensionality > index)

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

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[] (nd_item< dimensionality > index) const

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

Class cl::sycl::buffer< T, Dimensions, Allocator >

We have some read-write buffers and some read-only buffers, according to the constructor called. So we could have some static checking for correctness with the accessors used, but we do not have a way in the specification to have a read-only buffer type for this.

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

Finish allocator implementation

Think about the need of an allocator when constructing a buffer from other buffers

Add constructors from arrays so that in C++17 the range and type can be infered from the constructor

Add constructors from array_ref

Member cl::sycl::buffer< T, Dimensions, Allocator >::buffer (shared_ptr_class< T > &host_data, const range< Dimensions > &buffer_range, cl::sycl::mutex_class &m, Allocator allocator={})

update the specification to replace the pointer by a reference and provide the constructor with and without a mutex

Member cl::sycl::buffer< T, Dimensions, Allocator >::buffer (shared_ptr_class< T > host_data, const range< Dimensions > &buffer_range, Allocator allocator={})

add this mutex-less constructor to the specification

Member cl::sycl::buffer< T, Dimensions, Allocator >::buffer (unique_ptr_class< T, D > &&host_data, const range< Dimensions > &buffer_range, Allocator allocator={})

Update the API to add template <typename D = std::default_delete<T>> because the unique_ptr_class/std↩ ::unique_ptr have the destructor type as dependent

Member cl::sycl::buffer< T, Dimensions, Allocator >::buffer (InputIterator start_iterator, InputIterator end↩ _iterator, Allocator allocator={})

Implement the copy back at buffer destruction

Generalize this for n-D and provide column-major and row-major initialization

a reason to have this nD is that set_final_data(weak_ptr_class<T> & finalData) is actually doing this linearization anyway

Allow read-only buffer construction too

update the specification to deal with forward iterators instead and rewrite back only when it is non const and output iterator at least

Allow initialization from ranges and collections à la STL

Member cl::sycl::buffer< T, Dimensions, Allocator >::buffer (buffer< T, Dimensions, Allocator > &b, const id< Dimensions > &base_index, const range< Dimensions > &sub_range, Allocator allocator={})

To be implemented

Update the specification to replace index by id

Member cl::sycl::buffer< T, Dimensions, Allocator >::buffer (cl_mem mem_object, queue from_queue, event available_event={}, Allocator allocator={})

To be implemented

Improve the specification to allow CLHPP objects too

Member cl::sycl::buffer< T, Dimensions, Allocator >::get_access (handler &command_group_handler)

Do we need for an accessor to increase the reference count of a buffer object? It does make more sense for a host-side accessor.

Implement the modes and targets

Member cl::sycl::buffer< T, Dimensions, Allocator >::get_access ()

Implement the modes

More elegant solution

Member cl::sycl::buffer< T, Dimensions, Allocator >::get_range () const

rename to the equivalent from array_ref proposals? Such as size() in http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0009r2.html

Member cl::sycl::buffer< T, Dimensions, Allocator >::get_size () const

rename to something else. In http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf it is named bytes() for example

Member cl::sycl::buffer< T, Dimensions, Allocator >::is_read_only () const

Add to specification

Member cl::sycl::buffer< T, Dimensions, Allocator >::set_final_data (weak_ptr_class< T > finalData)

Update the API to take finalData by value instead of by reference. This way we can have an implicit conversion possible at the API call from a shared_ptr<>, avoiding an explicit weak_ptr<> creation

figure out how set_final_data() interact with the other way to write back some data or with some data sharing with the host that can not be undone

Member cl::sycl::buffer< T, Dimensions, Allocator >::use_count () const

Add to the specification, useful for validation

Class cl::sycl::context

The implementation is quite minimal for now.

Member cl::sycl::context::get_devices () const

To be implemented

Member cl::sycl::context::get_info () const

To be implemented

Member cl::sycl::context::get_platform ()

To be implemented

Class cl::sycl::detail::accessor< T, Dimensions, Mode, Target >

Use the access::mode

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor (std::shared_ptr< detail::buffer< T, Dimensions >> target_buffer)

fix the specification to rename target that shadows template parm

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor (std::shared_ptr< detail::buffer< T, Dimensions >> target_buffer, handler &command_group_handler)

fix the specification to rename target that shadows template parm

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::begin () const

Add these functions to the specification

The fact that the lambda capture make a const copy of the accessor is not yet elegantly managed... The issue is that begin()/end() dispatch is made according to the accessor constness and not from the array member constness...

try to solve it by using some enable_if on array constness?

The issue is that the end may not be known if it is implemented by a raw OpenCL cl_mem... So only provide on the device the iterators related to the start? Actually the accessor needs to know a part of the shape to have the multidimentional addressing. So this only require a size_t more...

Factor out these in a template helper

Do we need this in detail::accessor too or only in accessor?

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_back_cl_buffer ()

Move this into the buffer with queue/device-based caching

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_in_cl_buffer ()

Move this into the buffer with queue/device-based caching

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::dimensionality

in the specification: store the dimension for user request

Use another name, such as from C++17 committee discussions.

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_size () const

It is incompatible with buffer get_size() in the spec

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_read_access () const

Strangely, it is not really constexpr because it is not a static method...

to move in the access::mode enum class and add to the specification ?

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write_access () const

Strangely, it is not really constexpr because it is not a static method...

to move in the access::mode enum class and add to the specification ?

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::iterator

Add iterators to accessors in the specification

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator* ()

Add in the specification

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator* () const

Add in the specification?

Add the concept of 0-dim buffer and accessor for scalar and use an implicit conversion to value_type reference to access the value with the accessor?

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[] (nd_item< dimensionality > index)

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

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[] (nd_item< dimensionality > index) const

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

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::value_type

in the specification: store the types for user request as STL or C++AMP

Member cl::sycl::detail::address_space_array< T, AS >::address_space_array (std::initializer_list< std↩ ::remove_extent_t< T >> list)

Extend to more than 1 dimension

Class cl::sycl::detail::address_space_base< T, AS >

Verify/improve to deal with const/volatile?

Member cl::sycl::detail::address_space_base< T, AS >::opencl_type

Add to the specification

Member cl::sycl::detail::address_space_base< T, AS >::type

Add to the specification

Class cl::sycl::detail::address_space_fundamental< T, AS >

Verify/improve to deal with const/volatile?

Class cl::sycl::detail::address_space_object< T, AS >

Verify/improve to deal with const/volatile?

what about T having some final methods?

Member cl::sycl::detail::address_space_object< T, AS >::opencl_type

Add to the specification

Member cl::sycl::detail::address_space_variable< T, AS >::opencl_type

Add to the specification

Member cl::sycl::detail::buffer< T, Dimensions >::buffer (const T ∗host_data, const range< Dimensions > &r)

Clarify the semantics in the spec. What happens if the host change the host_data after buffer creation?

Member cl::sycl::detail::buffer< T, Dimensions >::get_access ()

Remove if not used

Member cl::sycl::detail::buffer< T, Dimensions >::get_size () const

rename to something else. In http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.↩ pdf it is named bytes() for example

Member cl::sycl::detail::buffer< T, Dimensions >::set_final_data (weak_ptr_class< T > &&finalData)

Add a write kernel dependency on the buffer so the buffer destructor has to wait for the kernel execution if the buffer is also accessed through a write accessor

Member cl::sycl::detail::buffer< T, Dimensions >::∼buffer ()

To implement and deal with reference counting buffer(buffer<T, Dimensions> b, index<Dimensions> base_↩ index, range<Dimensions> sub_range)

Allow CLHPP objects too?

Member cl::sycl::detail::buffer_add_to_task (BufferDetail buf, handler ∗command_group_handler, bool is↩ _write_mode)

To remove with some refactoring

Member cl::sycl::detail::buffer_base::read_only

Replace this by a static read-only type for the buffer

Member cl::sycl::detail::device::has_extension (const string_class &extension) const =0

virtual cannot be templated template <typename t>=""> virtual T get_info(info::device param) const = 0;

Class cl::sycl::detail::host_device

The implementation is quite minimal for now. :-)

Member cl::sycl::detail::host_device::get_platform () const override

To be implemented

Member cl::sycl::detail::host_device::has_extension (const string_class &extension) const override

To be implemented

Member cl::sycl::detail::host_platform::has_extension (const string_class &extension) const override

To be implemented

Class cl::sycl::detail::host_queue

Once a triSYCL queue is no longer blocking, make this a singleton

Member cl::sycl::detail::opencl_device::get_platform () const override

To be implemented

Member cl::sycl::detail::opencl_device::has_extension (const string_class &extension) const override

To be implemented

Member cl::sycl::detail::opencl_kernel::get () const override

Improve the spec to deprecate C OpenCL host API and move to C++ instead to avoid this ugly ownership management

Test error and throw. Externalize this feature in Boost.Compute?

Member cl::sycl::detail::opencl_queue::get_context () const override

Finish context

Member cl::sycl::detail::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::detail::parallel_for_workitem (const group< Dimensions > &g, ParallelForFunctor f)

Better type the functor

Member cl::sycl::detail::pipe< T >::write (const T &value, bool blocking=false)

provide a && version

Member cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::pipe_accessor (const std::shared_ptr< detail::pipe< T >> &p, handler &command_group_handler)

Use pipe_exception instead

Member cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::write (const value_type &value) const

provide a && version

Member cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity ()

Throw exception instead

Member cl::sycl::detail::pipe_reservation< PipeAccessor >::commit ()

Add to the specification that for simplicity a reservation can be commited several times but only the first one is taken into account

Member cl::sycl::detail::queue::~queue ()

Update according spec since queue destruction is non blocking

Member cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::operator< (const Parent &other) const

Add this to the spec

Member cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >:↩ :dimensionality

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

Member cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >::small_array_123 (BasicType e)

Add to the specification of the range, id...

Member cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >::small_array_123 (BasicType e)

Add to the specification of the range, id...

Member cl::sycl::detail::task::buffers_in_use

Use a set to check that some buffers are not used many times at least on writing

Member cl::sycl::detail::task::get_kernel ()

Specify this error in the spec

Member cl::sycl::device::device (const device_selector &ds)

Make it non-explicit in the specification?

Member cl::sycl::device::get_info (info::device param) const

Member cl::sycl::device::get_info () const

Member cl::sycl::device::type () const

Present in Boost.Compute, to be added to the specification

Member cl::sycl::device_selector::select_device () const

Remove this from specification

Class cl::sycl::device_type_selector

To be added to the specification

Class cl::sycl::device_typename_selector< DeviceType >

To be added to the specification

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

Class cl::sycl::exception_list

Do we need to define it in SYCL or can we rely on plain C++17 one?

Member cl::sycl::exception_ptr

Do we need this instead of reusing directly the one from C++11?

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

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

Member cl::sycl::group< dims >::get_group_range () const

Fix this comment and the specification

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

Add to the specification

Member cl::sycl::group< dims >::get_local_range (int dimension) const

Add to the specification

Member cl::sycl::group< dims >::get_nd_range () const

Also provide this access to the current nd_range

Member cl::sycl::group< dims >::get_offset () const

Add to the specification

Member cl::sycl::group< dims >::get_offset (int dimension) const

Add to the specification

Member cl::sycl::group< dims >::group (const id< dims > &i, const nd_range< dims > &ndr)

This should be private somehow, but it is used by the validation infrastructure

Member cl::sycl::group< dims >::group ()=default

Make most of them protected, reserved to implementation

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

This should be private since it is only used by the triSYCL implementation

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

In this implementation it is not const because the group<> is written in the parallel_for iterators. To fix according to the specification

Member cl::sycl::group< dims >::parallel_for_work_item (std::function< void(nd_item< dimensionality >)> f) const

Add this method in the specification

Member cl::sycl::group< dims >::parallel_for_work_item (std::function< void(item< dimensionality >)> f) const

Add this method in the specification

Member cl::sycl::handler::set_arg (int arg_index, accessor< DataType, Dimensions, Mode, Target > acc_obj)

Update the specification to use a ref && to the accessor instead?

It is not that clean to have set_arg() associated to a command handler. Rethink the specification?

It seems more logical to have these methods on kernel instead

Member cl::sycl::handler::set_arg (int arg_index, T scalar_value)

It is not that clean to have set_arg() associated to a command handler. Rethink the specification?

To be implemented

Member cl::sycl::handler::set_args (Ts &&...args)

Update the specification to add this function according to `https://cvs.khronos.org/bugzilla/show↩_bug.cgi?id=15978` proposal

Member cl::sycl::handler::single_task (kernel syclKernel)

Add in the spec a version taking a kernel and a functor, to have host fall-back

To be implemented

Member cl::sycl::handler::TRISYCL_ParallelForKernel_RANGE (1) TRISYCL_ParallelForKernel_RANGE(2) TRISYCL_ParallelForKernel_RANGE(3) template< std

Add in the spec a version taking a kernel and a functor, to have host fall-back

To be implemented

Class cl::sycl::image< dimensions >

implement image

Member cl::sycl::info::context

Should be unsigned int to be consistent with others?

Member cl::sycl::info::device

Should be unsigned int?

Member cl::sycl::info::device_type

To be moved in the specification from platform to device

Add opencl to the specification

there is no accelerator_selector and custom_accelerator

Member cl::sycl::info::queue

unsigned int?

To be implemented

To be implemented

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

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

Member cl::sycl::item< dims >::item ()=default

Make most of them protected, reserved to implementation

Member cl::sycl::item< dims >::set (id< dims > Index)

Move to private and add friends

Class cl::sycl::kernel

To be implemented

Check specification

Member cl::sycl::make_multi (multi_ptr< T, AS > pointer)

Implement the case with a plain pointer

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

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

Member cl::sycl::nd_item< dims >::get_item () const

Add to the specification

Member cl::sycl::nd_item< dims >::nd_item ()=default

Make most of them protected, reserved to implementation

Member cl::sycl::nd_item< dims >::nd_item (id< dims > global_index, nd_range< dims > ndr)

This is for validation purpose. Hide this to the programmer somehow

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

This is for the triSYCL implementation which is expected to call set_global() and set_local() later. This should be hidden to the user.

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 () const

get_offset() is lacking in the specification

Class cl::sycl::non_cl_error

Add to the specification

Clean implementation

Exceptions are named error in C++

Member cl::sycl::parallel_for_work_item (const group< Dimensions > &g, ParallelForFunctor f)

To be implemented

Deprecate this function in the specification to use instead the group method

Member cl::sycl::pipe_reservation< PipeAccessor >::pipe_reservation (detail::pipe_reservation< accessor_detail > &&pr)

Make it private and add required friends

Class cl::sycl::platform

triSYCL Implementation

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

Define a SYCL exception for this

Member cl::sycl::platform::get_info (info::platform param) const

Add to the specification

Class cl::sycl::queue

The implementation is quite minimal for now. :-)

All the queue methods should return a queue& instead of void to it is possible to chain opoerations

Member cl::sycl::queue::queue (const boost::compute::command_queue &q, async_handler ah=nullptr)

Deal with handler

Member cl::sycl::queue::submit (std::function< void(handler &)> cgf)

Add in the spec an implicit conversion of handler_event to queue& so it is possible to chain operations on the queue

Update the spec to replace std::function by a templated type to avoid memory allocation

Class cl::sycl::range< dims >

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

add to the specification this default parameter value?

add to the specification some way to specify an offset?

Member cl::sycl::range< dims >::get_count ()

Give back size() its real meaning in the specification

add this method to the specification

Namespace cl::sycl::trisycl

Refactor when updating to latest specification

Class cl::sycl::vec< DataType, NumElements >

add [] operator

add iterators on elements, with begin() and end()

having vec<> sub-classing array<> instead would solve the previous issues

move the implementation elsewhere

simplify the helpers by removing some template types since there are now inside the vec<> class.

rename in the specification element_type to value_type

Class handler_event

To be implemented

To be implemented

Member TRISYCL_ParallelForKernel_RANGE (N)

Add in the spec a version taking a kernel and a functor, to have host fall-back

Think to a cleaner solution

Think to a cleaner solution

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 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 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

- class cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >

  The accessor abstracts the way buffer or pipe data are accessed inside a kernel in a multidimensional variable length array way. More...
- class cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >

  The pipe accessor abstracts the way pipe data are accessed inside a kernel. More...
- class cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >

  The pipe accessor abstracts the way pipe data are accessed inside a kernel. More...
- class cl::sycl::detail::accessor< T, Dimensions, Mode, Target >

  The buffer accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way. More...
- class cl::sycl::detail::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...
- class cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >

  A helper class to wait for the final buffer destruction if the conditions for blocking are met. More...
- struct cl::sycl::image< dimensions >
- struct cl::sycl::detail::reserve_id< T >

  A private description of a reservation station. More...
- class cl::sycl::detail::pipe< T >

  Implement a pipe object. More...
- class cl::sycl::detail::pipe_accessor< T, AccessMode, Target >

  The accessor abstracts the way pipe data are accessed inside a kernel. More...
- class cl::sycl::pipe< T >

  A SYCL pipe. More...
- class cl::sycl::detail::pipe_reservation< PipeAccessor >

  The implementation of the pipe reservation station. More...
- struct cl::sycl::pipe_reservation< PipeAccessor >

  The pipe reservation station allows to reserve an array-like view inside the pipe for ordered race-free access from various work-items for example. More...
- class cl::sycl::static_pipe< T, Capacity >

  A SYCL static-scoped pipe equivalent to an OpenCL program-scoped pipe. More...

Typedefs

- template<typename T >
  using cl::sycl::buffer_allocator = std::allocator< T >

  The default buffer allocator used by the runtime, when no allocator is defined by the user.

Functions

- template<typename Accessor >
  static auto & cl::sycl::get_pipe_detail (Accessor &a)

  Top-level function to break circular dependencies on the the types to get the pipe implementation.
- template<typename BufferDetail >
  static std::shared_ptr< detail::task > cl::sycl::detail::buffer_add_to_task (BufferDetail buf, handler ∗command_group_handler, bool is_write_mode)

  Proxy function to avoid some circular type recursion.
- template<typename T , std::size_t Dimensions = 1>
  auto cl::sycl::detail::waiter (detail::buffer< T, Dimensions > ∗b)

  Helper function to create a new buffer_waiter.

8.1.1 Detailed Description

8.1.2 Class Documentation

8.1.2.1 class cl::sycl::accessor

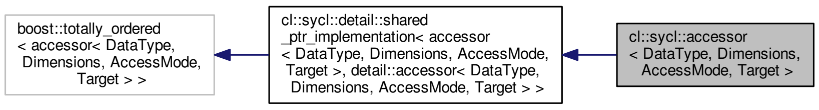
template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target↩ ::global_buffer>
class cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >

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

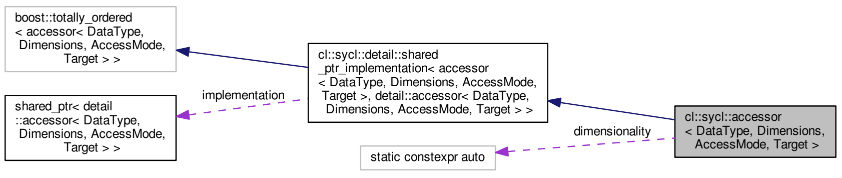
Todo Implement it for images according so section 3.3.4.5

Definition at line 45 of file accessor.hpp.

Inheritance diagram for cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >:


boost::totally_ordered < accessor< DataType, Dimensions, AccessMode, Target > >
cl::sycl::detail::shared _ptr_implementation< accessor < DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >
cl::sycl::accessor < DataType, Dimensions, AccessMode, Target >

Collaboration diagram for cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >:


boost::totally_ordered < accessor< DataType, Dimensions, AccessMode, Target > >
shared_ptr< detail ::accessor< DataType, Dimensions, AccessMode, Target > >
implementation
cl::sycl::detail::shared _ptr_implementation< accessor < DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >
dimensionality
static constexpr auto
cl::sycl::accessor < DataType, Dimensions, AccessMode, Target >

Public Types

- using value_type = DataType
- using reference = value_type &
- using const_reference = const value_type &

Public Member Functions

- template<typename Allocator >
  accessor (buffer< DataType, Dimensions, Allocator > &target_buffer, handler &command_group_handler)

  Construct a buffer accessor from a buffer using a command group handler object from the command group scope.
- template<typename Allocator >
  accessor (buffer< DataType, Dimensions, Allocator > &target_buffer)

  Construct a buffer accessor from a buffer using a command group handler object from the command group scope.
- template<typename Allocator >
  accessor (buffer< DataType, Dimensions, Allocator > &target_buffer, handler &command_group_handler, range< Dimensions > offset, range< Dimensions > range)

  Construct a buffer accessor from a buffer given a specific range for access permissions and an offset that provides the starting point for the access range using a command group handler object from the command group scope.
- accessor (range< Dimensions > allocation_size, handler &command_group_handler)

  Construct an accessor of dimensions Dimensions with elements of type DataType using the passed range to specify the size in each dimension.
- accessor_detail::reference operator[ ] (std::size_t index)

  Use the accessor with integers à la [][][].
- accessor_detail::reference operator[ ] (std::size_t index) const

  Use the accessor with integers à la [][][].
- auto & operator[ ] (id< dimensionality > index)

  To use the accessor with [id<>].
- auto & operator[ ] (id< dimensionality > index) const

  To use the accessor with [id<>].
- auto & operator[ ] (item< dimensionality > index)

  To use an accessor with [item<>].
- auto & operator[ ] (item< dimensionality > index) const

  To use an accessor with [item<>].
- auto & operator[ ] (nd_item< dimensionality > index)

  To use an accessor with an [nd_item<>].
- auto & operator[ ] (nd_item< dimensionality > index) const

  To use an accessor with an [nd_item<>].
- accessor_detail::reference operator∗ ()

  Get the first element of the accessor.
- accessor_detail::reference operator∗ () const

  Get the first element of the accessor.
- accessor_detail::iterator begin () const

  Forward all the iterator functions to the implementation.
- accessor_detail::iterator end () const
- accessor_detail::const_iterator cbegin () const
- accessor_detail::const_iterator cend () const
- accessor_detail::reverse_iterator rbegin () const
- accessor_detail::reverse_iterator rend () const
- accessor_detail::const_reverse_iterator crbegin () const
- accessor_detail::const_reverse_iterator crend () const

Static Public Attributes

- static constexpr auto dimensionality = Dimensions

Private Types

- using accessor_detail = detail::accessor< DataType, Dimensions, AccessMode, Target >
- using implementation_t = detail::shared_ptr_implementation< accessor< DataType, Dimensions, Access↩ Mode, Target >, accessor_detail >

Additional Inherited Members

8.1.2.1.1 Member Typedef Documentation

8.1.2.1.1.1 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> using cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor_detail = detail::accessor<DataType, Dimensions, AccessMode, Target> [private]

Definition at line 67 of file accessor.hpp.

8.1.2.1.1.2 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> using cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::const_reference = const value_type&

Definition at line 60 of file accessor.hpp.

8.1.2.1.1.3 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> using cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::implementation_t = detail::shared_ptr_implementation<accessor<DataType, Dimensions, AccessMode, Target>, accessor_detail> [private]

Definition at line 75 of file accessor.hpp.

8.1.2.1.1.4 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> using cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::reference = value_type&

Definition at line 59 of file accessor.hpp.

8.1.2.1.1.5 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> using cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::value_type = DataType

Definition at line 58 of file accessor.hpp.

8.1.2.1.2 Constructor & Destructor Documentation

8.1.2.1.2.1 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> template<typename Allocator > cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor (buffer< DataType, Dimensions, Allocator > & *target_buffer*, handler & *command_group_handler*) [inline]

Construct a buffer accessor from a buffer using a command group handler object from the command group scope.

Constructor only available for global_buffer or constant_buffer target.

access_target defines the form of access being obtained.

Todo Add template allocator type in all the accessor constructors in the specification or just use a more opaque Buffer type?

Todo fix specification where access mode should be target instead

Definition at line 98 of file accessor.hpp.

References cl::sycl::access::constant_buffer, cl::sycl::access::global_buffer, and cl::sycl::detail::shared_ptr_↩ implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >↩ ::implementation.

```
                                           : implementation_t {
    new detail::accessor<DataType, Dimensions, AccessMode, Target> {
      target_buffer.implementation->implementation, command_group_handler }
  } {
    static_assert(Target == access::target::global_buffer
                  || Target == access::target::constant_buffer,
                  "access target should be global_buffer or constant_buffer "
                  "when a handler is used");
  }
```

8.1.2.1.2.2 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> template<typename Allocator > cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor (buffer< DataType, Dimensions, Allocator > & *target_buffer*) [inline]

Construct a buffer accessor from a buffer using a command group handler object from the command group scope.

Constructor only available for host_buffer target.

access_target defines the form of access being obtained.

Todo add this lacking constructor to specification

Definition at line 120 of file accessor.hpp.

References cl::sycl::access::host_buffer, and cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
    : implementation_t {
    new detail::accessor<DataType, Dimensions, AccessMode, Target> {
      target_buffer.implementation->implementation }
  } {
    static_assert(Target == access::target::host_buffer,
                  "without a handler, access target should be host_buffer");
  }
```

8.1.2.1.2.3 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> template<typename Allocator > cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor (buffer< DataType, Dimensions, Allocator > & *target_buffer*, handler & *command_group_handler*, range< Dimensions > *offset*, range< Dimensions > *range*) [inline]

Construct a buffer accessor from a buffer given a specific range for access permissions and an offset that provides the starting point for the access range using a command group handler object from the command group scope.

This accessor limits the processing of the buffer to the [offset, offset+range[for every dimension. Any other parts of the buffer will be unaffected.

Constructor only available for access modes global_buffer, host_buffer or constant_buffer (see Table 3.25). access_target defines the form of access being obtained (see Table 3.26).

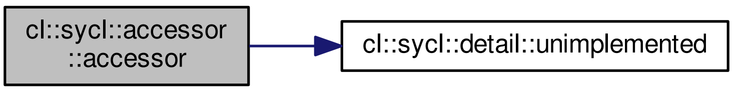
This accessor is recommended for discard-write and discard read write access modes, when the unaffected parts of the processing should be retained.

Definition at line 148 of file accessor.hpp.

References cl::sycl::detail::unimplemented().

```
                                        {
    detail::unimplemented();
  }
```

Here is the call graph for this function:


cl::sycl::accessor ::accessor
cl::sycl::detail::unimplemented

8.1.2.1.2.4 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor (range< Dimensions > *allocation_size*, handler & *command_group_handler*) [inline]

Construct an accessor of dimensions Dimensions with elements of type DataType using the passed range to specify the size in each dimension.

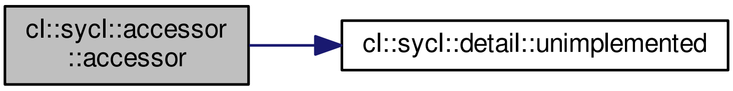
It needs as a parameter a command group handler object from the command group scope. Constructor only available if AccessMode is local, see Table 3.25.

Definition at line 164 of file accessor.hpp.

References cl::sycl::detail::unimplemented().

```
                                                {
    detail::unimplemented();
  }
```

Here is the call graph for this function:


cl::sycl::accessor ::accessor
cl::sycl::detail::unimplemented

8.1.2.1.3 Member Function Documentation

8.1.2.1.3.1 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> accessor_detail::iterator cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::begin ( ) const [inline]

Forward all the iterator functions to the implementation.

Todo Add these functions to the specification

Todo The fact that the lambda capture make a const copy of the accessor is not yet elegantly managed... The issue is that begin()/end() dispatch is made according to the accessor constness and not from the array member constness...

Todo try to solve it by using some enable_if on array constness?

Todo The issue is that the end may not be known if it is implemented by a raw OpenCL cl_mem... So only provide on the device the iterators related to the start? Actually the accessor needs to know a part of the shape to have the multidimentional addressing. So this only require a size_t more...

Todo Factor out these in a template helper

Definition at line 279 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
                                                    {
    return implementation->begin();
  }
```

8.1.2.1.3.2 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> accessor_detail::const_iterator cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::cbegin ( ) const [inline]

Definition at line 296 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
                                                           {
    return implementation->cbegin();
  }
```

8.1.2.1.3.3 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> accessor_detail::const_iterator cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::cend ( ) const [inline]

Definition at line 301 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
                                                    {
    return implementation->cend();
  }
```

8.1.2.1.3.4 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> accessor_detail::const_reverse_iterator cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::crbegin ( ) const [inline]

Definition at line 322 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
                                                               {
    return implementation->rbegin();
  }
```

8.1.2.1.3.5 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> accessor_detail::const_reverse_iterator cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::crend ( ) const [inline]

Definition at line 327 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
                                                             {
    return implementation->rend();
  }
```

8.1.2.1.3.6 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> accessor_detail::iterator cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::end ( ) const [inline]

Definition at line 285 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
                                             {
    return implementation->end();
  }
```

8.1.2.1.3.7 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> accessor_detail::reference cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator* ( ) [inline]

Get the first element of the accessor.

Useful with an accessor on a scalar for example.

Todo Add in the specification

Definition at line 237 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
                                              {
    return **implementation;
  }
```

8.1.2.1.3.8 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> accessor_detail::reference cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator* ( ) const [inline]

Get the first element of the accessor.

Useful with an accessor on a scalar for example.

Todo Add in the specification?

Todo Add the concept of 0-dim buffer and accessor for scalar and use an implicit conversion to value_type reference to access the value with the accessor?

Definition at line 252 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
                                                    {
    return **implementation;
  }
```

8.1.2.1.3.9 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> accessor_detail::reference cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[] (std::size_t *index*) [inline]

Use the accessor with integers à la [][][].

Use array_view_type::reference instead of auto& because it does not work in some dimensions.

Definition at line 175 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
                                                              {
    return (*implementation)[index];
  }
```

8.1.2.1.3.10 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> accessor_detail::reference cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[ ] (std::size_t *index*) const [inline]

Use the accessor with integers à la [][][].

Use array_view_type::reference instead of auto& because it does not work in some dimensions.

Definition at line 185 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
                                                                    {
    return (*implementation)[index];
  }
```

8.1.2.1.3.11 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> auto& cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[ ] (id< dimensionality > *index*) [inline]

To use the accessor with [id<>].

Definition at line 191 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
                                            {
    return (*implementation)[index];
  }
```

8.1.2.1.3.12 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> auto& cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[ ] (id< dimensionality > *index*) const [inline]

To use the accessor with [id<>].

Definition at line 197 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
                                                    {
    return (*implementation)[index];
  }
```

8.1.2.1.3.13 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> auto& cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[] (item< dimensionality > *index*) [inline]

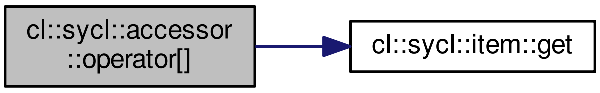
To use an accessor with [item<>].

Definition at line 203 of file accessor.hpp.

References cl::sycl::item< dims >::get().

```
                                                {
    return (*this)[index.get()];
  }
```

Here is the call graph for this function:



8.1.2.1.3.14 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> auto& cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[] (item< dimensionality > *index*) const [inline]

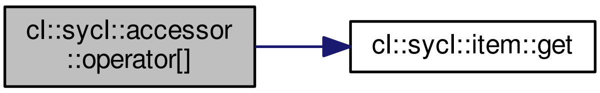
To use an accessor with [item<>].

Definition at line 209 of file accessor.hpp.

References cl::sycl::item< dims >::get().

```
                                                      {
    return (*this)[index.get()];
  }
```

Here is the call graph for this function:



8.1.2.1.3.15 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> auto& **cl::sycl::accessor**< DataType, Dimensions, AccessMode, Target >::operator[] (**nd_item**< **dimensionality** > *index*) [inline]

To use an accessor with an [nd_item<>].

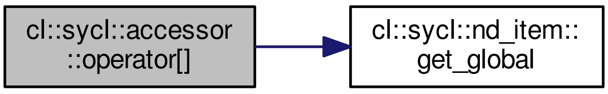
Todo Add in the specification because used by HPC-GPU slide 22

Definition at line 218 of file accessor.hpp.

References cl::sycl::nd_item< dims >::get_global().

```
                                                {
    return (*this)[index.get_global()];
  }
```

Here is the call graph for this function:


cl::sycl::accessor ::operator[]
cl::sycl::nd_item:: get_global

8.1.2.1.3.16 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> auto& **cl::sycl::accessor**< DataType, Dimensions, AccessMode, Target >::operator[] (**nd_item**< **dimensionality** > *index*) const [inline]

To use an accessor with an [nd_item<>].

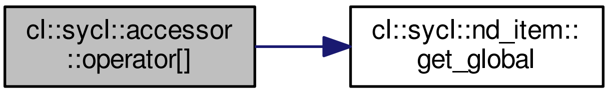
Todo Add in the specification because used by HPC-GPU slide 22

Definition at line 226 of file accessor.hpp.

References cl::sycl::nd_item< dims >::get_global().

```
                                                      {
    return (*this)[index.get_global()];
  }
```

Here is the call graph for this function:


cl::sycl::accessor ::operator[]
cl::sycl::nd_item:: get_global

8.1.2.1.3.17 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> accessor_detail::reverse_iterator cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::rbegin ( ) const [inline]

Definition at line 306 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
                                                                    {
    return implementation->rbegin();
  };
```

8.1.2.1.3.18 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> accessor_detail::reverse_iterator cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::rend ( ) const [inline]

Definition at line 311 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
                                                                  {
    return implementation->rend();
  }
```

8.1.2.1.4 Member Data Documentation

8.1.2.1.4.1 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::target::global_buffer> constexpr auto cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::dimensionality = Dimensions [static]

Todo in the specification: store the dimension for user request

Definition at line 57 of file accessor.hpp.

8.1.2.2 class cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >

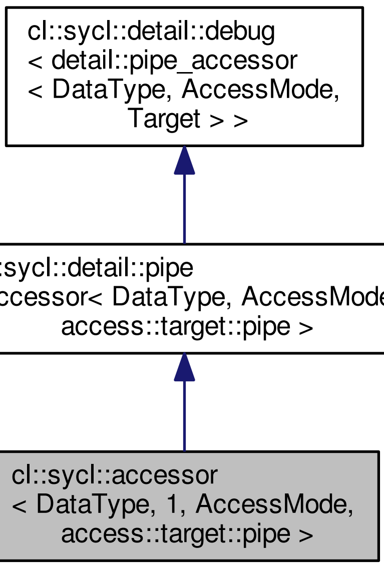
template<typename DataType, access::mode AccessMode>
class cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >

The pipe accessor abstracts the way pipe data are accessed inside a kernel.

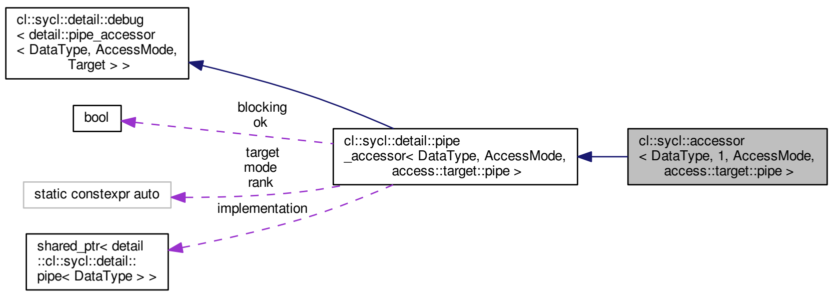
A specialization for an non-blocking pipe

Definition at line 341 of file accessor.hpp.

Inheritance diagram for cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >:


cl::sycl::detail::debug < detail::pipe_accessor < DataType, AccessMode, Target > >
cl::sycl::detail::pipe _accessor< DataType, AccessMode, access::target::pipe >
cl::sycl::accessor < DataType, 1, AccessMode, access::target::pipe >

Collaboration diagram for cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >:


cl::sycl::detail::debug < detail::pipe_accessor < DataType, AccessMode, Target > >
bool
blocking
ok
target
mode
rank
static constexpr auto
implementation
shared_ptr< detail ::cl::sycl::detail:: pipe< DataType > >
cl::sycl::detail::pipe _accessor< DataType, AccessMode, access::target::pipe >
cl::sycl::accessor < DataType, 1, AccessMode, access::target::pipe >

Public Types

- using accessor_detail = detail::pipe_accessor< DataType, AccessMode, access::target::pipe >

Public Member Functions

- accessor (pipe< DataType > &p, handler &command_group_handler)

  Construct a pipe accessor from a pipe using a command group handler object from the command group scope.
- pipe_reservation< accessor > reserve (std::size_t size) const

  Make a reservation inside the pipe.
- auto & get_pipe_detail ()

  Get the underlying pipe implementation.

Additional Inherited Members

8.1.2.2.1 Member Typedef Documentation

8.1.2.2.1.1 template<typename DataType , access::mode AccessMode> using **cl::sycl::accessor**< DataType, 1, AccessMode, **access::target::pipe** >::**accessor_detail** = **detail::pipe_accessor**<DataType, AccessMode, **access::target::pipe**>

Definition at line 346 of file accessor.hpp.

8.1.2.2.2 Constructor & Destructor Documentation

8.1.2.2.2.1 template<typename DataType , access::mode AccessMode> **cl::sycl::accessor**< DataType, 1, AccessMode, **access::target::pipe** >::**accessor** (**pipe**< DataType > & *p*, **handler** & *command_group_handler*) [inline]

Construct a pipe accessor from a pipe using a command group handler object from the command group scope.

access_target defines the form of access being obtained.

Definition at line 355 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< pipe< T >, detail::pipe< T > >::implementation.

```
00356     : accessor_detail { p.implementation, command_group_handler } { }
```

8.1.2.2.3 Member Function Documentation

8.1.2.2.3.1 template<typename DataType , access::mode AccessMode> auto& **cl::sycl::accessor**< DataType, 1, AccessMode, **access::target::pipe** >::get_pipe_detail ( ) [inline]

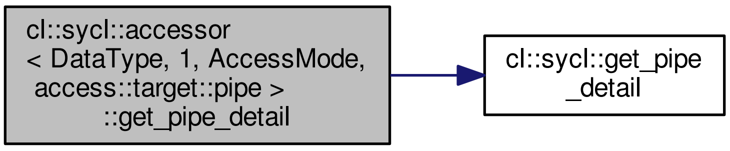
Get the underlying pipe implementation.

Definition at line 365 of file accessor.hpp.

References cl::sycl::get_pipe_detail().

```
00365                           {
00366     return accessor_detail::get_pipe_detail();
00367   }
```

Here is the call graph for this function:



8.1.2.2.3.2 template<typename DataType , access::mode AccessMode> pipe_reservation<accessor> cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >::reserve (std::size_t *size*) const [inline]

Make a reservation inside the pipe.

Definition at line 359 of file accessor.hpp.

```
00359                                                    {
00360     return accessor_detail::reserve(size);
00361   }
```

8.1.2.3 class cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >

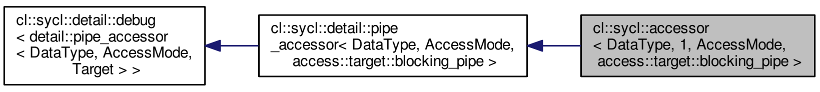
template<typename DataType, access::mode AccessMode>
class cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >

The pipe accessor abstracts the way pipe data are accessed inside a kernel.

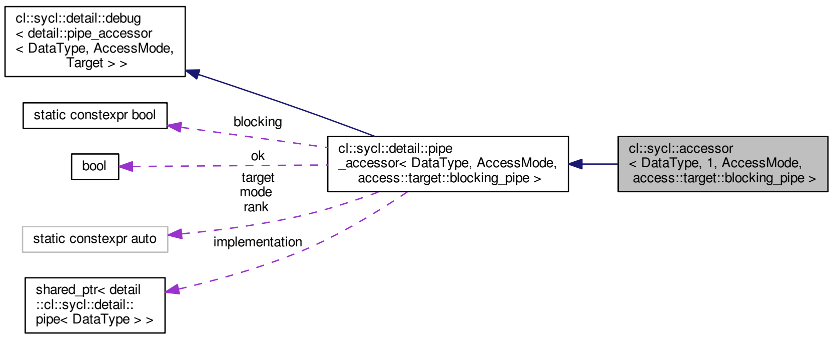
A specialization for a blocking pipe

Definition at line 379 of file accessor.hpp.

Inheritance diagram for cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >:


cl::sycl::detail::debug < detail::pipe_accessor < DataType, AccessMode, Target > >
cl::sycl::detail::pipe _accessor< DataType, AccessMode, access::target::blocking_pipe >
cl::sycl::accessor < DataType, 1, AccessMode, access::target::blocking_pipe >

Collaboration diagram for cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >:


cl::sycl::detail::debug < detail::pipe_accessor < DataType, AccessMode, Target > >
static constexpr bool
blocking
bool
ok
target
mode
rank
static constexpr auto
implementation
shared_ptr< detail ::cl::sycl::detail:: pipe< DataType > >
cl::sycl::detail::pipe _accessor< DataType, AccessMode, access::target::blocking_pipe >
cl::sycl::accessor < DataType, 1, AccessMode, access::target::blocking_pipe >

Public Types

- using accessor_detail = detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >

Public Member Functions

- accessor (pipe< DataType > &p, handler &command_group_handler)

  Construct a pipe accessor from a pipe using a command group handler object from the command group scope.
- pipe_reservation< accessor > reserve (std::size_t size) const

  Make a reservation inside the pipe.
- auto & get_pipe_detail ()

  Get the underlying pipe implementation.

Additional Inherited Members

8.1.2.3.1 Member Typedef Documentation

8.1.2.3.1.1 template<typename DataType , access::mode AccessMode> using cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >::accessor_detail = detail::pipe_accessor<DataType, AccessMode, access::target::blocking_pipe>

Definition at line 384 of file accessor.hpp.

8.1.2.3.2 Constructor & Destructor Documentation

8.1.2.3.2.1 template<typename DataType , access::mode AccessMode> cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >::accessor (pipe< DataType > & *p*, handler & *command_group_handler*) [inline]

Construct a pipe accessor from a pipe using a command group handler object from the command group scope.

access_target defines the form of access being obtained.

Definition at line 393 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< pipe< T >, detail::pipe< T > >::implementation.

```
    : accessor_detail { p.implementation, command_group_handler } { }
```

8.1.2.3.3 Member Function Documentation

8.1.2.3.3.1 template<typename DataType , access::mode AccessMode> auto& cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >::get_pipe_detail ( ) [inline]

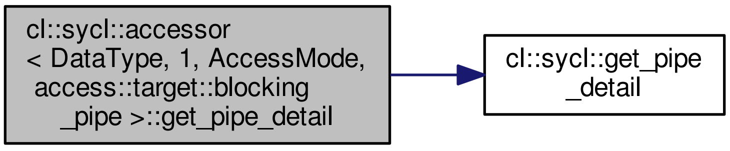
Get the underlying pipe implementation.

Definition at line 404 of file accessor.hpp.

References cl::sycl::get_pipe_detail().

```
                            {
    return accessor_detail::get_pipe_detail();
  }
```

Here is the call graph for this function:



8.1.2.3.3.2 template<typename DataType , access::mode AccessMode> **pipe_reservation<accessor>** **cl::sycl::accessor**< DataType, 1, AccessMode, **access::target::blocking_pipe** >::reserve (std::size_t *size*) const [inline]

Make a reservation inside the pipe.

Definition at line 398 of file accessor.hpp.

```
00398                                                          {
00399     return accessor_detail::reserve(size);
00400   }
```

8.1.2.4 class cl::sycl::detail::accessor

template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target>
class cl::sycl::detail::accessor< T, Dimensions, Mode, Target >

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

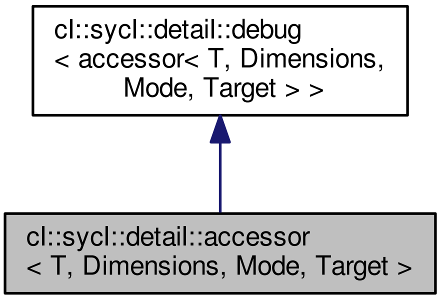
This implementation relies on boost::multi_array to provide this nice syntax and behaviour.

Right now the aim of this class is just to access to the buffer in a read-write mode, even if capturing the multi_↩
array_ref from a lambda make it const (since in examples we have lambda with [=] without mutable lambda).

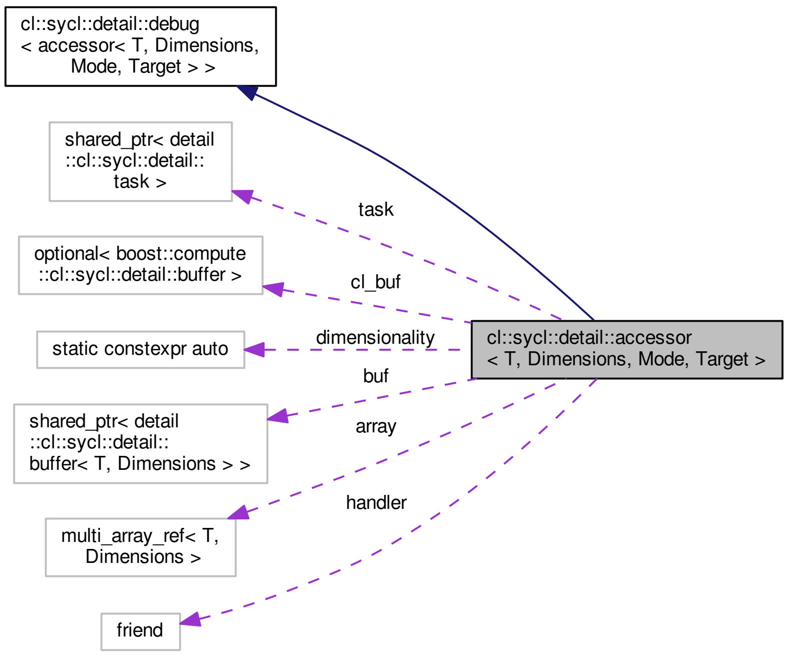
Todo Use the access::mode

Definition at line 58 of file accessor.hpp.

Inheritance diagram for cl::sycl::detail::accessor< T, Dimensions, Mode, Target >:


cl::sycl::detail::debug
< accessor< T, Dimensions,
Mode, Target > >
cl::sycl::detail::accessor
< T, Dimensions, Mode, Target >

Collaboration diagram for cl::sycl::detail::accessor< T, Dimensions, Mode, Target >:


cl::sycl::detail::debug < accessor< T, Dimensions, Mode, Target > >
shared_ptr< detail ::cl::sycl::detail:: task >
task
optional< boost::compute ::cl::sycl::detail::buffer >
cl_buf
static constexpr auto
dimensionality
cl::sycl::detail::accessor < T, Dimensions, Mode, Target >
buf
shared_ptr< detail ::cl::sycl::detail:: buffer< T, Dimensions > >
array
multi_array_ref< T, Dimensions >
handler
friend

Public Types

- using value_type = T
- using element = T
- using reference = typename array_view_type::reference
- using const_reference = typename array_view_type::const_reference
- using iterator = typename array_view_type::iterator

  Inherit the iterator types from the implementation.
- using const_iterator = typename array_view_type::const_iterator
- using reverse_iterator = typename array_view_type::reverse_iterator
- using const_reverse_iterator = typename array_view_type::const_reverse_iterator

Public Member Functions

- accessor (std::shared_ptr< detail::buffer< T, Dimensions >> target_buffer)

  Construct a host accessor from an existing buffer.
- accessor (std::shared_ptr< detail::buffer< T, Dimensions >> target_buffer, handler &command_group_↩ handler)

  Construct a device accessor from an existing buffer.
- std::size_t get_size () const

  Returns the size of the underlying buffer in number of elements.
- reference operator[ ] (std::size_t index)

  Use the accessor with integers à la [][][].
- reference operator[ ] (std::size_t index) const

  Use the accessor with integers à la [][][].

- auto & operator[ ] (id< dimensionality > index)

  To use the accessor with [id<>].
- auto & operator[ ] (id< dimensionality > index) const

  To use the accessor with [id<>].
- auto & operator[ ] (item< dimensionality > index)

  To use an accessor with [item<>].
- auto & operator[ ] (item< dimensionality > index) const

  To use an accessor with [item<>].
- auto & operator[ ] (nd_item< dimensionality > index)

  To use an accessor with an [nd_item<>].
- auto & operator[ ] (nd_item< dimensionality > index) const

  To use an accessor with an [nd_item<>].
- reference operator* ()

  Get the first element of the accessor.
- reference operator* () const

  Get the first element of the accessor.
- detail::buffer< T, Dimensions > & get_buffer ()

  Get the buffer used to create the accessor.
- constexpr bool is_read_access () const

  Test if the accessor has a read access right.
- constexpr bool is_write_access () const

  Test if the accessor has a write access right.
- iterator begin () const

  Forward all the iterator functions to the implementation.
- iterator end () const
- const_iterator cbegin () const
- const_iterator cend () const
- reverse_iterator rbegin () const
- reverse_iterator rend () const
- const_reverse_iterator crbegin () const
- const_reverse_iterator crend () const

Static Public Attributes

- static constexpr auto dimensionality = Dimensions

Private Types

- using array_view_type = boost::multi_array_ref< T, Dimensions >

  The implementation is a multi_array_ref wrapper.
- using writable_array_view_type = typename std::remove_const< array_view_type >::type

Private Member Functions

- auto get_cl_buffer () const

  Get the boost::compute::buffer or throw if unset.
- void copy_in_cl_buffer ()

  Lazily associate a CL buffer to the SYCL buffer and copy data in if required.
- void copy_back_cl_buffer ()

  Copy back the CL buffer to the SYCL if required.

Private Attributes

- std::shared_ptr< detail::buffer< T, Dimensions > > buf

  Keep a reference to the accessed buffer.
- array_view_type array

  The way the buffer is really accessed.
- std::shared_ptr< detail::task > task

  The task where the accessor is used in.
- boost::optional< boost::compute::buffer > cl_buf

  The OpenCL buffer used by an OpenCL accessor.
- friend handler

8.1.2.4.1 Member Typedef Documentation

8.1.2.4.1.1 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array_view_type = boost::multi_array_ref<T, Dimensions> [private]

The implementation is a multi_array_ref wrapper.

Definition at line 71 of file accessor.hpp.

8.1.2.4.1.2 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::const_iterator = typename array_view_type::const_iterator

Definition at line 114 of file accessor.hpp.

8.1.2.4.1.3 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::const_reference = typename array_view_type::const_reference

Definition at line 107 of file accessor.hpp.

8.1.2.4.1.4 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::const_reverse_iterator = typename array_view_type::const_reverse_iterator

Definition at line 117 of file accessor.hpp.

8.1.2.4.1.5 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::element = T

Definition at line 105 of file accessor.hpp.

8.1.2.4.1.6 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::iterator = typename array_view_type::iterator

Inherit the iterator types from the implementation.

Todo Add iterators to accessors in the specification

Definition at line 113 of file accessor.hpp.

8.1.2.4.1.7 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::reference = typename array_view_type::reference

Definition at line 106 of file accessor.hpp.

8.1.2.4.1.8 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::reverse_iterator = typename array_view_type::reverse_iterator

Definition at line 115 of file accessor.hpp.

8.1.2.4.1.9 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::value_type = T

Todo in the specification: store the types for user request as STL or C++AMP

Definition at line 104 of file accessor.hpp.

8.1.2.4.1.10 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::writable_array_view_type = typename std::remove_const<array_view_type>::type [private]

Definition at line 75 of file accessor.hpp.

8.1.2.4.2 Constructor & Destructor Documentation

8.1.2.4.2.1 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor (std::shared_ptr< detail::buffer< T, Dimensions >> *target_buffer*) [inline]

Construct a host accessor from an existing buffer.

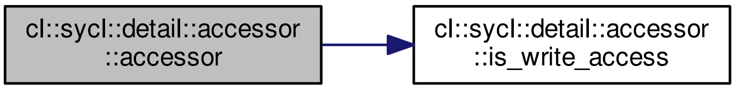
Todo fix the specification to rename target that shadows template parm

Definition at line 125 of file accessor.hpp.

References cl::sycl::access::host_buffer, cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write_↩ access(), and TRISYCL_DUMP_T.

```
                                                         :
    buf { target_buffer }, array { target_buffer->access } {
    TRISYCL_DUMP_T("Create a host accessor write = " <<
  is_write_access());
    static_assert(Target == access::target::host_buffer,
                  "without a handler, access target should be host_buffer");
    /* The host needs to wait for all the producers of the buffer to
       have finished */
    buf->wait();
  }
```

Here is the call graph for this function:


cl::sycl::detail::accessor ::accessor
cl::sycl::detail::accessor ::is_write_access

8.1.2.4.2.2 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor (std::shared_ptr< detail::buffer< T, Dimensions >> *target_buffer*, handler & *command_group_handler*) [inline]

Construct a device accessor from an existing buffer.

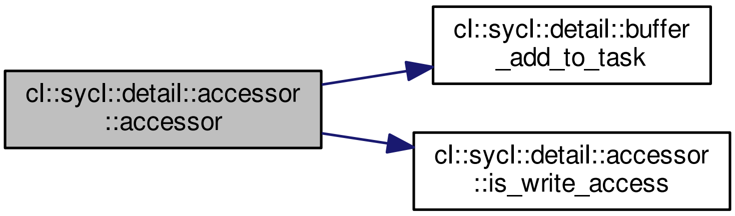
Todo fix the specification to rename target that shadows template parm

Definition at line 141 of file accessor.hpp.

References cl::sycl::detail::buffer_add_to_task(), cl::sycl::access::constant_buffer, cl::sycl::access::global_buffer, cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write_access(), and TRISYCL_DUMP_T.

```
                                              :
    buf { target_buffer }, array { target_buffer->access } {
    TRISYCL_DUMP_T("Create a kernel accessor write = " <<
is_write_access());
    static_assert(Target == access::target::global_buffer
                  || Target == access::target::constant_buffer,
                  "access target should be global_buffer or constant_buffer "
                  "when a handler is used");
    // Register the buffer to the task dependencies
    task = buffer_add_to_task(buf, &command_group_handler,
is_write_access());
  }
```

Here is the call graph for this function:


cl::sycl::detail::accessor ::accessor
cl::sycl::detail::buffer _add_to_task
cl::sycl::detail::accessor ::is_write_access

8.1.2.4.3 Member Function Documentation

8.1.2.4.3.1 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::begin ( ) const [inline]

Forward all the iterator functions to the implementation.

Todo Add these functions to the specification

Todo The fact that the lambda capture make a const copy of the accessor is not yet elegantly managed... The issue is that begin()/end() dispatch is made according to the accessor constness and not from the array member constness...

Todo try to solve it by using some enable_if on array constness?

Todo The issue is that the end may not be known if it is implemented by a raw OpenCL cl_mem... So only provide on the device the iterators related to the start? Actually the accessor needs to know a part of the shape to have the multidimentional addressing. So this only require a size_t more...

Todo Factor out these in a template helper

Todo Do we need this in detail::accessor too or only in accessor?

Definition at line 312 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array.

```
                               {
      return const_cast<writable_array_view_type &>(array).
      begin();
    }
```

8.1.2.4.3.2 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> const_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::cbegin ( ) const [inline]

Definition at line 329 of file accessor.hpp.

```
{ return array.begin(); }
```

8.1.2.4.3.3 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> const_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::cend ( ) const [inline]

Definition at line 332 of file accessor.hpp.

```
{ return array.end(); }
```

8.1.2.4.3.4 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> void cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_back_cl_buffer ( ) [inline], [private]

Copy back the CL buffer to the SYCL if required.

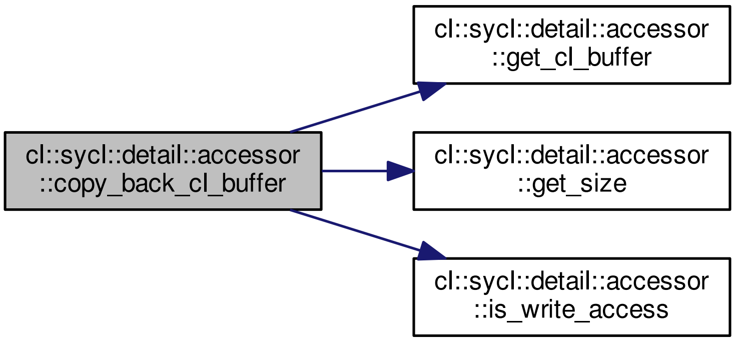
Todo Move this into the buffer with queue/device-based caching

Definition at line 396 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_cl_buffer(), cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_size(), and cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write↩ _access().

```
                          {
    // \todo Use if constexpr in C++17
    if (is_write_access())
      task->get_queue()->get_boost_compute()
        .enqueue_read_buffer(get_cl_buffer(),
                              0 /*< Offset */,
                              get_size()*sizeof(value_type),
                              array.data());
  }
```

Here is the call graph for this function:


cl::sycl::detail::accessor ::copy_back_cl_buffer
cl::sycl::detail::accessor ::get_cl_buffer
cl::sycl::detail::accessor ::get_size
cl::sycl::detail::accessor ::is_write_access

8.1.2.4.3.5 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> void cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_in_cl_buffer ( ) [inline], [private]

Lazily associate a CL buffer to the SYCL buffer and copy data in if required.

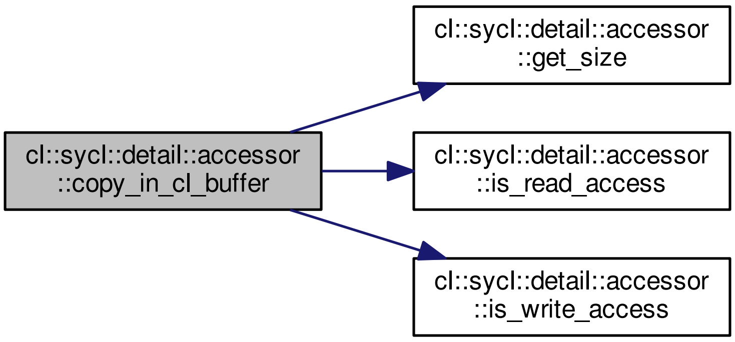
Todo Move this into the buffer with queue/device-based caching

Definition at line 376 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_size(), cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_read_access(), and cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write_access().

```
                        {
    // This should be a constexpr
    cl_mem_flags flags = is_read_access() && is_write_access() ?
      CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR
      : is_read_access() ? CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR
                         : CL_MEM_WRITE_ONLY;

    /* Create the OpenCL buffer and copy in data from the host if in
       read mode */
    cl_buf = { task->get_queue()->get_boost_compute().get_context(),
               get_size()*sizeof(value_type),
               flags,
               is_read_access() ? array.data() : 0 };
  }
```

Here is the call graph for this function:


cl::sycl::detail::accessor ::copy_in_cl_buffer
cl::sycl::detail::accessor ::get_size
cl::sycl::detail::accessor ::is_read_access
cl::sycl::detail::accessor ::is_write_access

8.1.2.4.3.6 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> const_reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::crbegin ( ) const [inline]

Definition at line 353 of file accessor.hpp.

```
{ return array.rbegin(); }
```

8.1.2.4.3.7 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> const_reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::crend ( ) const [inline]

Definition at line 356 of file accessor.hpp.

```
{ return array.rend(); }
```

8.1.2.4.3.8 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::end ( ) const [inline]

Definition at line 318 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array.

```
                           {
    return const_cast<writable_array_view_type &>(array).
  end();
  }
```

8.1.2.4.3.9 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> detail::buffer<T, Dimensions>& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_buffer ( ) [inline]

Get the buffer used to create the accessor.

Definition at line 251 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::buf.

```
                                                    {
    return *buf;
  }
```

8.1.2.4.3.10 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> auto cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_cl_buffer ( ) const [inline], [private]

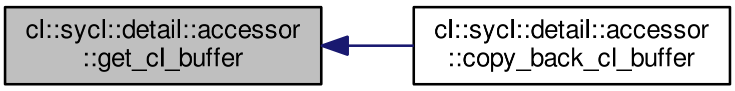
Get the boost::compute::buffer or throw if unset.

Definition at line 365 of file accessor.hpp.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_back_cl_buffer().

```
                                   {
    // This throws if not set
    return cl_buf.value();
  }
```

Here is the caller graph for this function:


cl::sycl::detail::accessor ::get_cl_buffer
cl::sycl::detail::accessor ::copy_back_cl_buffer

8.1.2.4.3.11 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> std::size_t cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_size ( ) const [inline]

Returns the size of the underlying buffer in number of elements.

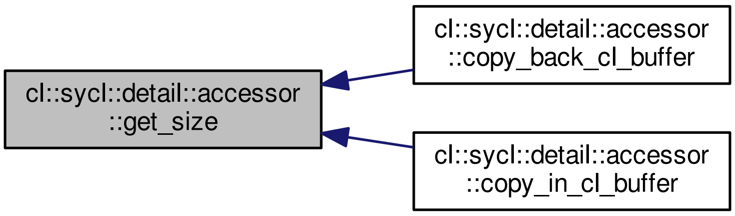
Todo It is incompatible with buffer get_size() in the spec

Definition at line 158 of file accessor.hpp.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_back_cl_buffer(), and cl::sycl↵ ::detail::accessor< T, Dimensions, Mode, Target >::copy_in_cl_buffer().

```
                             {
    return array.num_elements();
  }
```

Here is the caller graph for this function:


cl::sycl::detail::accessor ::get_size
cl::sycl::detail::accessor ::copy_back_cl_buffer
cl::sycl::detail::accessor ::copy_in_cl_buffer

8.1.2.4.3.12 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> constexpr bool cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_read_access ( ) const [inline]

Test if the accessor has a read access right.

Todo Strangely, it is not really constexpr because it is not a static method...

Todo to move in the access::mode enum class and add to the specification ?

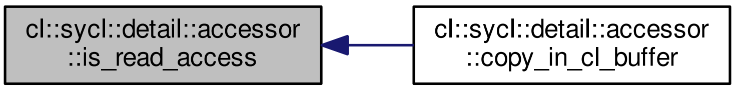
Definition at line 264 of file accessor.hpp.

References cl::sycl::access::discard_read_write, cl::sycl::access::read, and cl::sycl::access::read_write.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_in_cl_buffer().

```
                                          {
    return Mode == access::mode::read
      || Mode == access::mode::read_write
      || Mode == access::mode::discard_read_write;
  }
```

Here is the caller graph for this function:


cl::sycl::detail::accessor ::is_read_access
cl::sycl::detail::accessor ::copy_in_cl_buffer

8.1.2.4.3.13 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> constexpr bool cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write_access ( ) const [inline]

Test if the accessor has a write access right.

Todo Strangely, it is not really constexpr because it is not a static method...

Todo to move in the access::mode enum class and add to the specification ?

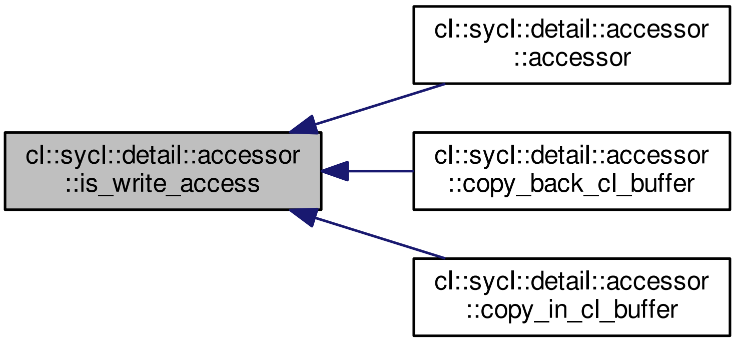
Definition at line 279 of file accessor.hpp.

References cl::sycl::access::discard_read_write, cl::sycl::access::discard_write, cl::sycl::access::read_write, and cl::sycl::access::write.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor(), cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_back_cl_buffer(), and cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_in_cl_buffer().

```
                                               {
    return Mode == access::mode::write
      || Mode == access::mode::read_write
      || Mode == access::mode::discard_write
      || Mode == access::mode::discard_read_write;
  }
```

Here is the caller graph for this function:


cl::sycl::detail::accessor ::accessor
cl::sycl::detail::accessor ::is_write_access
cl::sycl::detail::accessor ::copy_back_cl_buffer
cl::sycl::detail::accessor ::copy_in_cl_buffer

8.1.2.4.3.14 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> reference cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator* ( ) [inline]

Get the first element of the accessor.

Useful with an accessor on a scalar for example.

Todo Add in the specification

Definition at line 230 of file accessor.hpp.

```
                          {
    return *array.data();
  }
```

8.1.2.4.3.15 template< typename T , std::size_t Dimensions, access::mode Mode, access::target Target> reference cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator* ( ) const [inline]

Get the first element of the accessor.

Useful with an accessor on a scalar for example.

Todo Add in the specification?

Todo Add the concept of 0-dim buffer and accessor for scalar and use an implicit conversion to value_type reference to access the value with the accessor?

Definition at line 245 of file accessor.hpp.

```
                            {
    return *array.data();
  }
```

8.1.2.4.3.16 template< typename T , std::size_t Dimensions, access::mode Mode, access::target Target> reference cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (std::size_t *index*) [inline]

Use the accessor with integers à la [][][].

Use array_view_type::reference instead of auto& because it does not work in some dimensions.

Definition at line 168 of file accessor.hpp.

```
                                        {
    return array[index];
  }
```

8.1.2.4.3.17 template< typename T , std::size_t Dimensions, access::mode Mode, access::target Target> reference cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (std::size_t *index*) const [inline]

Use the accessor with integers à la [][][].

Use array_view_type::reference instead of auto& because it does not work in some dimensions.

Definition at line 178 of file accessor.hpp.

```
                                              {
    return array[index];
  }
```

8.1.2.4.3.18 template< typename T , std::size_t Dimensions, access::mode Mode, access::target Target> auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (id< dimensionality > *index*) [inline]

To use the accessor with [id<>].

Definition at line 184 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array.

```
                                             {
    return array(index);
  }
```

8.1.2.4.3.19 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (id< dimensionality > *index*) const [inline]

To use the accessor with [id<>].

Definition at line 190 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array.

```
                                                  {
    return array(index);
  }
```

8.1.2.4.3.20 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (item< dimensionality > *index*) [inline]

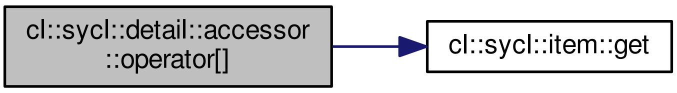
To use an accessor with [item<>].

Definition at line 196 of file accessor.hpp.

References cl::sycl::item< dims >::get().

```
                                              {
    return (*this)[index.get()];
  }
```

Here is the call graph for this function:



8.1.2.4.3.21 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (item< dimensionality > *index*) const [inline]

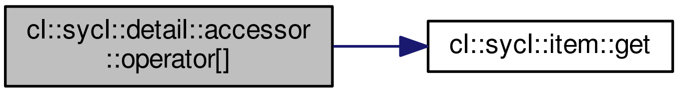
To use an accessor with [item<>].

Definition at line 202 of file accessor.hpp.

References cl::sycl::item< dims >::get().

```
                                                    {
    return (*this)[index.get()];
  }
```

Here is the call graph for this function:



8.1.2.4.3.22 template< typename T , std::size_t Dimensions, access::mode Mode, access::target Target> auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (nd_item< dimensionality > *index*) [inline]

To use an accessor with an [nd_item<>].

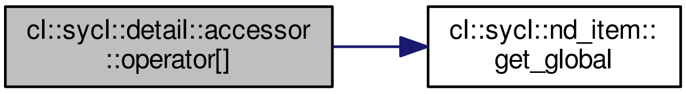
Todo Add in the specification because used by HPC-GPU slide 22

Definition at line 211 of file accessor.hpp.

References cl::sycl::nd_item< dims >::get_global().

```
                                                      {
      return (*this)[index.get_global()];
    }
```

Here is the call graph for this function:


cl::sycl::detail::accessor ::operator[]
cl::sycl::nd_item:: get_global

8.1.2.4.3.23 template< typename T , std::size_t Dimensions, access::mode Mode, access::target Target> auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (nd_item< dimensionality > *index*) const [inline]

To use an accessor with an [nd_item<>].

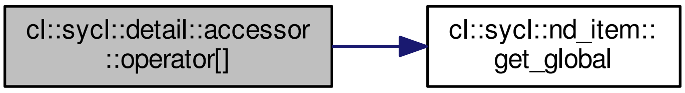
Todo Add in the specification because used by HPC-GPU slide 22

Definition at line 219 of file accessor.hpp.

References cl::sycl::nd_item< dims >::get_global().

```
                                                            {
      return (*this)[index.get_global()];
    }
```

Here is the call graph for this function:


cl::sycl::detail::accessor ::operator[]
cl::sycl::nd_item:: get_global

8.1.2.4.3.24 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::rbegin ( ) const [inline]

Definition at line 336 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array.

```
                                     {
      return const_cast<writable_array_view_type &>(array).
      rbegin();
    }
```

8.1.2.4.3.25 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::rend ( ) const [inline]

Definition at line 342 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array.

```
                                   {
      return const_cast<writable_array_view_type &>(array).
      rend();
    }
```

8.1.2.4.4 Member Data Documentation

8.1.2.4.4.1 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> array_view_type cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array [mutable], [private]

The way the buffer is really accessed.

Use a mutable member because the accessor needs to be captured by value in the lambda which is then read-only. This is to avoid the user to use mutable lambda or have a lot of const_cast as previously done in this implementation

Definition at line 84 of file accessor.hpp.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::begin(), cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::end(), cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ](), cl::sycl↩ ::detail::accessor< T, Dimensions, Mode, Target >::rbegin(), and cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::rend().

8.1.2.4.4.2 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> std::shared_ptr<detail::buffer<T, Dimensions> > cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::buf [private]

Keep a reference to the accessed buffer.

Beware that it owns the buffer, which means that the accessor has to be destroyed to release the buffer and potentially unblock a kernel at the end of its execution

Definition at line 68 of file accessor.hpp.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_buffer().

8.1.2.4.4.3 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> boost::optional<boost::compute::buffer> cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::cl_buf [private]

The OpenCL buffer used by an OpenCL accessor.

Definition at line 91 of file accessor.hpp.

8.1.2.4.4.4 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> constexpr auto cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::dimensionality = Dimensions [static]

Todo in the specification: store the dimension for user request

Todo Use another name, such as from C++17 committee discussions.

Definition at line 100 of file accessor.hpp.

8.1.2.4.4.5 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> friend cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::handler [private]

Definition at line 361 of file accessor.hpp.

8.1.2.4.4.6 template<typename T , std::size_t Dimensions, access::mode Mode, access::target Target> std::shared_ptr<detail::task> cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::task [private]

The task where the accessor is used in.

Definition at line 87 of file accessor.hpp.

8.1.2.5 class cl::sycl::detail::buffer

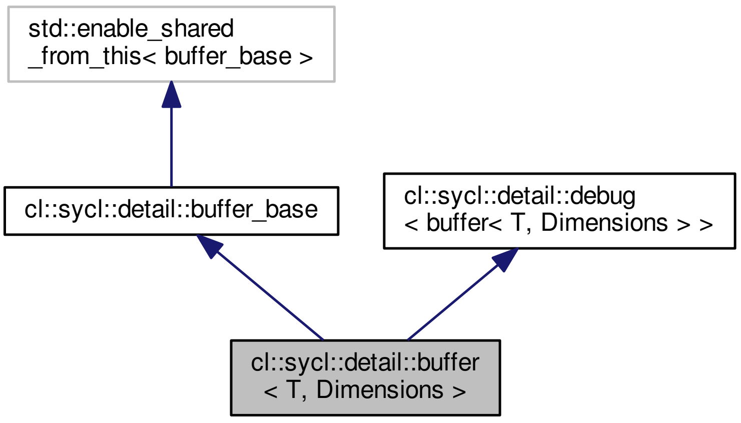
template< typename T, std::size_t Dimensions = 1>
class cl::sycl::detail::buffer< T, Dimensions >

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.

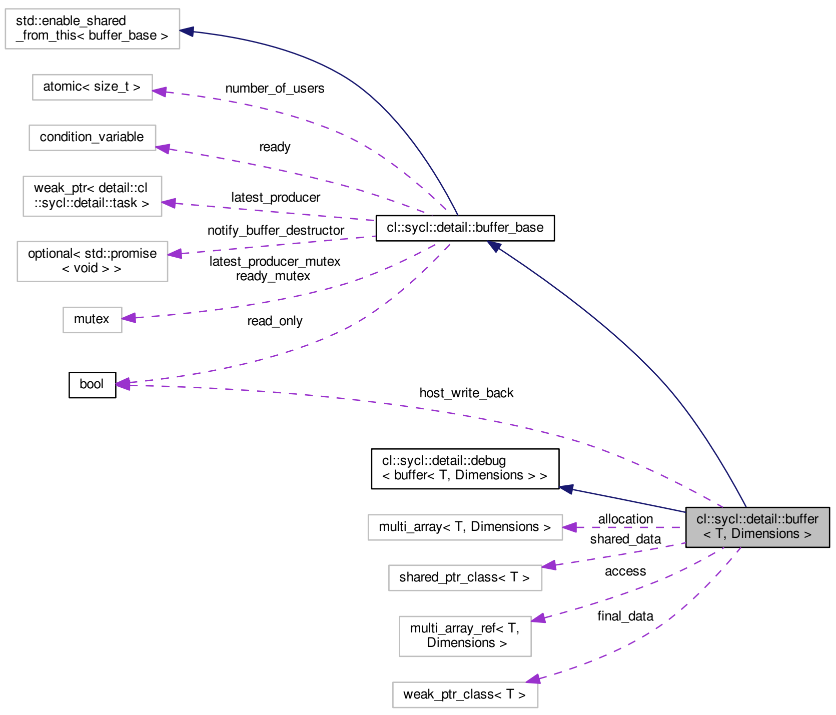
In the case we initialize it from a pointer, for now we just wrap the data with boost::multi_array_ref to provide the VLA semantics without any storage.

Definition at line 35 of file accessor.hpp.

Inheritance diagram for cl::sycl::detail::buffer< T, Dimensions >:



Collaboration diagram for cl::sycl::detail::buffer< T, Dimensions >:


std::enable_shared_from_this< buffer_base >
atomic< size_t >
number_of_users
condition_variable
ready
weak_ptr< detail::cl::sycl::detail::task >
latest_producer
notify_buffer_destructor
cl::sycl::detail::buffer_base
optional< std::promise < void > >
latest_producer_mutex
ready_mutex
mutex
read_only
bool
host_write_back
cl::sycl::detail::debug < buffer< T, Dimensions > >
multi_array< T, Dimensions >
allocation
shared_data
cl::sycl::detail::buffer < T, Dimensions >
shared_ptr_class< T >
access
multi_array_ref< T, Dimensions >
final_data
weak_ptr_class< T >

Public Types

- using element = T
- using value_type = T

Public Member Functions

- buffer (const range< Dimensions > &r)

  Create a new read-write buffer of size.
- buffer (T ∗host_data, const range< Dimensions > &r)

  Create a new read-write buffer from.
- buffer (const T ∗host_data, const range< Dimensions > &r)

  Create a new read-only buffer from.
- buffer (shared_ptr_class< T > &host_data, const range< Dimensions > &r)

  Create a new buffer with associated memory, using the data in host_data.
- template<typename Iterator >
  buffer (Iterator start_iterator, Iterator end_iterator)

  Create a new allocated 1D buffer from the given elements.
- ∼buffer ()

  Create a new sub-buffer without allocation to have separate accessors later.
- template<access::mode Mode, access::target Target = access::target::global_buffer>
  detail::accessor< T, Dimensions, Mode, Target > get_access ()

  Return an accessor of the required mode.
- auto get_range () const

Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.

- auto get_count () const

  Returns the total number of elements in the buffer.
- size_t get_size () const

  Returns the size of the buffer storage in bytes.
- void set_final_data (weak_ptr_class< T > &&finalData)

  Set the weak pointer to copy back data on buffer deletion.

Private Member Functions

- boost::optional< std::future< void > > get_destructor_future ()

  Get a `future` to wait from inside the `cl::sycl::buffer` in case there is something to copy back to the host.

Private Attributes

- boost::multi_array< T, Dimensions > allocation

  If some allocation is requested, it is managed by this multi_array to ease initialization from data.
- boost::multi_array_ref< T, Dimensions > access

  This is the multi-dimensional interface to the data that may point to either allocation in the case of storage managed by SYCL itself or to some other memory location in the case of host memory or storage<> abstraction use.
- weak_ptr_class< T > final_data

  The weak pointer to copy back data on buffer deletion.
- shared_ptr_class< T > shared_data

  The shared pointer in the case the buffer memory is shared with the host.
- bool host_write_back = false

Friends

- template<typename U , std::size_t D, access::mode Mode, access::target Target>
  class detail::accessor

Additional Inherited Members

8.1.2.5.1 Member Typedef Documentation

8.1.2.5.1.1 template<typename T, std::size_t Dimensions = 1> using cl::sycl::detail::buffer< T, Dimensions >::element = T

Definition at line 44 of file buffer.hpp.

8.1.2.5.1.2 template<typename T, std::size_t Dimensions = 1> using cl::sycl::detail::buffer< T, Dimensions >::value_type = T

Definition at line 45 of file buffer.hpp.

8.1.2.5.2 Constructor & Destructor Documentation

8.1.2.5.2.1 template<typename T, std::size_t Dimensions = 1> cl::sycl::detail::buffer< T, Dimensions >::buffer (const range< Dimensions > & *r*) `[inline]`

Create a new read-write buffer of size.

Parameters

r	

Definition at line 82 of file buffer.hpp.

```
                                        : buffer_base { false },
                                          allocation { r },
                                          access { allocation }
                                          {}
```

8.1.2.5.2.2 template<typename T, std::size_t Dimensions = 1> cl::sycl::detail::buffer< T, Dimensions >::buffer (T ∗ *host_data*, const range< Dimensions > & *r*) [inline]

Create a new read-write buffer from.

Parameters

host_data	of size
r	without further allocation

Definition at line 90 of file buffer.hpp.

```
                                                      : buffer_base { false },
                                                        access { host_data, r },
                                                        host_write_back { true }
                                                        {}
```

8.1.2.5.2.3 template<typename T, std::size_t Dimensions = 1> cl::sycl::detail::buffer< T, Dimensions >::buffer (const T ∗ *host_data*, const range< Dimensions > & *r*) [inline]

Create a new read-only buffer from.

Parameters

host_data	of size
r	without further allocation

Todo Clarify the semantics in the spec. What happens if the host change the host_data after buffer creation?

Definition at line 102 of file buffer.hpp.

```
                                                  :
    /* \todo Need to solve this const buffer issue in a clean way

       Just allocate memory? */
    buffer_base { true },
    access { const_cast<T *>(host_data), r }
    {}
```

8.1.2.5.2.4 template<typename T, std::size_t Dimensions = 1> cl::sycl::detail::buffer< T, Dimensions >::buffer (shared_ptr_class< T > & *host_data*, const range< Dimensions > & *r*) [inline]

Create a new buffer with associated memory, using the data in host_data.

The ownership of the host_data is shared between the runtime and the user. In order to enable both the user application and the SYCL runtime to use the same pointer, a cl::sycl::mutex_class is used.

Definition at line 119 of file buffer.hpp.

```
00121     : buffer_base { false },
00122     access { host_data.get(), r },
00123     shared_data { host_data }
00124     {}
```

8.1.2.5.2.5 template<typename T, std::size_t Dimensions = 1> template<typename Iterator > cl::sycl::detail::buffer< T, Dimensions >::buffer (Iterator *start_iterator*, Iterator *end_iterator*) [inline]

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

Definition at line 129 of file buffer.hpp.

```
00129                                                            :
00130     buffer_base { false },
00131     // The size of a multi_array is set at creation time
00132     allocation { boost::extents[std::distance(start_iterator, end_iterator)] },
00133     access { allocation }
00134     {
00135       /* Then assign allocation since this is the only multi_array
00136          method with this iterator interface */
00137       allocation.assign(start_iterator, end_iterator);
00138     }
```

8.1.2.5.2.6 template<typename T, std::size_t Dimensions = 1> cl::sycl::detail::buffer< T, Dimensions >::~buffer ( ) [inline]

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

Todo To implement and deal with reference counting buffer(buffer<T, Dimensions> b, index<Dimensions> base↩_index, range<Dimensions> sub_range)

Todo Allow CLHPP objects too?

The buffer content may be copied back on destruction to some final location

Definition at line 160 of file buffer.hpp.

References cl::sycl::access::global_buffer.

```
00160               {
00161     /* If there is a final_data set and that points to something
00162        alive, copy back the data through the shared pointer */
00163     if (auto p = final_data.lock())
00164       std::copy_n(access.data(), access.num_elements(), p.get());
00165     /* If data are shared with the host but not concretely, we would
00166        have to copy back the data to the host */
00167     // else if (shared_data)
00168     //   std::copy_n(access.data(), access.num_elements(), shared_data.get());
00169   }
```

8.1.2.5.3 Member Function Documentation

8.1.2.5.3.1 template<typename T, std::size_t Dimensions = 1> template<access::mode Mode, access::target Target = access::target::global_buffer> detail::accessor<T, Dimensions, Mode, Target> cl::sycl::detail::buffer< T, Dimensions >::get_access ( ) [inline]

Return an accessor of the required mode.

Parameters

M	

Todo Remove if not used

Definition at line 177 of file buffer.hpp.

```
                                                                          {
    return { *this };
  }
```

8.1.2.5.3.2 template<typename T, std::size_t Dimensions = 1> auto cl::sycl::detail::buffer< T, Dimensions >::get_count ( ) const [inline]

Returns the total number of elements in the buffer.

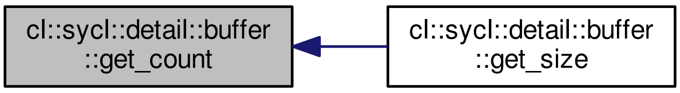
Equal to get_range()[0] ∗ ... ∗ get_range()[dimensions-1].

Definition at line 204 of file buffer.hpp.

Referenced by cl::sycl::detail::buffer< T, Dimensions >::get_size().

```
                              {
    return allocation.num_elements();
  }
```

Here is the caller graph for this function:


cl::sycl::detail::buffer ::get_count
cl::sycl::detail::buffer ::get_size

8.1.2.5.3.3 template<typename T, std::size_t Dimensions = 1> boost::optional<std::future<void> > cl::sycl::detail::buffer< T, Dimensions >::get_destructor_future ( ) [inline], [private]

Get a `future` to wait from inside the `cl::sycl::buffer` in case there is something to copy back to the host.

Returns

A `future` in the `optional` if there is something to wait for, otherwise an empty `optional`

Definition at line 240 of file buffer.hpp.

References cl::sycl::detail::buffer_base::notify_buffer_destructor, and cl::sycl::detail::buffer< T, Dimensions >↩
::shared_data.

```
                                                           {
    boost::optional<std::future<void>> f;
    /* If there is only 1 shared_ptr user of the buffer, this is the
       caller of this function, the \c buffer_waiter, so there is no
       need to get a \ future otherwise there will be a dead-lock if
       there is only 1 thread waiting for itself.

       Since \c use_count() is applied to a \c shared_ptr just created
       for this purpose, it actually increase locally the count by 1,
       so check for 1 + 1 use count instead...
    */
    if (shared_from_this().use_count() > 2)
      // \todo Double check the specification and add unit tests
      if (host_write_back || !final_data.expired() ||
shared_data) {
        // Create a promise to wait for
        notify_buffer_destructor = std::promise<void> {};
        // And return the future to wait for it
        f = notify_buffer_destructor->get_future();
      }
    return f;
  }
```

8.1.2.5.3.4 template<typename T, std::size_t Dimensions = 1> auto cl::sycl::detail::buffer< T, Dimensions >::get_range ( ) const [inline]

Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.

Definition at line 186 of file buffer.hpp.

```
                            {
    /* Interpret the shape which is a pointer to the first element as an
       array of Dimensions elements so that the range<Dimensions>
       constructor is happy with this collection

       \todo Add also a constructor in range<> to accept a const
       std::size_t *?
    */
    return range<Dimensions> {
      *(const std::size_t (*)[Dimensions])(allocation.shape())
        };
  }
```

8.1.2.5.3.5 template<typename T, std::size_t Dimensions = 1> size_t cl::sycl::detail::buffer< T, Dimensions >::get_size ( ) const [inline]

Returns the size of the buffer storage in bytes.

Equal to get_count()∗sizeof(T).

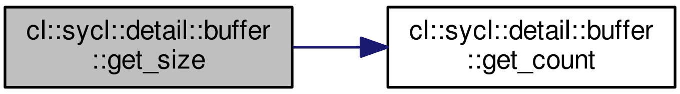
Todo rename to something else. In `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf` it is named bytes() for example

Definition at line 217 of file buffer.hpp.

References cl::sycl::detail::buffer< T, Dimensions >::get_count().

```
                          {
    return get_count()*sizeof(T);
  }
```

Here is the call graph for this function:


cl::sycl::detail::buffer ::get_size
cl::sycl::detail::buffer ::get_count

8.1.2.5.3.6 template<typename T, std::size_t Dimensions = 1> void cl::sycl::detail::buffer< T, Dimensions >::set_final_data (weak_ptr_class< T > && *finalData*) [inline]

Set the weak pointer to copy back data on buffer deletion.

Todo Add a write kernel dependency on the buffer so the buffer destructor has to wait for the kernel execution if the buffer is also accessed through a write accessor

Definition at line 228 of file buffer.hpp.

```
                                                         {
    final_data = finalData;
  }
```

8.1.2.5.4 Friends And Related Function Documentation

8.1.2.5.4.1 template<typename T, std::size_t Dimensions = 1> template<typename U , std::size_t D, access::mode Mode, access::target Target> friend class detail::accessor [friend]

Definition at line 59 of file buffer.hpp.

8.1.2.5.5 Member Data Documentation

8.1.2.5.5.1 template<typename T, std::size_t Dimensions = 1> boost::multi_array_ref<T, Dimensions> cl::sycl::detail::buffer< T, Dimensions >::access [private]

This is the multi-dimensional interface to the data that may point to either allocation in the case of storage managed by SYCL itself or to some other memory location in the case of host memory or storage<> abstraction use.

Definition at line 67 of file buffer.hpp.

8.1.2.5.5.2 template<typename T, std::size_t Dimensions = 1> boost::multi_array<T, Dimensions> cl::sycl::detail::buffer< T, Dimensions >::allocation [private]

If some allocation is requested, it is managed by this multi_array to ease initialization from data.

Definition at line 51 of file buffer.hpp.

8.1.2.5.5.3 template<typename T, std::size_t Dimensions = 1> weak_ptr_class<T> cl::sycl::detail::buffer< T, Dimensions >::final_data [private]

The weak pointer to copy back data on buffer deletion.

Definition at line 70 of file buffer.hpp.

8.1.2.5.5.4 template<typename T, std::size_t Dimensions = 1> bool cl::sycl::detail::buffer< T, Dimensions >::host_write_back = false [private]

Definition at line 77 of file buffer.hpp.

8.1.2.5.5.5 template<typename T, std::size_t Dimensions = 1> shared_ptr_class<T> cl::sycl::detail::buffer< T, Dimensions >::shared_data [private]

The shared pointer in the case the buffer memory is shared with the host.

Definition at line 74 of file buffer.hpp.

Referenced by cl::sycl::detail::buffer< T, Dimensions >::get_destructor_future().

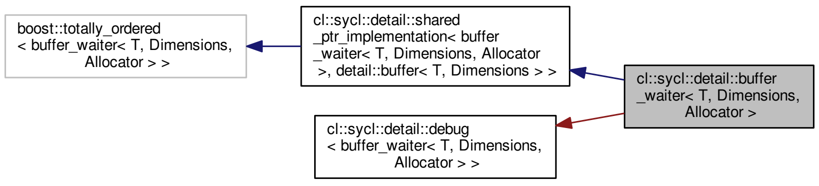
8.1.2.6 class cl::sycl::detail::buffer_waiter

**template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>>
class cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >**

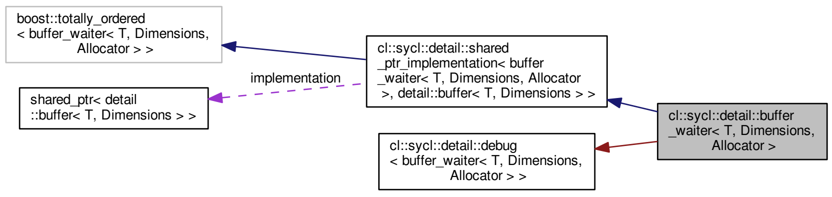
A helper class to wait for the final buffer destruction if the conditions for blocking are met.

Definition at line 36 of file buffer_waiter.hpp.

Inheritance diagram for cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >:


boost::totally_ordered < buffer_waiter< T, Dimensions, Allocator > >
cl::sycl::detail::shared _ptr_implementation< buffer _waiter< T, Dimensions, Allocator >, detail::buffer< T, Dimensions > >
cl::sycl::detail::buffer _waiter< T, Dimensions, Allocator >
cl::sycl::detail::debug < buffer_waiter< T, Dimensions, Allocator > >

Collaboration diagram for cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >:


boost::totally_ordered < buffer_waiter< T, Dimensions, Allocator > >
shared_ptr< detail ::buffer< T, Dimensions > >
implementation
cl::sycl::detail::shared _ptr_implementation< buffer _waiter< T, Dimensions, Allocator >, detail::buffer< T, Dimensions > >
cl::sycl::detail::buffer _waiter< T, Dimensions, Allocator >
cl::sycl::detail::debug < buffer_waiter< T, Dimensions, Allocator > >

Public Member Functions

- buffer_waiter (detail::buffer< T, Dimensions > ∗b)

    Create a new buffer_waiter on top of a detail::buffer.
- ∼buffer_waiter ()

    The buffer_waiter destructor waits for any data to be written back to the host, if any.

Private Types

- using implementation_t = detail::shared_ptr_implementation< buffer_waiter< T, Dimensions, Allocator >, detail::buffer< T, Dimensions >>

Additional Inherited Members

8.1.2.6.1 Member Typedef Documentation

8.1.2.6.1.1 template<typename T , std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> using **cl::sycl::detail::buffer_waiter**< T, Dimensions, Allocator >**::implementation_t** = **detail::shared_ptr_implementation**<**buffer_waiter**<T, Dimensions, Allocator>, **detail::buffer**<T, Dimensions>> [private]

Definition at line 46 of file buffer_waiter.hpp.

8.1.2.6.2 Constructor & Destructor Documentation

8.1.2.6.2.1 template<typename T , std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> **cl::sycl::detail::buffer_waiter**< T, Dimensions, Allocator >::**buffer_waiter** (**detail::buffer**< T, Dimensions > ∗ ***b***) [inline]

Create a new buffer_waiter on top of a detail::buffer.

Definition at line 54 of file buffer_waiter.hpp.

```
00054 : implementation_t { b } {}
```

8.1.2.6.2.2 template<typename T , std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> **cl::sycl::detail::buffer_waiter**< T, Dimensions, Allocator >::∼**buffer_waiter** ( ) [inline]

The buffer_waiter destructor waits for any data to be written back to the host, if any.

Definition at line 60 of file buffer_waiter.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer_waiter< T, Dimensions, Allocator >, detail::buffer< T, Dimensions > >::implementation, and TRISYCL_DUMP_T.

```
00060                    {
00061     /* Get a future from the implementation if we have to wait for its
00062        destruction */
00063     auto f = implementation->get_destructor_future();
00064     if (f) {
00065       /* No longer carry for the implementation buffer which is free to
00066          live its life up to its destruction */
00067       implementation.reset();
00068       TRISYCL_DUMP_T("~buffer_waiter() is waiting");
00069       // Then wait for its end in some other thread
00070       f->wait();
00071       TRISYCL_DUMP_T("~buffer_waiter() is done");
00072     }
00073   }
```

8.1.2.7 struct cl::sycl::image

template<std::size_t dimensions>
struct cl::sycl::image< dimensions >

Todo implement image

Definition at line 23 of file image.hpp.

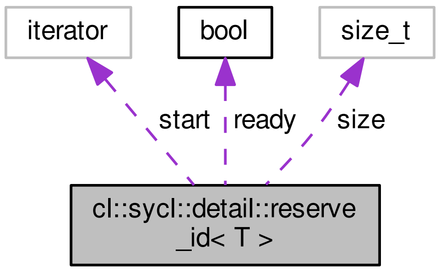
8.1.2.8 struct cl::sycl::detail::reserve_id

template<typename T>
struct cl::sycl::detail::reserve_id< T >

A private description of a reservation station.

Definition at line 40 of file pipe.hpp.

Collaboration diagram for cl::sycl::detail::reserve_id< T >:



Public Member Functions

- reserve_id (typename boost::circular_buffer< T >::iterator start, std::size_t size)

  Track a reservation not committed yet.

Public Attributes

- boost::circular_buffer< T >::iterator start

  Start of the reservation in the pipe storage.
- std::size_t size

  Number of elements in the reservation.
- bool ready = false

8.1.2.8.1 Constructor & Destructor Documentation

8.1.2.8.1.1 template<typename T > cl::sycl::detail::reserve_id< T >::reserve_id (typename boost::circular_buffer< T >::iterator *start*, std::size_t *size*) [inline]

Track a reservation not committed yet.

Parameters

in	*start*	point to the start of the reservation in the pipe storage
in	*size*	is the number of elements in the reservation

Definition at line 58 of file pipe.hpp.

```
00059                               : start { start }, size { size } {}
```

8.1.2.8.2 Member Data Documentation

8.1.2.8.2.1 template<typename T > bool cl::sycl::detail::reserve_id< T >::ready = false

Definition at line 49 of file pipe.hpp.

8.1.2.8.2.2 template<typename T > std::size_t cl::sycl::detail::reserve_id< T >::size

Number of elements in the reservation.

Definition at line 45 of file pipe.hpp.

Referenced by cl::sycl::detail::pipe< value_type >::empty(), cl::sycl::detail::pipe< value_type >::reserve_read(), cl::sycl::detail::pipe< value_type >::reserve_write(), and cl::sycl::detail::pipe< value_type >::size_with_lock().

8.1.2.8.2.3 template<typename T > boost::circular_buffer<T>::iterator cl::sycl::detail::reserve_id< T >::start

Start of the reservation in the pipe storage.

Definition at line 42 of file pipe.hpp.

8.1.2.9 class cl::sycl::detail::pipe

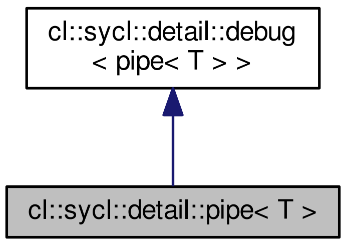
template<typename T>
class cl::sycl::detail::pipe< T >

Implement a pipe object.

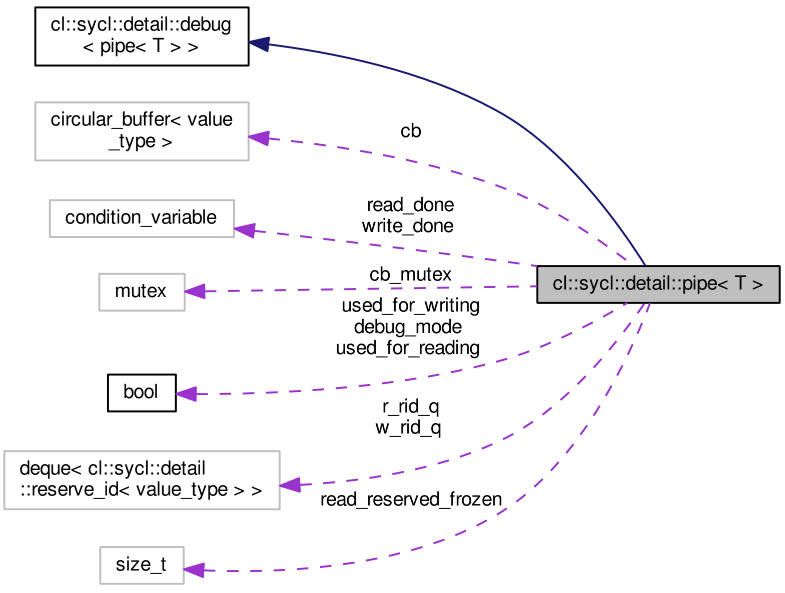
Use some mutable members so that the pipe object can be changed even when the accessors are captured in a lambda.

Definition at line 70 of file pipe.hpp.

Inheritance diagram for cl::sycl::detail::pipe< T >:


cl::sycl::detail::debug < pipe< T > >
cl::sycl::detail::pipe< T >

Collaboration diagram for cl::sycl::detail::pipe< T >:


cl::sycl::detail::debug < pipe< T > >
circular_buffer< value _type >
condition_variable
mutex
bool
deque< cl::sycl::detail ::reserve_id< value_type > >
size_t
cl::sycl::detail::pipe< T >
cb
read_done
write_done
cb_mutex
used_for_writing
debug_mode
used_for_reading
r_rid_q
w_rid_q
read_reserved_frozen

Public Types

- using value_type = T
- using implementation_t = boost::circular_buffer< value_type >

    Implement the pipe with a circular buffer.
- using rid_iterator = typename decltype(w_rid_q)::iterator

Public Member Functions

- pipe (std::size_t capacity)

    Create a pipe as a circular buffer of the required capacity.
- std::size_t capacity () const

    Return the maximum number of elements that can fit in the pipe.
- std::size_t size_with_lock () const

    The size() method used outside needs to lock the datastructure.
- bool empty_with_lock () const

    The empty() method used outside needs to lock the datastructure.
- bool full_with_lock () const
- bool write (const T &value, bool blocking=false)

    Try to write a value to the pipe.
- bool read (T &value, bool blocking=false)

    Try to read a value from the pipe.
- std::size_t reserved_for_reading () const

    Compute the amount of elements blocked by read reservations, not yet committed.
- std::size_t reserved_for_writing () const

    Compute the amount of elements blocked by write reservations, not yet committed.

- bool reserve_read (std::size_t s, rid_iterator &rid, bool blocking=false)
  Reserve some part of the pipe for reading.
- bool reserve_write (std::size_t s, rid_iterator &rid, bool blocking=false)
  Reserve some part of the pipe for writing.
- void move_read_reservation_forward ()
  Process the read reservations that are ready to be released in the reservation queue.
- void move_write_reservation_forward ()
  Process the write reservations that are ready to be released in the reservation queue.

Public Attributes

- bool used_for_reading = false
  True when the pipe is currently used for reading.
- bool used_for_writing = false
  True when the pipe is currently used for writing.

Private Member Functions

- std::size_t size () const
  Get the current number of elements in the pipe that can be read.
- bool empty () const
  Test if the pipe is empty.
- bool full () const
  Test if the pipe is full.

Private Attributes

- boost::circular_buffer< value_type > cb
  The circular buffer to store the elements.
- std::mutex cb_mutex
  To protect the access to the circular buffer.
- std::deque< reserve_id< value_type > > w_rid_q
  The queue of pending write reservations.
- std::deque< reserve_id< value_type > > r_rid_q
  The queue of pending read reservations.
- std::size_t read_reserved_frozen
  Track the number of frozen elements related to read reservations.
- std::condition_variable read_done
  To signal that a read has been successful.
- std::condition_variable write_done
  To signal that a write has been successful.
- bool debug_mode = false
  To control the debug mode, disabled by default.

8.1.2.9.1 Member Typedef Documentation

8.1.2.9.1.1 template<typename T> using cl::sycl::detail::pipe< T >::implementation_t = boost::circular_buffer<value_type>

Implement the pipe with a circular buffer.

Definition at line 77 of file pipe.hpp.

8.1.2.9.1.2 template<typename T> using cl::sycl::detail::pipe< T >::rid_iterator = typename decltype(w_rid_q)::iterator

Definition at line 95 of file pipe.hpp.

8.1.2.9.1.3 template<typename T> using cl::sycl::detail::pipe< T >::value_type = T

Definition at line 74 of file pipe.hpp.

8.1.2.9.2 Constructor & Destructor Documentation

8.1.2.9.2.1 template<typename T> cl::sycl::detail::pipe< T >::pipe (std::size_t *capacity*) [inline]

Create a pipe as a circular buffer of the required capacity.

Definition at line 123 of file pipe.hpp.

```
: cb { capacity }, read_reserved_frozen { 0 } { }
```

8.1.2.9.3 Member Function Documentation

8.1.2.9.3.1 template<typename T> std::size_t cl::sycl::detail::pipe< T >::capacity ( ) const [inline]

Return the maximum number of elements that can fit in the pipe.

Definition at line 128 of file pipe.hpp.

```
                          {
    // No lock required since it is fixed and set at construction time
    return cb.capacity();
  }
```

8.1.2.9.3.2 template<typename T> bool cl::sycl::detail::pipe< T >::empty ( ) const [inline], [private]

Test if the pipe is empty.

This is obviously a volatile value which is constrained by restricted relativity.

Note that on some devices it may be costly to implement on the write side (for example on FPGA).

Definition at line 163 of file pipe.hpp.

```
                   {
    TRISYCL_DUMP_T("empty() cb.size() = " << cb.size()
                   << " size() = " << size());
    // It is empty when the size is zero, taking into account reservations
    return size() == 0;
  }
```

8.1.2.9.3.3 template<typename T> bool cl::sycl::detail::pipe< T >::empty_with_lock ( ) const [inline]

The empty() method used outside needs to lock the datastructure.

Definition at line 194 of file pipe.hpp.

```
                                  {
    std::lock_guard<std::mutex> lg { cb_mutex };
    return empty();
  }
```

8.1.2.9.3.4 template<typename T> bool cl::sycl::detail::pipe< T >::full ( ) const [inline], [private]

Test if the pipe is full.

This is obviously a volatile value which is constrained by restricted relativity.

Note that on some devices it may be costly to implement on the read side (for example on FPGA).

Definition at line 179 of file pipe.hpp.

```
                       {
    return cb.full();
  }
```

8.1.2.9.3.5 template<typename T> bool cl::sycl::detail::pipe< T >::full_with_lock ( ) const [inline]

Definition at line 201 of file pipe.hpp.

```
                                 {
    std::lock_guard<std::mutex> lg { cb_mutex };
    return full();
  }
```

8.1.2.9.3.6 template<typename T> void cl::sycl::detail::pipe< T >::move_read_reservation_forward ( ) [inline]

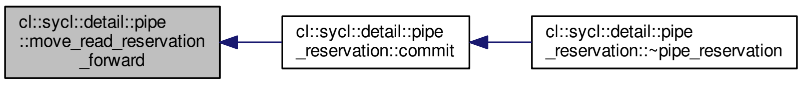
Process the read reservations that are ready to be released in the reservation queue.

Definition at line 422 of file pipe.hpp.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::commit().

```
                                          {
    // Lock the pipe to avoid nuisance
    std::lock_guard<std::mutex> lg { cb_mutex };

    for (;;) {
      if (r_rid_q.empty())
        // No pending reservation, so nothing to do
        break;
      if (!r_rid_q.front().ready)
        /* If the first reservation is not ready to be released, stop
           because it is blocking all the following in the queue
           anyway */
        break;
      // Remove the reservation to be released from the queue
      r_rid_q.pop_front();
      std::size_t n_to_pop;
      if (r_rid_q.empty())
        // If it was the last one, remove all the reservation
        n_to_pop = read_reserved_frozen;
      else
        // Else remove everything up to the next reservation
        n_to_pop =  r_rid_q.front().start - cb.begin();
      // No longer take into account these reserved slots
      read_reserved_frozen -= n_to_pop;
      // Release the elements from the FIFO
      while (n_to_pop--)
        cb.pop_front();
      // Notify the clients waiting for some room to write in the pipe
      read_done.notify_all();
      /* ...and process the next reservation to see if it is ready to
         be released too */
    }
  }
```

Here is the caller graph for this function:


cl::sycl::detail::pipe ::move_read_reservation _forward
cl::sycl::detail::pipe _reservation::commit
cl::sycl::detail::pipe _reservation::~pipe_reservation

8.1.2.9.3.7 template<typename T> void cl::sycl::detail::pipe< T >::move_write_reservation_forward ( ) [inline]

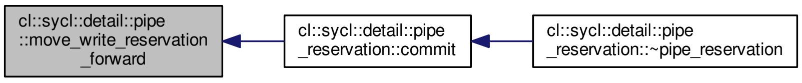
Process the write reservations that are ready to be released in the reservation queue.

Definition at line 460 of file pipe.hpp.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::commit().

```
                                                    {
    // Lock the pipe to avoid nuisance
    std::lock_guard<std::mutex> lg { cb_mutex };

    for (;;) {
      if (w_rid_q.empty())
        // No pending reservation, so nothing to do
        break;
      // Get the first reservation
      const auto &rid = w_rid_q.front();
      if (!rid.ready)
        /* If the reservation is not ready to be released, stop
           because it is blocking all the following in the queue
           anyway */
        break;
      // Remove the reservation to be released from the queue
      w_rid_q.pop_front();
      // Notify the clients waiting to read something from the pipe
      write_done.notify_all();
      /* ...and process the next reservation to see if it is ready to
         be released too */
    }
  }
```

Here is the caller graph for this function:


cl::sycl::detail::pipe ::move_write_reservation _forward
cl::sycl::detail::pipe _reservation::commit
cl::sycl::detail::pipe _reservation::~pipe_reservation

8.1.2.9.3.8 template<typename T> bool cl::sycl::detail::pipe< T >::read (T & *value*, bool *blocking* = false) [inline]

Try to read a value from the pipe.

Parameters

out	*value*	is the reference to where to store what is read
in	*blocking*	specify if the call wait for the operation to succeed

Returns

true on success

If there is a pending reservation, read the next element to be read and update the number of reserved elements

Definition at line 255 of file pipe.hpp.

```
                                              {
      // Lock the pipe to avoid being disturbed
      std::unique_lock<std::mutex> ul { cb_mutex };
      TRISYCL_DUMP_T("Read pipe empty = " << empty());

      if (blocking)
        /* If in blocking mode, wait for the not empty condition, that
           may be changed when a write is done */
        write_done.wait(ul, [&] { return !empty(); });
      else if (empty())
        return false;

      TRISYCL_DUMP_T("Read pipe front = " << cb.front()
                     << " back = " << cb.back()
                     << " reserved_for_reading() = " << reserved_for_reading());
      if (read_reserved_frozen)
        /** If there is a pending reservation, read the next element to
            be read and update the number of reserved elements */
        value = cb.begin()[read_reserved_frozen++];
      else {
        /* There is no pending read reservation, so pop the read value
           from the pipe */
        value = cb.front();
        cb.pop_front();
      }

      TRISYCL_DUMP_T("Read pipe value = " << value);
      // Notify the clients waiting for some room to write in the pipe
      read_done.notify_all();
      return true;
    }
```

8.1.2.9.3.9 template<typename T> bool cl::sycl::detail::pipe< T >::reserve_read (std::size_t *s*, rid_iterator & *rid*, bool *blocking* = false) [inline]

Reserve some part of the pipe for reading.

Parameters

in	*s*	is the number of element to reserve
out	*rid*	is an iterator to a description of the reservation that has been done if successful
in	*blocking*	specify if the call wait for the operation to succeed

Returns

true if the reservation was successful

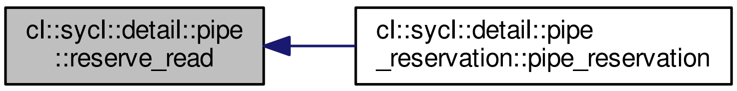
Definition at line 332 of file pipe.hpp.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation().

```
                                             {
      // Lock the pipe to avoid being disturbed
      std::unique_lock<std::mutex> ul { cb_mutex };

      TRISYCL_DUMP_T("Before read reservation cb.size() = " << cb.size()
                     << " size() = " << size());
```

```
    if (s == 0)
      // Empty reservation requested, so nothing to do
      return false;

    if (blocking)
      /* If in blocking mode, wait for enough elements to read in the
         pipe for the reservation. This condition can change when a
         write is done */
      write_done.wait(ul, [&] { return s <= size(); });
    else if (s > size())
      // Not enough elements to read in the pipe for the reservation
      return false;

    // Compute the location of the first element of the reservation
    auto first = cb.begin() + read_reserved_frozen;
    // Increment the number of frozen elements
    read_reserved_frozen += s;
    /* Add a description of the reservation at the end of the
       reservation queue */
    r_rid_q.emplace_back(first, s);
    // Return the iterator to the last reservation descriptor
    rid = r_rid_q.end() - 1;
    TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
                   << " size() = " << size());
    return true;
  }
```

Here is the caller graph for this function:


cl::sycl::detail::pipe ::reserve_read
cl::sycl::detail::pipe _reservation::pipe_reservation

8.1.2.9.3.10 template<typename T> bool cl::sycl::detail::pipe< T >::reserve_write (std::size_t *s*, rid_iterator & *rid*, bool *blocking* = false) [inline]

Reserve some part of the pipe for writing.

Parameters

in	*s*	is the number of element to reserve
out	*rid*	is an iterator to a description of the reservation that has been done if successful
in	*blocking*	specify if the call wait for the operation to succeed

Returns

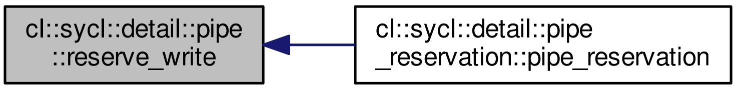
true if the reservation was successful

Definition at line 380 of file pipe.hpp.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation().

```
                                             {
    // Lock the pipe to avoid being disturbed
    std::unique_lock<std::mutex> ul { cb_mutex };

    TRISYCL_DUMP_T("Before write reservation cb.size() = " << cb.size()
                   << " size() = " << size());
    if (s == 0)
      // Empty reservation requested, so nothing to do
      return false;

    if (blocking)
      /* If in blocking mode, wait for enough room in the pipe, that
         may be changed when a read is done. Do not use a difference
         here because it is only about unsigned values */
      read_done.wait(ul, [&] { return cb.size() + s <= capacity(); });
    else if (cb.size() + s > capacity())
      // Not enough room in the pipe for the reservation
      return false;

    /* If there is enough room in the pipe, just create default values
         in it to do the reservation */
    for (std::size_t i = 0; i != s; ++i)
      cb.push_back();
    /* Compute the location of the first element a posteriori since it
         may not exist a priori if cb was empty before */
    auto first = cb.end() - s;
    /* Add a description of the reservation at the end of the
       reservation queue */
    w_rid_q.emplace_back(first, s);
    // Return the iterator to the last reservation descriptor
    rid = w_rid_q.end() - 1;
    TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
                   << " size() = " << size());
    return true;
  }
```

Here is the caller graph for this function:


cl::sycl::detail::pipe ::reserve_write
cl::sycl::detail::pipe _reservation::pipe_reservation

8.1.2.9.3.11 template<typename T> std::size_t cl::sycl::detail::pipe< T >::reserved_for_reading ( ) const [inline]

Compute the amount of elements blocked by read reservations, not yet committed.

This includes some normal reads to pipes between/after un-committed reservations

This function assumes that the data structure is locked

Definition at line 296 of file pipe.hpp.

```
                                         {
    return read_reserved_frozen;
  }
```

8.1.2.9.3.12 template<typename T> std::size_t cl::sycl::detail::pipe< T >::reserved_for_writing ( ) const [inline]

Compute the amount of elements blocked by write reservations, not yet committed.

This includes some normal writes to pipes between/after un-committed reservations

This function assumes that the data structure is locked

Definition at line 309 of file pipe.hpp.

```
                                         {
    if (w_rid_q.empty())
      // No on-going reservation
      return 0;
    else
      /* The reserved size is from the first element of the first
         on-going reservation up to the end of the pipe content */
      return cb.end() - w_rid_q.front().start;
  }
```

8.1.2.9.3.13 template<typename T> std::size_t cl::sycl::detail::pipe< T >::size ( ) const [inline], [private]

Get the current number of elements in the pipe that can be read.

This is obviously a volatile value which is constrained by the theory of restricted relativity.

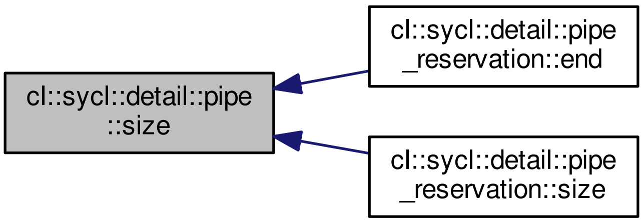
Note that on some devices it may be costly to implement (for example on FPGA).

Definition at line 143 of file pipe.hpp.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::end(), and cl::sycl::detail::pipe_reservation< PipeAccessor >::size().

```
                          {
    TRISYCL_DUMP_T("size() cb.size() = " << cb.size()
                   << " cb.end() = " << (void *)&*cb.end()
                   << " reserved_for_reading() = " << reserved_for_reading()
                   << " reserved_for_writing() = " << reserved_for_writing());
    /* The actual number of available elements depends from the
       elements blocked by some reservations.
       This prevents a consumer to read into reserved area. */
    return cb.size() - reserved_for_reading() -
reserved_for_writing();
  }
```

Here is the caller graph for this function:


cl::sycl::detail::pipe ::size
cl::sycl::detail::pipe _reservation::end
cl::sycl::detail::pipe _reservation::size

8.1.2.9.3.14 template<typename T> std::size_t cl::sycl::detail::pipe< T >::size_with_lock ( ) const [inline]

The size() method used outside needs to lock the datastructure.

Definition at line 187 of file pipe.hpp.

```
                                    {
    std::lock_guard<std::mutex> lg { cb_mutex };
    return size();
  }
```

8.1.2.9.3.15 template<typename T> bool cl::sycl::detail::pipe< T >::write (const T & *value*, bool *blocking* = false) [inline]

Try to write a value to the pipe.

Parameters

in	*value*	is what we want to write
in	*blocking*	specify if the call wait for the operation to succeed

Returns

true on success

Todo provide a && version

Definition at line 218 of file pipe.hpp.

```
                                                       {
    // Lock the pipe to avoid being disturbed
    std::unique_lock<std::mutex> ul { cb_mutex };
    TRISYCL_DUMP_T("Write pipe full = " << full()
                   << " value = " << value);

    if (blocking)
      /* If in blocking mode, wait for the not full condition, that
         may be changed when a read is done */
      read_done.wait(ul, [&] { return !full(); });
    else if (full())
      return false;

    cb.push_back(value);
    TRISYCL_DUMP_T("Write pipe front = " << cb.front()
                   << " back = " << cb.back()
                   << " cb.begin() = " << (void *)&*cb.begin()
                   << " cb.size() = " << cb.size()
                   << " cb.end() = " << (void *)&*cb.end()
                   << " reserved_for_reading() = " << reserved_for_reading()
                   << " reserved_for_writing() = " << reserved_for_writing());
    // Notify the clients waiting to read something from the pipe
    write_done.notify_all();
    return true;
  }
```

8.1.2.9.4 Member Data Documentation

8.1.2.9.4.1 template<typename T> boost::circular_buffer<value_type> cl::sycl::detail::pipe< T >::cb [private]

The circular buffer to store the elements.

Definition at line 82 of file pipe.hpp.

8.1.2.9.4.2 template<typename T> std::mutex **cl::sycl::detail::pipe**< T >::cb_mutex `[mutable]`, `[private]`

To protect the access to the circular buffer.

In case the object is capture in a lambda per copy, make it mutable.

Definition at line 88 of file pipe.hpp.

8.1.2.9.4.3 template<typename T> bool **cl::sycl::detail::pipe**< T >::debug_mode = false `[private]`

To control the debug mode, disabled by default.

Definition at line 112 of file pipe.hpp.

8.1.2.9.4.4 template<typename T> std::deque<**reserve_id**<**value_type**> > **cl::sycl::detail::pipe**< T >::r_rid_q `[private]`

The queue of pending read reservations.

Definition at line 100 of file pipe.hpp.

8.1.2.9.4.5 template<typename T> std::condition_variable **cl::sycl::detail::pipe**< T >::read_done `[private]`

To signal that a read has been successful.

Definition at line 106 of file pipe.hpp.

8.1.2.9.4.6 template<typename T> std::size_t **cl::sycl::detail::pipe**< T >::read_reserved_frozen `[private]`

Track the number of frozen elements related to read reservations.

Definition at line 103 of file pipe.hpp.

8.1.2.9.4.7 template<typename T> bool **cl::sycl::detail::pipe**< T >::used_for_reading = false

True when the pipe is currently used for reading.

Definition at line 117 of file pipe.hpp.

8.1.2.9.4.8 template<typename T> bool **cl::sycl::detail::pipe**< T >::used_for_writing = false

True when the pipe is currently used for writing.

Definition at line 120 of file pipe.hpp.

8.1.2.9.4.9 template<typename T> std::deque<**reserve_id**<**value_type**> > **cl::sycl::detail::pipe**< T >::w_rid_q `[private]`

The queue of pending write reservations.

Definition at line 91 of file pipe.hpp.

8.1.2.9.4.10 template< typename T > std::condition_variable cl::sycl::detail::pipe< T >::write_done [private]

To signal that a write has been successful.

Definition at line 109 of file pipe.hpp.

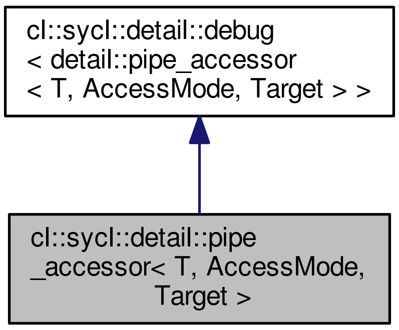
8.1.2.10 class cl::sycl::detail::pipe_accessor

template< typename T, access::mode AccessMode, access::target Target >
class cl::sycl::detail::pipe_accessor< T, AccessMode, Target >

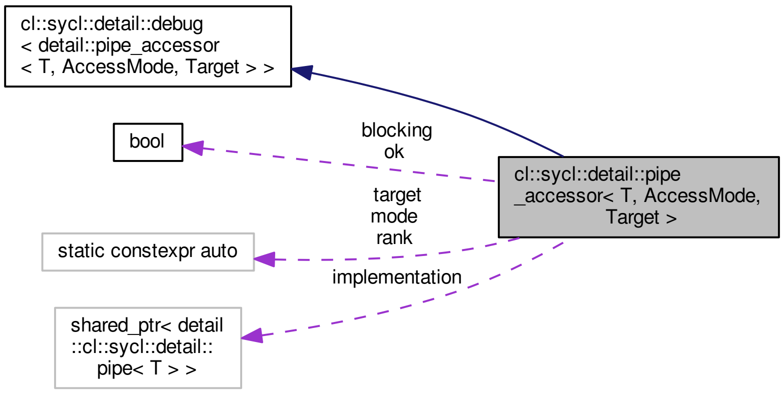
The accessor abstracts the way pipe data are accessed inside a kernel.

Definition at line 44 of file pipe_accessor.hpp.

Inheritance diagram for cl::sycl::detail::pipe_accessor< T, AccessMode, Target >:


cl::sycl::detail::debug < detail::pipe_accessor < T, AccessMode, Target > >
cl::sycl::detail::pipe _accessor< T, AccessMode, Target >

Collaboration diagram for cl::sycl::detail::pipe_accessor< T, AccessMode, Target >:


cl::sycl::detail::debug < detail::pipe_accessor < T, AccessMode, Target > >
bool
blocking
ok
cl::sycl::detail::pipe _accessor< T, AccessMode, Target >
target
mode
rank
static constexpr auto
implementation
shared_ptr< detail ::cl::sycl::detail:: pipe< T > >

Public Types

- using value_type = T

  The STL-like types.
- using reference = value_type &
- using const_reference = const value_type &

Public Member Functions

- pipe_accessor (const std::shared_ptr< detail::pipe< T >> &p, handler &command_group_handler)

  Construct a pipe accessor from an existing pipe.
- pipe_accessor ()=default
- std::size_t capacity () const

  Return the maximum number of elements that can fit in the pipe.
- std::size_t size () const

  Get the current number of elements in the pipe.
- bool empty () const

  Test if the pipe is empty.
- bool full () const

  Test if the pipe is full.
- operator bool () const

  In an explicit bool context, the accessor gives the success status of the last access.
- const pipe_accessor & write (const value_type &value) const

  Try to write a value to the pipe.
- const pipe_accessor & operator<< (const value_type &value) const

  Some syntactic sugar to use.
- const pipe_accessor & read (value_type &value) const

  Try to read a value from the pipe.
- value_type read () const

  Read a value from a blocking pipe.
- const pipe_accessor & operator>> (value_type &value) const

  Some syntactic sugar to use.
- detail::pipe_reservation< pipe_accessor > reserve (std::size_t size) const
- void set_debug (bool enable) const

  Set debug mode.
- auto & get_pipe_detail ()
- ∼pipe_accessor ()

Static Public Attributes

- static constexpr auto rank = 1
- static constexpr auto mode = AccessMode
- static constexpr auto target = Target
- static constexpr bool blocking

Private Attributes

- std::shared_ptr< detail::pipe< T > > implementation

  The real pipe implementation behind the hood.
- bool ok = false

  Store the success status of last pipe operation.

8.1.2.10.1 Member Typedef Documentation

8.1.2.10.1.1 template< typename T, access::mode AccessMode, access::target Target> using cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::const_reference = const value_type&

Definition at line 59 of file pipe_accessor.hpp.

8.1.2.10.1.2 template< typename T, access::mode AccessMode, access::target Target> using cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::reference = value_type&

Definition at line 58 of file pipe_accessor.hpp.

8.1.2.10.1.3 template< typename T, access::mode AccessMode, access::target Target> using cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::value_type = T

The STL-like types.

Definition at line 57 of file pipe_accessor.hpp.

8.1.2.10.2 Constructor & Destructor Documentation

8.1.2.10.2.1 template< typename T, access::mode AccessMode, access::target Target> cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::pipe_accessor (const std::shared_ptr< detail::pipe< T >> & *p*, handler & *command_group_handler*) [inline]

Construct a pipe accessor from an existing pipe.

Todo Use pipe_exception instead

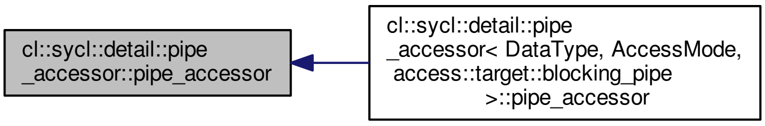
Definition at line 83 of file pipe_accessor.hpp.

```
                                            :
    implementation { p } {
    //    TRISYCL_DUMP_T("Create a kernel pipe accessor write = "
    //                   << is_write_access());
    // Verify that the pipe is not already used in the requested mode
    if (mode == access::mode::write)
      if (implementation->used_for_writing)
        /// \todo Use pipe_exception instead
        throw std::logic_error { "The pipe is already used for writing." };
      else
        implementation->used_for_writing = true;
    else
      if (implementation->used_for_reading)
        throw std::logic_error { "The pipe is already used for reading." };
      else
        implementation->used_for_reading = true;
  }
```

8.1.2.10.2.2 template<typename T, access::mode AccessMode, access::target Target> cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::pipe_accessor ( ) [default]

Referenced by cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >::pipe_↩ accessor().

Here is the caller graph for this function:


cl::sycl::detail::pipe _accessor::pipe_accessor
cl::sycl::detail::pipe _accessor< DataType, AccessMode, access::target::blocking_pipe >::pipe_accessor

8.1.2.10.2.3 template<typename T, access::mode AccessMode, access::target Target> cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::∼pipe_accessor ( ) [inline]

Free the pipe for a future usage for the current mode

Definition at line 272 of file pipe_accessor.hpp.

```
                     {
    /// Free the pipe for a future usage for the current mode
    if (mode == access::mode::write)
      implementation->used_for_writing = false;
    else
      implementation->used_for_reading = false;
  }
```

8.1.2.10.3 Member Function Documentation

8.1.2.10.3.1 template<typename T, access::mode AccessMode, access::target Target> std::size_t cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::capacity ( ) const [inline]

Return the maximum number of elements that can fit in the pipe.

Definition at line 107 of file pipe_accessor.hpp.

```
                              {
    return implementation->capacity();
  }
```

8.1.2.10.3.2 template<typename T, access::mode AccessMode, access::target Target> bool cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::empty ( ) const [inline]

Test if the pipe is empty.

This is obviously a volatile value which is constrained by restricted relativity.

Note that on some devices it may be costly to implement on the write side (for example on FPGA).

Definition at line 132 of file pipe_accessor.hpp.

```
                       {
    return implementation->empty_with_lock();
  }
```

8.1.2.10.3.3 template<typename T, access::mode AccessMode, access::target Target> bool cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::full ( ) const [inline]

Test if the pipe is full.

This is obviously a volatile value which is constrained by restricted relativity.

Note that on some devices it may be costly to implement on the read side (for example on FPGA).

Definition at line 145 of file pipe_accessor.hpp.

```
                      {
    return implementation->full_with_lock();
  }
```

8.1.2.10.3.4 template<typename T, access::mode AccessMode, access::target Target> auto& cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::get_pipe_detail ( ) [inline]

Definition at line 267 of file pipe_accessor.hpp.

```
                            {
    return implementation;
  }
```

8.1.2.10.3.5 template<typename T, access::mode AccessMode, access::target Target> cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::operator bool ( ) const [inline], [explicit]

In an explicit bool context, the accessor gives the success status of the last access.

It is not impacted by reservation success.

The explicitness is related to avoid

```
 some_pipe <<
some_value
```

to be interpreted as

```
 some_bool <<
some_value
```

when the type of

```
some_value
```

is not the same type as the pipe type.

Returns

true on success of the previous read or write operation

Definition at line 162 of file pipe_accessor.hpp.

```
                                    {
    return ok;
  }
```

8.1.2.10.3.6 template<typename T, access::mode AccessMode, access::target Target> const pipe_accessor& cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::operator<< (const value_type & *value*) const [inline]

Some syntactic sugar to use.

```
a << v
```

instead of

```
a.write(v)
```

Definition at line 192 of file pipe_accessor.hpp.

```
                                                                    {
    static_assert(mode == access::mode::write,
                  "'<<' operator on a pipe accessor is only possible"
                  " with write access mode");
    // Return a reference to *this so we can apply a sequence of >>
    return write(value);
  }
```

8.1.2.10.3.7 template<typename T, access::mode AccessMode, access::target Target> const pipe_accessor& cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::operator>> (value_type & *value*) const [inline]

Some syntactic sugar to use.

```
a >> v
```

instead of

```
a.read(v)
```

Definition at line 247 of file pipe_accessor.hpp.

```
                                                              {
    static_assert(mode == access::mode::read,
                  "'>>' operator on a pipe accessor is only possible"
                  " with read access mode");
    // Return a reference to *this so we can apply a sequence of >>
    return read(value);
  }
```

8.1.2.10.3.8 template<typename T, access::mode AccessMode, access::target Target> const pipe_accessor& cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::read (value_type & *value*) const [inline]

Try to read a value from the pipe.

Parameters

out	*value*	is the reference to where to store what is read

Returns

this
so we can apply a sequence of read for example (but do not do this on a non blocking pipe...)

This function is const so it can work when the accessor is passed by copy in the [=] kernel lambda, which is not mutable by default

Definition at line 213 of file pipe_accessor.hpp.

```
                                                     {
    static_assert(mode == access::mode::read,
                  "'.read(value_type &value)' method on a pipe accessor"
                  " is only possible with read access mode");
    ok = implementation->read(value, blocking);
    // Return a reference to *this so we can apply a sequence of read
    return *this;
  }
```

8.1.2.10.3.9 template<typename T, access::mode AccessMode, access::target Target> value_type cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::read ( ) const [inline]

Read a value from a blocking pipe.

Returns

the read value directly, since it cannot fail on blocking pipe

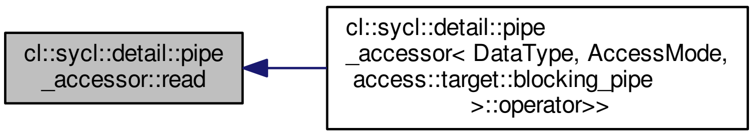
This function is const so it can work when the accessor is passed by copy in the [=] kernel lambda, which is not mutable by default

Definition at line 232 of file pipe_accessor.hpp.

Referenced by cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >↩
::operator>>().

```
                           {
    static_assert(mode == access::mode::read,
                  "'.read()' method on a pipe accessor is only possible"
                  " with read access mode");
    static_assert(blocking,
                  "'.read()' method on a pipe accessor is only possible"
                  " with a blocking pipe");
    value_type value;
    implementation->read(value, blocking);
    return value;
  }
```

Here is the caller graph for this function:



8.1.2.10.3.10 template<typename T, access::mode AccessMode, access::target Target> detail::pipe_reservation<pipe_accessor> cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::reserve (std::size_t *size*) const [inline]

Definition at line 256 of file pipe_accessor.hpp.

```
                                                                    {
    return { *implementation, size };
  }
```

8.1.2.10.3.11 template<typename T, access::mode AccessMode, access::target Target> void cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::set_debug (bool *enable*) const [inline]

Set debug mode.

Definition at line 262 of file pipe_accessor.hpp.

```
                                   {
    implementation->debug_mode = enable;
  }
```

8.1.2.10.3.12 template<typename T, access::mode AccessMode, access::target Target> std::size_t cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::size ( ) const [inline]

Get the current number of elements in the pipe.

This is obviously a volatile value which is constrained by restricted relativity.

Note that on some devices it may be costly to implement (for example on FPGA).

Definition at line 119 of file pipe_accessor.hpp.

```
                         {
    return implementation->size_with_lock();
  }
```

8.1.2.10.3.13 template<typename T, access::mode AccessMode, access::target Target> const pipe_accessor& cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::write (const value_type & *value*) const [inline]

Try to write a value to the pipe.

Parameters

in	*value*	is what we want to write

Returns

this so we can apply a sequence of write for example (but do not do this on a non blocking pipe...)

Todo provide a && version

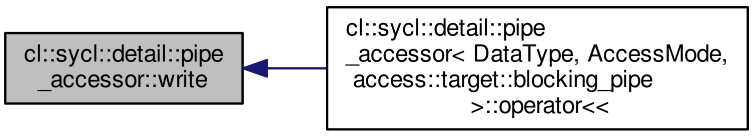
This function is const so it can work when the accessor is passed by copy in the [=] kernel lambda, which is not mutable by default

Definition at line 180 of file pipe_accessor.hpp.

Referenced by cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >↩
::operator<<().

```
                                                                {
    static_assert(mode == access::mode::write,
                  "'.write(const value_type &value)' method on a pipe accessor"
                  " is only possible with write access mode");
    ok = implementation->write(value, blocking);
    // Return a reference to *this so we can apply a sequence of write
    return *this;
  }
```

Here is the caller graph for this function:



8.1.2.10.4 Member Data Documentation

8.1.2.10.4.1 template<typename T, access::mode AccessMode, access::target Target> constexpr bool cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::blocking [static]

Initial value:

```
=
    (target == cl::sycl::access::target::blocking_pipe)
```

Definition at line 53 of file pipe_accessor.hpp.

8.1.2.10.4.2 template<typename T, access::mode AccessMode, access::target Target> std::shared_ptr<detail::pipe<T> > cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::implementation [private]

The real pipe implementation behind the hood.

Definition at line 64 of file pipe_accessor.hpp.

Referenced by cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >::get↩
_pipe_detail(), and cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >↩
::reserve().

8.1.2.10.4.3 template<typename T, access::mode AccessMode, access::target Target> constexpr auto cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::mode = AccessMode [static]

Definition at line 50 of file pipe_accessor.hpp.

8.1.2.10.4.4 template<typename T, access::mode AccessMode, access::target Target> bool cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::ok = false [mutable], [private]

Store the success status of last pipe operation.

It is not impacted by reservation success.

It does exist even if the pipe accessor is not evaluated in a boolean context for, but a use-def analysis can optimise it out in that case and not use some storage

Use a mutable state here so that it can work with a [=] lambda capture without having to declare the whole lambda as mutable

Definition at line 77 of file pipe_accessor.hpp.

Referenced by cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >::operator bool().

8.1.2.10.4.5 template<typename T, access::mode AccessMode, access::target Target> constexpr auto cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::rank = 1 [static]

Definition at line 49 of file pipe_accessor.hpp.

8.1.2.10.4.6 template<typename T, access::mode AccessMode, access::target Target> constexpr auto cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::target = Target [static]

Definition at line 51 of file pipe_accessor.hpp.

8.1.2.11 class cl::sycl::pipe

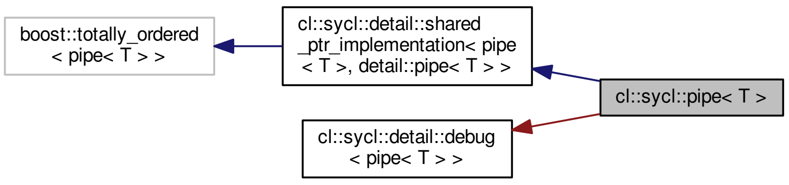
template<typename T>
class cl::sycl::pipe< T >

A SYCL pipe.

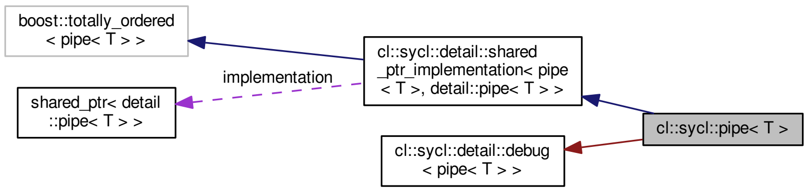
Implement a FIFO-style object that can be used through accessors to send some objects T from the input to the output

Definition at line 29 of file accessor.hpp.

Inheritance diagram for cl::sycl::pipe< T >:


boost::totally_ordered < pipe< T > >
cl::sycl::detail::shared_ptr_implementation< pipe < T >, detail::pipe< T > >
cl::sycl::pipe< T >
cl::sycl::detail::debug < pipe< T > >

Collaboration diagram for cl::sycl::pipe< T >:


boost::totally_ordered < pipe< T > >
shared_ptr< detail ::pipe< T > >
implementation
cl::sycl::detail::shared _ptr_implementation< pipe < T >, detail::pipe< T > >
cl::sycl::pipe< T >
cl::sycl::detail::debug < pipe< T > >

Public Types

- using value_type = T

  The STL-like types.

Public Member Functions

- pipe (std::size_t capacity)

  Construct a pipe able to store up to capacity T objects.
- template<access::mode Mode, access::target Target = access::target::pipe>
  accessor< value_type, 1, Mode, Target > get_access (handler &command_group_handler)

  Get an accessor to the pipe with the required mode.
- std::size_t capacity () const

  Return the maximum number of elements that can fit in the pipe.

Private Types

- using implementation_t = detail::shared_ptr_implementation< pipe< T >, detail::pipe< T >>

Additional Inherited Members

8.1.2.11.1 Member Typedef Documentation

8.1.2.11.1.1 template<typename T> using **cl::sycl::pipe< T >::implementation_t = detail::shared_ptr_implementation<pipe<T>, detail::pipe<T>>** [private]

Definition at line 41 of file pipe.hpp.

8.1.2.11.1.2 template<typename T> using **cl::sycl::pipe< T >::value_type = T**

The STL-like types.

Definition at line 51 of file pipe.hpp.

8.1.2.11.2 Constructor & Destructor Documentation

8.1.2.11.2.1 template<typename T> cl::sycl::pipe< T >::pipe (std::size_t *capacity*) [inline]

Construct a pipe able to store up to capacity T objects.

Definition at line 55 of file pipe.hpp.

References cl::sycl::access::pipe.

```
    : implementation_t { new detail::pipe<T> { capacity } } { }
```

8.1.2.11.3 Member Function Documentation

8.1.2.11.3.1 template<typename T> std::size_t cl::sycl::pipe< T >::capacity ( ) const [inline]

Return the maximum number of elements that can fit in the pipe.

Definition at line 81 of file pipe.hpp.

References cl::sycl::detail::shared_ptr_implementation< pipe< T >, detail::pipe< T > >::implementation.

```
                              {
    return implementation->capacity();
    }
```

8.1.2.11.3.2 template<typename T> template<access::mode Mode, access::target Target = access::target::pipe> accessor<value_type, 1, Mode, Target> cl::sycl::pipe< T >::get_access (handler & *command_group_handler*) [inline]

Get an accessor to the pipe with the required mode.

Parameters

	Mode	is the requested access mode
	Target	is the type of pipe access required
in	*command_group_handler*	is the command group handler in which the kernel is to be executed

Definition at line 71 of file pipe.hpp.

References cl::sycl::access::blocking_pipe, cl::sycl::detail::shared_ptr_implementation< pipe< T >, detail::pipe< T > >::implementation, and cl::sycl::access::pipe.

```
                                              {
    static_assert(Target == access::target::pipe
                  || Target == access::target::blocking_pipe,
                  "get_access(handler) with pipes can only deal with "
                  "access::pipe or access::blocking_pipe");
    return { implementation, command_group_handler };
  }
```

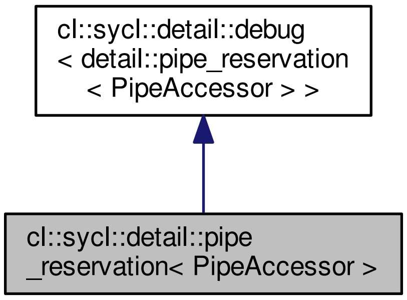
8.1.2.12 class cl::sycl::detail::pipe_reservation

template<typename PipeAccessor>
class cl::sycl::detail::pipe_reservation< PipeAccessor >

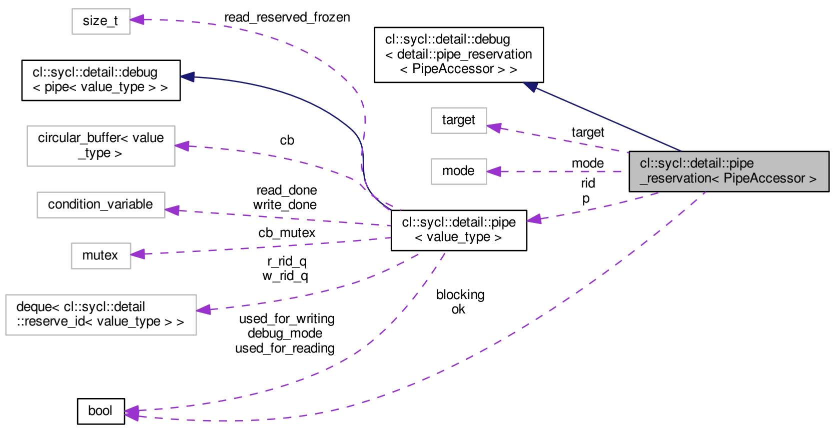
The implementation of the pipe reservation station.

Definition at line 33 of file pipe_reservation.hpp.

Inheritance diagram for cl::sycl::detail::pipe_reservation< PipeAccessor >:


cl::sycl::detail::debug < detail::pipe_reservation < PipeAccessor > >
cl::sycl::detail::pipe _reservation< PipeAccessor >

Collaboration diagram for cl::sycl::detail::pipe_reservation< PipeAccessor >:


size_t
read_reserved_frozen
cl::sycl::detail::debug < pipe< value_type > >
cl::sycl::detail::debug < detail::pipe_reservation < PipeAccessor > >
circular_buffer< value _type >
cb
target
target
mode
mode
rid
p
cl::sycl::detail::pipe _reservation< PipeAccessor >
condition_variable
read_done
write_done
cl::sycl::detail::pipe < value_type >
mutex
cb_mutex
r_rid_q
w_rid_q
deque< cl::sycl::detail ::reserve_id< value_type > >
blocking
ok
used_for_writing
debug_mode
used_for_reading
bool

Public Types

- using iterator = typename detail::pipe< value_type >::implementation_t::iterator
- using const_iterator = typename detail::pipe< value_type >::implementation_t::const_iterator

Public Member Functions

- void assume_validity ()

  Test that the reservation is in a usable state.
- pipe_reservation (detail::pipe< value_type > &p, std::size_t s)

  Create a pipe reservation station that reserves the pipe itself.
- pipe_reservation (const pipe_reservation &)=delete

  No copy constructor with some spurious commit in the destructor of the original object.
- pipe_reservation (pipe_reservation &&orig)

  Only a move constructor is required to move it into the shared_ptr.
- pipe_reservation ()=default

  Keep the default constructors too.
- operator bool ()

  Test if the reservation succeeded and thus if the reservation can be committed.
- iterator begin ()

  Start of the reservation area.
- iterator end ()

  Past the end of the reservation area.
- std::size_t size ()

  Get the number of elements in the reservation station.
- reference operator[ ] (std::size_t index)

  Access to an element of the reservation.
- void commit ()

  Commit the reservation station.
- ∼pipe_reservation ()

  An implicit commit is made in the destructor.

Public Attributes

- bool ok = false

  True if the reservation was successful and still uncommitted.
- detail::pipe< value_type >::rid_iterator rid

  Point into the reservation buffer. Only valid if ok is true.
- detail::pipe< value_type > & p

  Keep a reference on the pipe to access to the data and methods.

Static Public Attributes

- static constexpr access::mode mode = accessor_type::mode
- static constexpr access::target target = accessor_type::target

Private Types

- using accessor_type = PipeAccessor
- using value_type = typename accessor_type::value_type
- using reference = typename accessor_type::reference

Static Private Attributes

- static constexpr bool blocking

8.1.2.12.1 Member Typedef Documentation

8.1.2.12.1.1 template<typename PipeAccessor> using cl::sycl::detail::pipe_reservation< PipeAccessor >::accessor_type = PipeAccessor [private]

Definition at line 35 of file pipe_reservation.hpp.

8.1.2.12.1.2 template<typename PipeAccessor> using cl::sycl::detail::pipe_reservation< PipeAccessor >::const_iterator = typename detail::pipe<value_type>::implementation_t::const_iterator

Definition at line 46 of file pipe_reservation.hpp.

8.1.2.12.1.3 template<typename PipeAccessor> using cl::sycl::detail::pipe_reservation< PipeAccessor >::iterator = typename detail::pipe<value_type>::implementation_t::iterator

Definition at line 44 of file pipe_reservation.hpp.

8.1.2.12.1.4 template<typename PipeAccessor> using cl::sycl::detail::pipe_reservation< PipeAccessor >::reference = typename accessor_type::reference [private]

Definition at line 39 of file pipe_reservation.hpp.

8.1.2.12.1.5 template<typename PipeAccessor> using cl::sycl::detail::pipe_reservation< PipeAccessor >::value_type = typename accessor_type::value_type [private]

Definition at line 38 of file pipe_reservation.hpp.

8.1.2.12.2 Constructor & Destructor Documentation

8.1.2.12.2.1 template<typename PipeAccessor> cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation (detail::pipe< value_type > & *p*, std::size_t *s*) [inline]

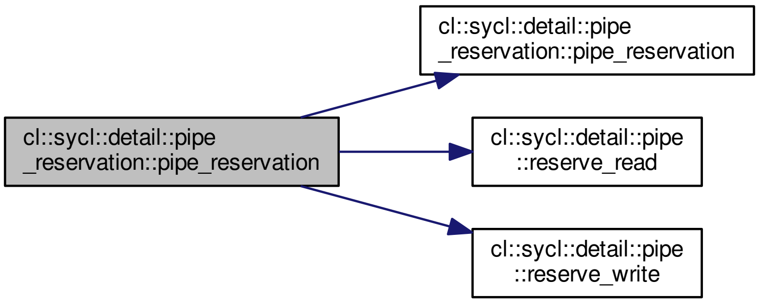
Create a pipe reservation station that reserves the pipe itself.

Definition at line 78 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation(), cl::sycl::access::read, cl::sycl::detail::pipe< T >::reserve_read(), cl::sycl::detail::pipe< T >::reserve_write(), and cl::sycl::access::write.

```
                                                          : p { p } {
    static_assert(mode == access::mode::write
                  || mode == access::mode::read,
                  "A pipe can only be accesed in read or write mode,"
                  " exclusively");

    /* Since this test is constexpr and dependent of a template
       parameter, it should be equivalent to a specialization of the
       method but in a clearer way */
    if (mode == access::mode::write)
      ok = p.reserve_write(s, rid, blocking);
    else
      ok = p.reserve_read(s, rid, blocking);
  }
```

Here is the call graph for this function:



8.1.2.12.2.2 template<typename PipeAccessor> cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation (const pipe_reservation< PipeAccessor > &) [delete]

No copy constructor with some spurious commit in the destructor of the original object.

8.1.2.12.2.3 template<typename PipeAccessor> cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation (pipe_reservation< PipeAccessor > && *orig*) [inline]

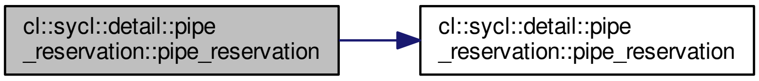
Only a move constructor is required to move it into the shared_ptr.

Definition at line 101 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation().

```
                                            :
    ok {orig.ok },
    rid {orig.rid },
    p { orig.p } {
      /* Even when an object is moved, the destructor of the old
         object is eventually called, so leave the old object in a
         destructable state but without any commit capability */
      orig.ok = false;
    }
```

Here is the call graph for this function:


cl::sycl::detail::pipe
_reservation::pipe_reservation
cl::sycl::detail::pipe
_reservation::pipe_reservation

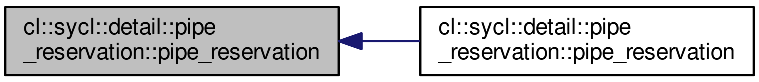
8.1.2.12.2.4 template<typename PipeAccessor> cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation ( ) [default]

Keep the default constructors too.

Otherwise there is no move semantics and the copy is made by creating a new reservation and destructing the old one with a spurious commit in the meantime...

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation().

Here is the caller graph for this function:


cl::sycl::detail::pipe
_reservation::pipe_reservation
cl::sycl::detail::pipe
_reservation::pipe_reservation

8.1.2.12.2.5 template<typename PipeAccessor> cl::sycl::detail::pipe_reservation< PipeAccessor >::∼pipe_reservation ( ) [inline]

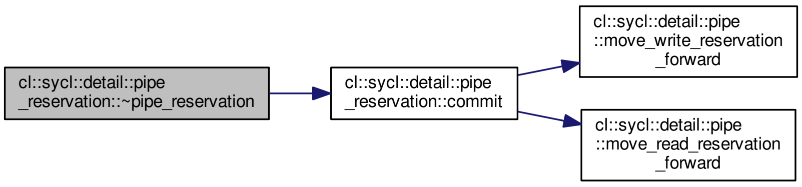
An implicit commit is made in the destructor.

Definition at line 185 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::commit().

```
                        {
    commit();
  }
```

Here is the call graph for this function:



8.1.2.12.3 Member Function Documentation

8.1.2.12.3.1 template<typename PipeAccessor> void cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity ( ) [inline]

Test that the reservation is in a usable state.

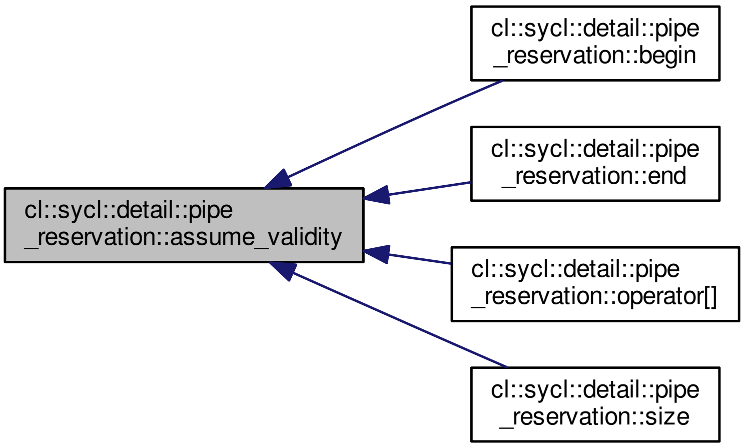
Todo Throw exception instead

Definition at line 71 of file pipe_reservation.hpp.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::begin(), cl::sycl::detail::pipe_reservation< PipeAccessor >::end(), cl::sycl::detail::pipe_reservation< PipeAccessor >::operator[ ](), and cl::sycl::detail↩ ::pipe_reservation< PipeAccessor >::size().

```
                          {
    assert(ok);
  }
```

Here is the caller graph for this function:


cl::sycl::detail::pipe _reservation::assume_validity
cl::sycl::detail::pipe _reservation::begin
cl::sycl::detail::pipe _reservation::end
cl::sycl::detail::pipe _reservation::operator[]
cl::sycl::detail::pipe _reservation::size

8.1.2.12.3.2 template<typename PipeAccessor> iterator cl::sycl::detail::pipe_reservation< PipeAccessor >::begin ( ) [inline]

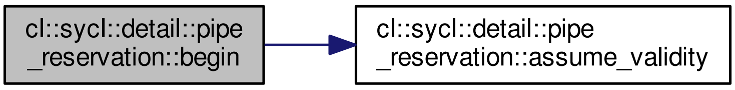
Start of the reservation area.

Definition at line 134 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity().

```
                   {
    assume_validity();
    return rid->start;
  }
```

Here is the call graph for this function:


cl::sycl::detail::pipe _reservation::begin
cl::sycl::detail::pipe _reservation::assume_validity

8.1.2.12.3.3 template< typename PipeAccessor > void cl::sycl::detail::pipe_reservation< PipeAccessor >::commit ( ) [inline]

Commit the reservation station.

Todo Add to the specification that for simplicity a reservation can be commited several times but only the first one is taken into account

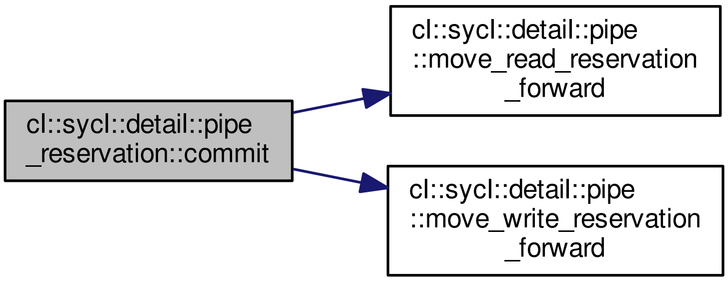
Definition at line 170 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe< T >::move_read_reservation_forward(), cl::sycl::detail::pipe< T >::move_write↩_reservation_forward(), TRISYCL_DUMP_T, and cl::sycl::access::write.

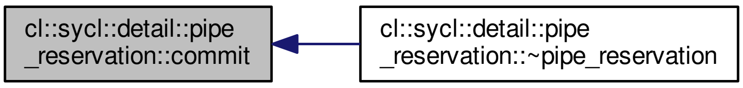
Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::∼pipe_reservation().

```
                  {
    if (ok) {
      // If the reservation is in a committable state, commit
      TRISYCL_DUMP_T("Commit");
      rid->ready = true;
      if (mode == access::mode::write)
        p.move_write_reservation_forward();
      else
        p.move_read_reservation_forward();
      ok = false;
    }
  }
```

Here is the call graph for this function:


cl::sycl::detail::pipe _reservation::commit
cl::sycl::detail::pipe ::move_read_reservation _forward
cl::sycl::detail::pipe ::move_write_reservation _forward

Here is the caller graph for this function:


cl::sycl::detail::pipe _reservation::commit
cl::sycl::detail::pipe _reservation::~pipe_reservation

8.1.2.12.3.4 template<typename PipeAccessor> iterator cl::sycl::detail::pipe_reservation< PipeAccessor >::end ( ) [inline]

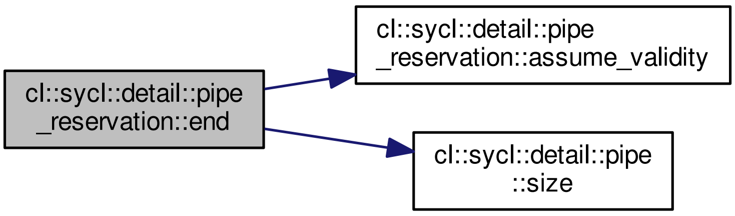
Past the end of the reservation area.

Definition at line 141 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity(), and cl::sycl::detail::pipe< T >↩
::size().

```
                  {
    assume_validity();
    return rid->start + rid->size;
  }
```

Here is the call graph for this function:


cl::sycl::detail::pipe _reservation::end
cl::sycl::detail::pipe _reservation::assume_validity
cl::sycl::detail::pipe ::size

8.1.2.12.3.5 template<typename PipeAccessor> cl::sycl::detail::pipe_reservation< PipeAccessor >::operator bool ( ) [inline]

Test if the reservation succeeded and thus if the reservation can be committed.

Note that it is up to the user to ensure that all the reservation elements have been initialized correctly in the case of a write for example

Definition at line 128 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::ok.

```
                   {
    return ok;
  }
```

8.1.2.12.3.6 template<typename PipeAccessor> reference cl::sycl::detail::pipe_reservation< PipeAccessor >::operator[] (std::size_t *index*) [inline]

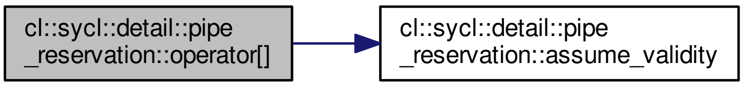
Access to an element of the reservation.

Definition at line 155 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity(), and TRISYCL_DUMP_T.

```
                                        {
    assume_validity();
    TRISYCL_DUMP_T("[] index = " << index
                   << " Reservation write address = " << &(rid->start[index]));

    return rid->start[index];
  }
```

Here is the call graph for this function:


cl::sycl::detail::pipe _reservation::operator[]
cl::sycl::detail::pipe _reservation::assume_validity

8.1.2.12.3.7 template<typename PipeAccessor> std::size_t cl::sycl::detail::pipe_reservation< PipeAccessor >::size ( ) [inline]

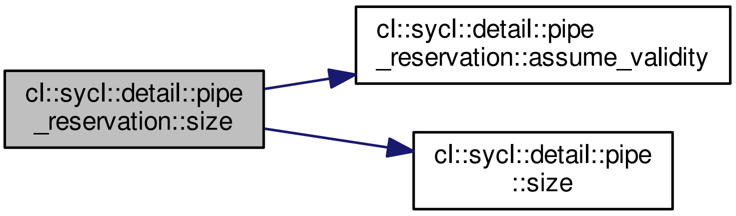
Get the number of elements in the reservation station.

Definition at line 148 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity(), and cl::sycl::detail::pipe< T >↩ ::size().

```
                   {
    assume_validity();
    return rid->size;
  }
```

Here is the call graph for this function:


cl::sycl::detail::pipe _reservation::size
cl::sycl::detail::pipe _reservation::assume_validity
cl::sycl::detail::pipe ::size

8.1.2.12.4 Member Data Documentation

8.1.2.12.4.1 template<typename PipeAccessor> constexpr bool cl::sycl::detail::pipe_reservation< PipeAccessor >::blocking [static], [private]

Initial value:

```
=
    (accessor_type::target ==
      cl::sycl::access::target::blocking_pipe)
```

Definition at line 36 of file pipe_reservation.hpp.

8.1.2.12.4.2 template<typename PipeAccessor> constexpr access::mode cl::sycl::detail::pipe_reservation< PipeAccessor >::mode = accessor_type::mode [static]

Definition at line 49 of file pipe_reservation.hpp.

8.1.2.12.4.3 template<typename PipeAccessor> bool cl::sycl::detail::pipe_reservation< PipeAccessor >::ok = false

True if the reservation was successful and still uncommitted.

B default a pipe_reservation is not reserved and cannot be committed

Definition at line 55 of file pipe_reservation.hpp.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::operator bool().

8.1.2.12.4.4 template<typename PipeAccessor> detail::pipe<value_type>& cl::sycl::detail::pipe_reservation< PipeAccessor >::p

Keep a reference on the pipe to access to the data and methods.

Note that with inlining and CSE it should not use more register when compiler optimization is in use.

Definition at line 64 of file pipe_reservation.hpp.

8.1.2.12.4.5 template<typename PipeAccessor> detail::pipe<value_type>::rid_iterator cl::sycl::detail::pipe_reservation< PipeAccessor >::rid

Point into the reservation buffer. Only valid if ok is true.

Definition at line 58 of file pipe_reservation.hpp.

8.1.2.12.4.6 template<typename PipeAccessor> constexpr access::target cl::sycl::detail::pipe_reservation< PipeAccessor >::target = accessor_type::target [static]

Definition at line 50 of file pipe_reservation.hpp.

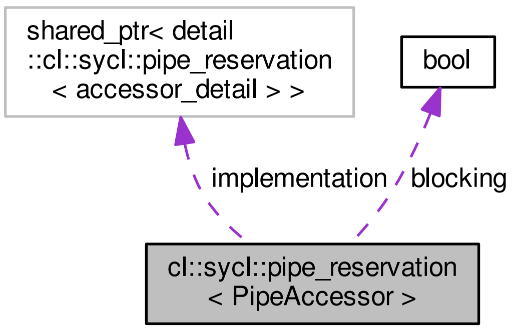
8.1.2.13 struct cl::sycl::pipe_reservation

template<typename PipeAccessor>
struct cl::sycl::pipe_reservation< PipeAccessor >

The pipe reservation station allows to reserve an array-like view inside the pipe for ordered race-free access from various work-items for example.

Definition at line 30 of file pipe_reservation.hpp.

Collaboration diagram for cl::sycl::pipe_reservation< PipeAccessor >:


shared_ptr< detail ::cl::sycl::pipe_reservation < accessor_detail > >
bool
implementation
blocking
cl::sycl::pipe_reservation < PipeAccessor >

Public Types

- using accessor_type = PipeAccessor
- using accessor_detail = typename accessor_type::accessor_detail
- using value_type = typename accessor_type::value_type

    The STL-like types.
- using reference = value_type &
- using const_reference = const value_type &
- using pointer = value_type ∗
- using const_pointer = const value_type ∗
- using size_type = std::size_t
- using difference_type = ptrdiff_t
- using iterator = typename detail::pipe_reservation< accessor_detail >::iterator
- using const_iterator = typename detail::pipe_reservation< accessor_detail >::const_iterator
- using reverse_iterator = std::reverse_iterator< iterator >
- using const_reverse_iterator = std::reverse_iterator< const_iterator >

Public Member Functions

- pipe_reservation ()=default

  Use default constructors so that we can create a new buffer copy from another one, with either a l-value or a r-value (for std::move() for example).
- pipe_reservation (accessor_type &accessor, std::size_t s)

  Create a pipe_reservation for an accessor and a number of elements.
- pipe_reservation (detail::pipe_reservation< accessor_detail > &&pr)

  Create a pipe_reservation from the implementation detail.
- operator bool () const

  Test if the pipe_reservation has been correctly allocated.
- std::size_t size () const

  Get the number of reserved element(s)
- reference operator[ ] (std::size_t index) const

  Access to a given element of the reservation.
- void commit () const

  Force a commit operation.
- iterator begin () const

  Get an iterator on the first element of the reservation station.
- iterator end () const

  Get an iterator past the end of the reservation station.
- const_iterator cbegin () const

  Build a constant iterator on the first element of the reservation station.
- const_iterator cend () const

  Build a constant iterator past the end of the reservation station.
- reverse_iterator rbegin () const

  Get a reverse iterator on the last element of the reservation station.
- reverse_iterator rend () const

  Get a reverse iterator on the first element past the end of the reservation station.
- const_reverse_iterator crbegin () const

  Get a constant reverse iterator on the last element of the reservation station.
- const_reverse_iterator crend () const

  Get a constant reverse iterator on the first element past the end of the reservation station.

Public Attributes

- std::shared_ptr< detail::pipe_reservation< accessor_detail > > implementation

  Point to the underlying implementation that can be shared in the SYCL model with a handler semantics.

Static Public Attributes

- static constexpr bool blocking

8.1.2.13.1 Member Typedef Documentation

8.1.2.13.1.1 template< typename PipeAccessor > using cl::sycl::pipe_reservation< PipeAccessor >::accessor_detail = typename accessor_type::accessor_detail

Definition at line 34 of file pipe_reservation.hpp.

8.1.2.13.1.2 template<typename PipeAccessor > using cl::sycl::pipe_reservation< PipeAccessor >::accessor_type = PipeAccessor

Definition at line 31 of file pipe_reservation.hpp.

8.1.2.13.1.3 template<typename PipeAccessor > using cl::sycl::pipe_reservation< PipeAccessor >::const_iterator = typename detail::pipe_reservation<accessor_detail>::const_iterator

Definition at line 46 of file pipe_reservation.hpp.

8.1.2.13.1.4 template<typename PipeAccessor > using cl::sycl::pipe_reservation< PipeAccessor >::const_pointer = const value_type*

Definition at line 40 of file pipe_reservation.hpp.

8.1.2.13.1.5 template<typename PipeAccessor > using cl::sycl::pipe_reservation< PipeAccessor >::const_reference = const value_type&

Definition at line 38 of file pipe_reservation.hpp.

8.1.2.13.1.6 template<typename PipeAccessor > using cl::sycl::pipe_reservation< PipeAccessor >::const_reverse_iterator = std::reverse_iterator<const_iterator>

Definition at line 48 of file pipe_reservation.hpp.

8.1.2.13.1.7 template<typename PipeAccessor > using cl::sycl::pipe_reservation< PipeAccessor >::difference_type = ptrdiff_t

Definition at line 42 of file pipe_reservation.hpp.

8.1.2.13.1.8 template<typename PipeAccessor > using cl::sycl::pipe_reservation< PipeAccessor >::iterator = typename detail::pipe_reservation<accessor_detail>::iterator

Definition at line 44 of file pipe_reservation.hpp.

8.1.2.13.1.9 template<typename PipeAccessor > using cl::sycl::pipe_reservation< PipeAccessor >::pointer = value_type*

Definition at line 39 of file pipe_reservation.hpp.

8.1.2.13.1.10 template<typename PipeAccessor > using cl::sycl::pipe_reservation< PipeAccessor >::reference = value_type&

Definition at line 37 of file pipe_reservation.hpp.

8.1.2.13.1.11 template<typename PipeAccessor > using cl::sycl::pipe_reservation< PipeAccessor >::reverse_iterator = std::reverse_iterator<iterator>

Definition at line 47 of file pipe_reservation.hpp.

8.1.2.13.1.12 template< typename PipeAccessor > using cl::sycl::pipe_reservation< PipeAccessor >::size_type = std::size_t

Definition at line 41 of file pipe_reservation.hpp.

8.1.2.13.1.13 template< typename PipeAccessor > using cl::sycl::pipe_reservation< PipeAccessor >::value_type = typename accessor_type::value_type

The STL-like types.

Definition at line 36 of file pipe_reservation.hpp.

8.1.2.13.2 Constructor & Destructor Documentation

8.1.2.13.2.1 template< typename PipeAccessor > cl::sycl::pipe_reservation< PipeAccessor >::pipe_reservation ( ) [default]

Use default constructors so that we can create a new buffer copy from another one, with either a l-value or a r-value (for std::move() for example).

Since we just copy the shared_ptr<> above, this is where/how the sharing magic is happening with reference counting in this case.

8.1.2.13.2.2 template< typename PipeAccessor > cl::sycl::pipe_reservation< PipeAccessor >::pipe_reservation (accessor_type & *accessor*, std::size_t *s*) [inline]

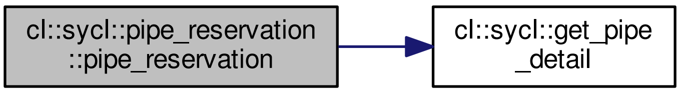
Create a pipe_reservation for an accessor and a number of elements.

Definition at line 66 of file pipe_reservation.hpp.

References cl::sycl::get_pipe_detail().

```
    : implementation {
    new detail::pipe_reservation<accessor_detail> {
      get_pipe_detail(accessor), s }
  } {}
```

Here is the call graph for this function:


cl::sycl::pipe_reservation ::pipe_reservation
cl::sycl::get_pipe _detail

8.1.2.13.2.3 template<typename PipeAccessor > cl::sycl::pipe_reservation< PipeAccessor >::pipe_reservation (detail::pipe_reservation< accessor_detail > && *pr*) [inline]

Create a pipe_reservation from the implementation detail.

This is an internal constructor to allow reserve() on the implementation to lift a full-fledged object through accessor↩ ::reserve().

Todo Make it private and add required friends

Definition at line 81 of file pipe_reservation.hpp.

```
00082     : implementation {
00083     new detail::pipe_reservation<accessor_detail> { std::move(pr) } }
00084   {}
```

8.1.2.13.3 Member Function Documentation

8.1.2.13.3.1 template<typename PipeAccessor > iterator cl::sycl::pipe_reservation< PipeAccessor >::begin ( ) const [inline]

Get an iterator on the first element of the reservation station.

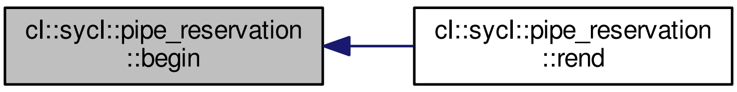
Definition at line 119 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

Referenced by cl::sycl::pipe_reservation< PipeAccessor >::rend().

```
00119                          {
00120     return implementation->begin();
00121   }
```

Here is the caller graph for this function:


cl::sycl::pipe_reservation ::begin
cl::sycl::pipe_reservation ::rend

8.1.2.13.3.2 template<typename PipeAccessor > const_iterator cl::sycl::pipe_reservation< PipeAccessor >::cbegin ( ) const [inline]

Build a constant iterator on the first element of the reservation station.

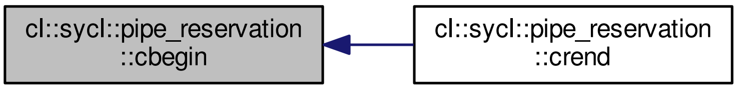
Definition at line 131 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

Referenced by cl::sycl::pipe_reservation< PipeAccessor >::crend().

```
                                   {
    return implementation->begin();
  }
```

Here is the caller graph for this function:


cl::sycl::pipe_reservation ::cbegin
cl::sycl::pipe_reservation ::crend

8.1.2.13.3.3 template<typename PipeAccessor > const_iterator cl::sycl::pipe_reservation< PipeAccessor >::cend ( ) const [inline]

Build a constant iterator past the end of the reservation station.

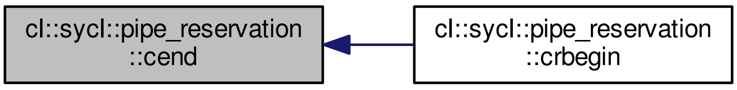
Definition at line 137 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

Referenced by cl::sycl::pipe_reservation< PipeAccessor >::crbegin().

```
                                 {
    return implementation->end();
  }
```

Here is the caller graph for this function:


cl::sycl::pipe_reservation ::cend
cl::sycl::pipe_reservation ::crbegin

8.1.2.13.3.4 template<typename PipeAccessor > void cl::sycl::pipe_reservation< PipeAccessor >::commit ( ) const [inline]

Force a commit operation.

Normally the commit is implicitly done in the destructor, but sometime it is useful to do it earlier.

Definition at line 113 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

```
                    {
    return implementation->commit();
  }
```

8.1.2.13.3.5 template<typename PipeAccessor > const_reverse_iterator cl::sycl::pipe_reservation< PipeAccessor >::crbegin ( ) const [inline]

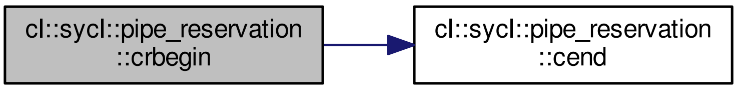
Get a constant reverse iterator on the last element of the reservation station.

Definition at line 157 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::cend().

```
                                       {
    return std::make_reverse_iterator(cend());
  }
```

Here is the call graph for this function:


cl::sycl::pipe_reservation ::crbegin
cl::sycl::pipe_reservation ::cend

8.1.2.13.3.6 template<typename PipeAccessor > const_reverse_iterator cl::sycl::pipe_reservation< PipeAccessor >::crend ( ) const [inline]

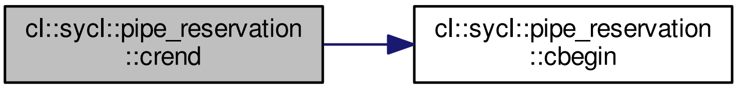
Get a constant reverse iterator on the first element past the end of the reservation station.

Definition at line 164 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::cbegin().

```
                                     {
    return std::make_reverse_iterator(cbegin());
  }
```

Here is the call graph for this function:


cl::sycl::pipe_reservation ::crend
cl::sycl::pipe_reservation ::cbegin

8.1.2.13.3.7 template<typename PipeAccessor > iterator cl::sycl::pipe_reservation< PipeAccessor >::end ( ) const [inline]

Get an iterator past the end of the reservation station.

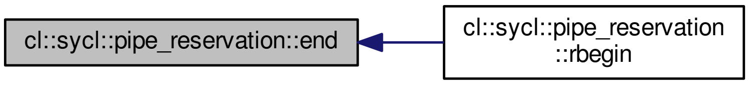
Definition at line 125 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

Referenced by cl::sycl::pipe_reservation< PipeAccessor >::rbegin().

```
                             {
    return implementation->end();
  }
```

Here is the caller graph for this function:


cl::sycl::pipe_reservation::end
cl::sycl::pipe_reservation ::rbegin

8.1.2.13.3.8 template<typename PipeAccessor > cl::sycl::pipe_reservation< PipeAccessor >::operator bool ( ) const [inline]

Test if the pipe_reservation has been correctly allocated.

Returns

true if the pipe_reservation can be used and committed

Definition at line 91 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

```
                              {
    return *implementation;
  }
```

8.1.2.13.3.9 template<typename PipeAccessor > reference cl::sycl::pipe_reservation< PipeAccessor >::operator[] (std::size_t *index*) const [inline]

Access to a given element of the reservation.

Definition at line 103 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

```
                                                  {
    return (*implementation)[index];
  }
```

8.1.2.13.3.10 template<typename PipeAccessor > reverse_iterator cl::sycl::pipe_reservation< PipeAccessor >::rbegin ( ) const [inline]

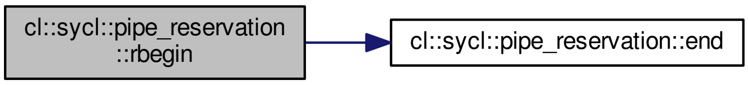
Get a reverse iterator on the last element of the reservation station.

Definition at line 143 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::end().

```
00143                                  {
00144     return std::make_reverse_iterator(end());
00145   }
```

Here is the call graph for this function:


cl::sycl::pipe_reservation ::rbegin
cl::sycl::pipe_reservation::end

8.1.2.13.3.11 template<typename PipeAccessor > reverse_iterator cl::sycl::pipe_reservation< PipeAccessor >::rend ( ) const [inline]

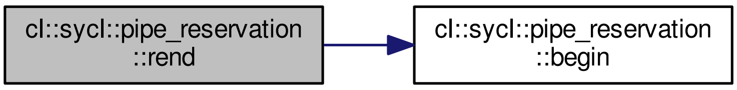
Get a reverse iterator on the first element past the end of the reservation station.

Definition at line 150 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::begin().

```
00150                                {
00151     return std::make_reverse_iterator(begin());
00152   }
```

Here is the call graph for this function:


cl::sycl::pipe_reservation ::rend
cl::sycl::pipe_reservation ::begin

8.1.2.13.3.12 template< typename PipeAccessor > std::size_t cl::sycl::pipe_reservation< PipeAccessor >::size ( ) const [inline]

Get the number of reserved element(s)

Definition at line 97 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

```
                              {
    return implementation->size();
  }
```

8.1.2.13.4 Member Data Documentation

8.1.2.13.4.1 template< typename PipeAccessor > constexpr bool cl::sycl::pipe_reservation< PipeAccessor >::blocking [static]

Initial value:

```
=
    (accessor_type::target ==
      cl::sycl::access::target::blocking_pipe)
```

Definition at line 32 of file pipe_reservation.hpp.

8.1.2.13.4.2 template< typename PipeAccessor > std::shared_ptr<detail::pipe_reservation<accessor_detail> > cl::sycl::pipe_reservation< PipeAccessor >::implementation

Point to the underlying implementation that can be shared in the SYCL model with a handler semantics.

Definition at line 53 of file pipe_reservation.hpp.

Referenced by cl::sycl::pipe_reservation< PipeAccessor >::begin(), cl::sycl::pipe_reservation< PipeAccessor >::cbegin(), cl::sycl::pipe_reservation< PipeAccessor >::cend(), cl::sycl::pipe_reservation< PipeAccessor >↩ ::commit(), cl::sycl::pipe_reservation< PipeAccessor >::end(), cl::sycl::pipe_reservation< PipeAccessor >↩ ::operator bool(), cl::sycl::pipe_reservation< PipeAccessor >::operator[ ](), and cl::sycl::pipe_reservation< Pipe↩ Accessor >::size().

8.1.2.14 class cl::sycl::static_pipe

template<typename T, std::size_t Capacity>
class cl::sycl::static_pipe< T, Capacity >

A SYCL static-scoped pipe equivalent to an OpenCL program-scoped pipe.

Implement a FIFO-style object that can be used through accessors to send some objects T from the input to the output.

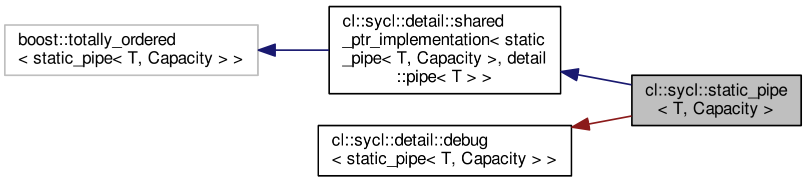
Compared to a normal pipe, a static_pipe takes a constexpr size and is expected to be declared in a compile-unit static context so the compiler can generate everything at compile time.

This is useful to generate a fixed and optimized hardware implementation on FPGA for example, where the interconnection graph can be also inferred at compile time.

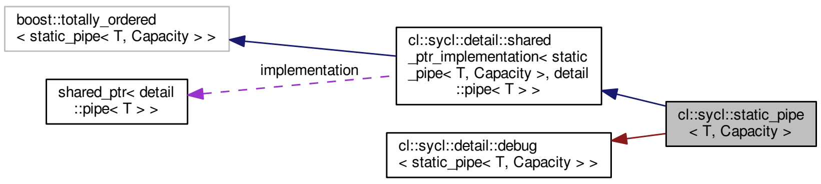
It is not directly mapped to the OpenCL program-scoped pipe because in SYCL there is not this concept of separated program. But the SYCL device compiler is expected to generate some OpenCL program(s) with program-scoped pipes when a SYCL static-scoped pipe is used. These details are implementation defined.

Definition at line 50 of file static_pipe.hpp.

Inheritance diagram for cl::sycl::static_pipe< T, Capacity >:


boost::totally_ordered < static_pipe< T, Capacity > >
cl::sycl::detail::shared _ptr_implementation< static _pipe< T, Capacity >, detail ::pipe< T > >
cl::sycl::static_pipe < T, Capacity >
cl::sycl::detail::debug < static_pipe< T, Capacity > >

Collaboration diagram for cl::sycl::static_pipe< T, Capacity >:


boost::totally_ordered < static_pipe< T, Capacity > >
shared_ptr< detail ::pipe< T > >
implementation
cl::sycl::detail::shared _ptr_implementation< static _pipe< T, Capacity >, detail ::pipe< T > >
cl::sycl::static_pipe < T, Capacity >
cl::sycl::detail::debug < static_pipe< T, Capacity > >

Public Types

- using value_type = T

  The STL-like types.

Public Member Functions

- static_pipe ()

  Construct a static-scoped pipe able to store up to Capacity T objects.
- template<access::mode Mode, access::target Target = access::target::pipe>
  accessor< value_type, 1, Mode, Target > get_access (handler &command_group_handler)

  Get an accessor to the pipe with the required mode.
- std::size_t constexpr capacity () const

  Return the maximum number of elements that can fit in the pipe.

Private Types

- using implementation_t = detail::shared_ptr_implementation< static_pipe< T, Capacity >, detail::pipe< T >>

Additional Inherited Members

8.1.2.14.1 Member Typedef Documentation

8.1.2.14.1.1 template<typename T , std::size_t Capacity> using cl::sycl::static_pipe< T, Capacity >::implementation_t = detail::shared_ptr_implementation<static_pipe<T, Capacity>, detail::pipe<T>> [private]

Definition at line 60 of file static_pipe.hpp.

8.1.2.14.1.2 template<typename T , std::size_t Capacity> using cl::sycl::static_pipe< T, Capacity >::value_type = T

The STL-like types.

Definition at line 68 of file static_pipe.hpp.

8.1.2.14.2 Constructor & Destructor Documentation

8.1.2.14.2.1 template<typename T , std::size_t Capacity> cl::sycl::static_pipe< T, Capacity >::static_pipe ( ) [inline]

Construct a static-scoped pipe able to store up to Capacity T objects.

Definition at line 72 of file static_pipe.hpp.

References cl::sycl::access::pipe.

```
00073     : implementation_t { new detail::pipe<T> { Capacity } } { }
```

8.1.2.14.3 Member Function Documentation

8.1.2.14.3.1 template<typename T , std::size_t Capacity> std::size_t constexpr cl::sycl::static_pipe< T, Capacity >::capacity ( ) const [inline]

Return the maximum number of elements that can fit in the pipe.

This is a constexpr since the capacity is in the type.

Definition at line 101 of file static_pipe.hpp.

```
00101                                          {
00102     return Capacity;
00103     }
```

8.1.2.14.3.2 template<typename T , std::size_t Capacity> template<access::mode Mode, access::target Target = access::target::pipe> accessor<value_type, 1, Mode, Target> cl::sycl::static_pipe< T, Capacity >::get_access (handler & *command_group_handler*) [inline]

Get an accessor to the pipe with the required mode.

Parameters

	Mode	is the requested access mode
	Target	is the type of pipe access required
in	*command_group_handler*	is the command group handler in which the kernel is to be executed

Definition at line 88 of file static_pipe.hpp.

References cl::sycl::access::blocking_pipe, cl::sycl::detail::shared_ptr_implementation< static_pipe< T, Capacity >, detail::pipe< T > >::implementation, and cl::sycl::access::pipe.

```
                                                    {
    static_assert(Target == access::target::pipe
                  || Target == access::target::blocking_pipe,
                  "get_access(handler) with pipes can only deal with "
                  "access::pipe or access::blocking_pipe");
    return { implementation, command_group_handler };
  }
```

8.1.3 Typedef Documentation

8.1.3.1 template<typename T > using **cl::sycl::buffer_allocator** = typedef std::allocator<T>

`#include <include/CL/sycl/buffer_allocator.hpp>`

The default buffer allocator used by the runtime, when no allocator is defined by the user.

Reuse the C++ default allocator.

Definition at line 28 of file buffer_allocator.hpp.

8.1.4 Function Documentation

8.1.4.1 template<typename BufferDetail > static std::shared_ptr<detail::task> cl::sycl::detail::buffer_add_to_task (BufferDetail *buf*, **handler** ∗ *command_group_handler*, **bool** *is_write_mode*) [static]

`#include <include/CL/sycl/buffer/detail/buffer.hpp>`

Proxy function to avoid some circular type recursion.

Returns

a shared_ptr<task>

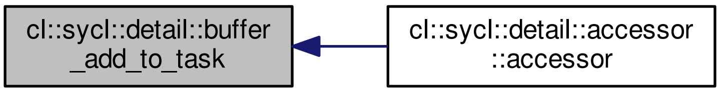
Todo To remove with some refactoring

Definition at line 281 of file buffer.hpp.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor().

```
                                    {
    return buf->add_to_task(command_group_handler, is_write_mode);
  }
```

Here is the caller graph for this function:


cl::sycl::detail::buffer_add_to_task
cl::sycl::detail::accessor::accessor

8.1.4.2 template<typename Accessor > static auto& cl::sycl::get_pipe_detail (Accessor & *a*) [inline], [static]

`#include <include/CL/sycl/accessor.hpp>`

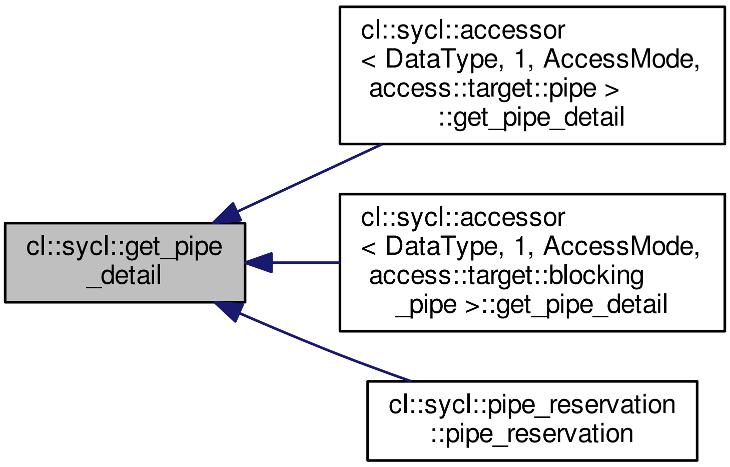
Top-level function to break circular dependencies on the the types to get the pipe implementation.

Definition at line 414 of file accessor.hpp.

Referenced by cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >::get_pipe_detail(), cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >::get_pipe_detail(), and cl::sycl::pipe_reservation< PipeAccessor >::pipe_reservation().

```
                                              {
  return a.get_pipe_detail();
  }
```

Here is the caller graph for this function:


cl::sycl::get_pipe_detail
cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >::get_pipe_detail
cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >::get_pipe_detail
cl::sycl::pipe_reservation::pipe_reservation

8.1.4.3 template<typename T , std::size_t Dimensions = 1> auto cl::sycl::detail::waiter (detail::buffer< T, Dimensions > * b) [inline]

```
#include <include/CL/sycl/buffer/detail/buffer_waiter.hpp>
```

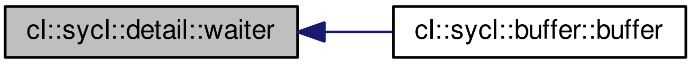
Helper function to create a new buffer_waiter.

Definition at line 80 of file buffer_waiter.hpp.

Referenced by cl::sycl::buffer< T, Dimensions, Allocator >::buffer().

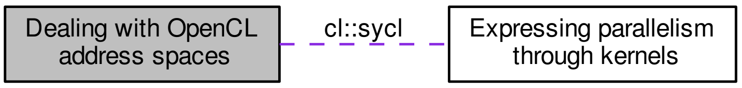
```
00080                                                    {
00081   return new buffer_waiter<T, Dimensions> { b };
00082 }
```

Here is the caller graph for this function:



8.2 Dealing with OpenCL address spaces

Collaboration diagram for Dealing with OpenCL address spaces:


Dealing with OpenCL address spaces
cl::sycl
Expressing parallelism through kernels

Namespaces

- cl::sycl

Classes

- struct cl::sycl::detail::opencl_type< T, AS >

  Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device. More...
- struct cl::sycl::detail::opencl_type< T, constant_address_space >

  Add an attribute for __constant address space. More...
- struct cl::sycl::detail::opencl_type< T, generic_address_space >

  Add an attribute for __generic address space. More...
- struct cl::sycl::detail::opencl_type< T, global_address_space >

  Add an attribute for __global address space. More...
- struct cl::sycl::detail::opencl_type< T, local_address_space >

  Add an attribute for __local address space. More...
- struct cl::sycl::detail::opencl_type< T, private_address_space >

  Add an attribute for __private address space. More...
- struct cl::sycl::detail::address_space_array< T, AS >

  Implementation of an array variable with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_fundamental< T, AS >

  Implementation of a fundamental type with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_object< T, AS >

  Implementation of an object type with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_ptr< T, AS >

  Implementation for an OpenCL address space pointer. More...
- struct cl::sycl::detail::address_space_base< T, AS >

  Implementation of the base infrastructure to wrap something in an OpenCL address space. More...
- struct cl::sycl::detail::address_space_variable< T, AS >

  Implementation of a variable with an OpenCL address space. More...

Typedefs

- template< typename T , address_space AS>
  using cl::sycl::detail::addr_space = typename std::conditional< std::is_pointer< T >::value, address_↵
  space_ptr< T, AS >, typename std::conditional< std::is_class< T >::value, address_space_object< T, AS
  >, typename std::conditional< std::is_array< T >::value, address_space_array< T, AS >, address_space↵
  _fundamental< T, AS > >::type >::type >::type

  Dispatch the address space implementation according to the requested type.
- template< typename T >
  using cl::sycl::constant = detail::addr_space< T, constant_address_space >

  Declare a variable to be in the OpenCL constant address space.
- template< typename T >
  using cl::sycl::generic = detail::addr_space< T, generic_address_space >

  Declare a variable to be in the OpenCL 2 generic address space.
- template< typename T >
  using cl::sycl::global = detail::addr_space< T, global_address_space >

  Declare a variable to be in the OpenCL global address space.
- template< typename T >
  using cl::sycl::local = detail::addr_space< T, local_address_space >

  Declare a variable to be in the OpenCL local address space.
- template< typename T >
  using cl::sycl::priv = detail::addr_space< T, private_address_space >

  Declare a variable to be in the OpenCL private address space.
- template< typename Pointer , address_space AS>
  using cl::sycl::multi_ptr = detail::address_space_ptr< Pointer, AS >

  A pointer that can be statically associated to any address-space.

Enumerations

- enum cl::sycl::address_space {
  cl::sycl::constant_address_space, cl::sycl::generic_address_space, cl::sycl::global_address_space, cl↵
  ::sycl::local_address_space,
  cl::sycl::private_address_space }

  Enumerate the different OpenCL 2 address spaces.

Functions

- template< typename T , address_space AS>
  multi_ptr< T, AS > cl::sycl::make_multi (multi_ptr< T, AS > pointer)

  Construct a cl::sycl::multi_ptr<> with the right type.

8.2.1 Detailed Description

8.2.2 Class Documentation

8.2.2.1 struct cl::sycl::detail::opencl_type

template< typename T, address_space AS>
struct cl::sycl::detail::opencl_type< T, AS >

Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device.

In the general case, do not add any OpenCL address space qualifier

Definition at line 27 of file address_space.hpp.

Public Types

- using type = T

8.2.2.1.1 Member Typedef Documentation

8.2.2.1.1.1 template<typename T, address_space AS> using cl::sycl::detail::opencl_type< T, AS >::type = T

Definition at line 28 of file address_space.hpp.

8.2.2.2 struct cl::sycl::detail::opencl_type< T, constant_address_space >

template<typename T>
struct cl::sycl::detail::opencl_type< T, constant_address_space >

Add an attribute for __constant address space.

Definition at line 33 of file address_space.hpp.

Public Types

- using type = T

8.2.2.2.1 Member Typedef Documentation

8.2.2.2.1.1 template<typename T > using cl::sycl::detail::opencl_type< T, constant_address_space >::type = T

Definition at line 40 of file address_space.hpp.

8.2.2.3 struct cl::sycl::detail::opencl_type< T, generic_address_space >

template<typename T>
struct cl::sycl::detail::opencl_type< T, generic_address_space >

Add an attribute for __generic address space.

Definition at line 45 of file address_space.hpp.

Public Types

- using type = T

8.2.2.3.1 Member Typedef Documentation

8.2.2.3.1.1 template<typename T > using cl::sycl::detail::opencl_type< T, generic_address_space >::type = T

Definition at line 52 of file address_space.hpp.

8.2.2.4 struct cl::sycl::detail::opencl_type< T, global_address_space >

template<typename T>
struct cl::sycl::detail::opencl_type< T, global_address_space >

Add an attribute for __global address space.

Definition at line 57 of file address_space.hpp.

Public Types

- using type = T

8.2.2.4.1 Member Typedef Documentation

8.2.2.4.1.1 template<typename T > using cl::sycl::detail::opencl_type< T, global_address_space >::type = T

Definition at line 64 of file address_space.hpp.

8.2.2.5 struct cl::sycl::detail::opencl_type< T, local_address_space >

template<typename T>
struct cl::sycl::detail::opencl_type< T, local_address_space >

Add an attribute for __local address space.

Definition at line 69 of file address_space.hpp.

Public Types

- using type = T

8.2.2.5.1 Member Typedef Documentation

8.2.2.5.1.1 template<typename T > using cl::sycl::detail::opencl_type< T, local_address_space >::type = T

Definition at line 76 of file address_space.hpp.

8.2.2.6 struct cl::sycl::detail::opencl_type< T, private_address_space >

template<typename T>
struct cl::sycl::detail::opencl_type< T, private_address_space >

Add an attribute for __private address space.

Definition at line 81 of file address_space.hpp.

Public Types

- using type = T

8.2.2.6.1 Member Typedef Documentation

8.2.2.6.1.1 template< typename T > using cl::sycl::detail::opencl_type< T, private_address_space >::type = T

Definition at line 88 of file address_space.hpp.

8.2.2.7 struct cl::sycl::detail::address_space_array

template<typename T, address_space AS>
struct cl::sycl::detail::address_space_array< T, AS >

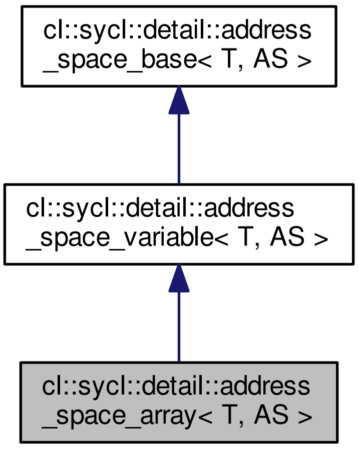
Implementation of an array variable with an OpenCL address space.

Parameters

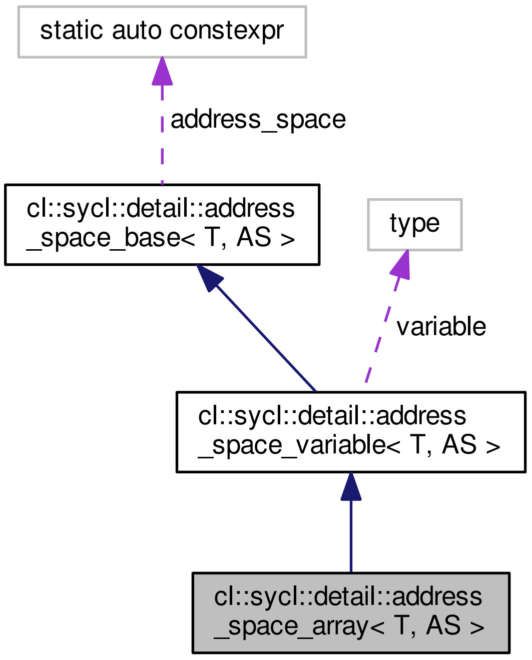
T	is the type of the basic object to be created
AS	is the address space to place the object into

Definition at line 95 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_array< T, AS >:


cl::sycl::detail::address_space_base< T, AS >
cl::sycl::detail::address_space_variable< T, AS >
cl::sycl::detail::address_space_array< T, AS >

Collaboration diagram for cl::sycl::detail::address_space_array< T, AS >:


static auto constexpr
address_space
cl::sycl::detail::address_space_base< T, AS >
type
variable
cl::sycl::detail::address_space_variable< T, AS >
cl::sycl::detail::address_space_array< T, AS >

Public Types

- using super = address_space_variable< T, AS >

  Keep track of the base class as a short-cut.

Public Member Functions

- address_space_array (const T &array)

  Allow to create an address space array from an array.
- address_space_array (std::initializer_list< std::remove_extent_t< T >> list)

  Allow to create an address space array from an initializer list.

Additional Inherited Members

8.2.2.7.1 Member Typedef Documentation

8.2.2.7.1.1 template<typename T , address_space AS> using **cl::sycl::detail::address_space_array< T, AS >::super = address_space_variable<T, AS>**

Keep track of the base class as a short-cut.

Definition at line 308 of file address_space.hpp.

8.2.2.7.2 Constructor & Destructor Documentation

8.2.2.7.2.1 template<typename T , address_space AS> cl::sycl::detail::address_space_array< T, AS >::address_space_array (const T & *array*) [inline]

Allow to create an address space array from an array.

Definition at line 316 of file address_space.hpp.

```
                                         {
    std::copy(std::begin(array), std::end(array), std::begin(super::variable));
  };
```

8.2.2.7.2.2 template<typename T , address_space AS> cl::sycl::detail::address_space_array< T, AS >::address_space_array (std::initializer_list< std::remove_extent_t< T >> *list*) [inline]

Allow to create an address space array from an initializer list.

Todo Extend to more than 1 dimension

Definition at line 325 of file address_space.hpp.

```
                                                                  {
    std::copy(std::begin(list), std::end(list), std::begin(super::variable));
  };
```

8.2.2.8 struct cl::sycl::detail::address_space_fundamental

template<typename T, address_space AS>
struct cl::sycl::detail::address_space_fundamental< T, AS >

Implementation of a fundamental type with an OpenCL address space.

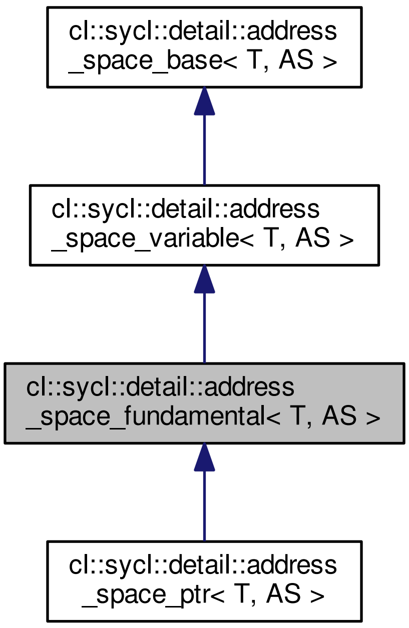
Parameters

T	is the type of the basic object to be created
AS	is the address space to place the object into

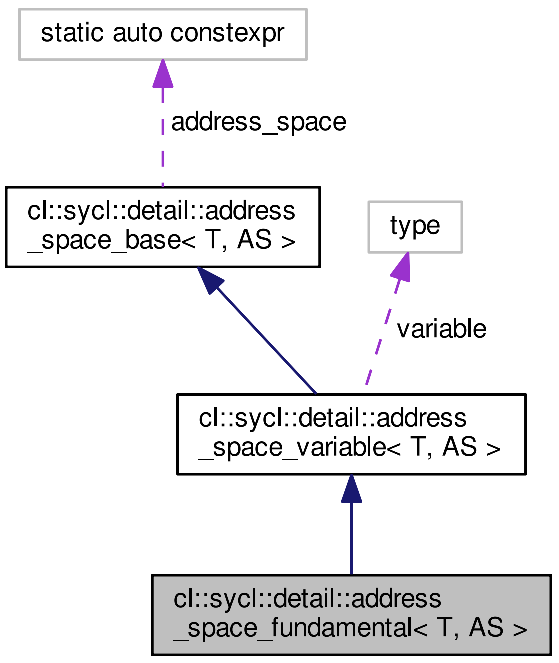
Todo Verify/improve to deal with const/volatile?

Definition at line 98 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_fundamental< T, AS >:


cl::sycl::detail::address _space_base< T, AS >
cl::sycl::detail::address _space_variable< T, AS >
cl::sycl::detail::address _space_fundamental< T, AS >
cl::sycl::detail::address _space_ptr< T, AS >

Collaboration diagram for cl::sycl::detail::address_space_fundamental< T, AS >:


static auto constexpr
address_space
cl::sycl::detail::address_space_base< T, AS >
type
variable
cl::sycl::detail::address_space_variable< T, AS >
cl::sycl::detail::address_space_fundamental< T, AS >

Public Types

- using super = address_space_variable< T, AS >

  Keep track of the base class as a short-cut.

Public Member Functions

- address_space_fundamental ()=default

  Also request for the default constructors that have been disabled by the declaration of another constructor.
- template<typename SomeType , cl::sycl::address_space SomeAS>
  address_space_fundamental (address_space_fundamental< SomeType, SomeAS > &v)

  Allow for example assignment of a global<float> to a priv<double> for example.

Additional Inherited Members

8.2.2.8.1 Member Typedef Documentation

8.2.2.8.1.1 template<typename T, address_space AS> using cl::sycl::detail::address_space_fundamental< T, AS >::super = address_space_variable<T, AS>

Keep track of the base class as a short-cut.

Definition at line 219 of file address_space.hpp.

8.2.2.8.2 Constructor & Destructor Documentation

8.2.2.8.2.1 template<typename T, address_space AS> cl::sycl::detail::address_space_fundamental< T, AS >::address_space_fundamental ( ) [default]

Also request for the default constructors that have been disabled by the declaration of another constructor.

This ensures for example that we can write

```
generic<float *> q;
```

without initialization.

8.2.2.8.2.2 template<typename T, address_space AS> template<typename SomeType , cl::sycl::address_space SomeAS> cl::sycl::detail::address_space_fundamental< T, AS >::address_space_fundamental (address_space_fundamental< SomeType, SomeAS > & *v*) [inline]

Allow for example assignment of a global<float> to a priv<double> for example.

Since it needs 2 implicit conversions, it does not work with the conversion operators already define, so add 1 more explicit conversion here so that the remaining implicit conversion can be found by the compiler.

Strangely

```
template <typename SomeType, address_space SomeAS>
address_space_base(addr_space<SomeType, SomeAS>& v)
: variable(SomeType(v)) { }
```

cannot be used here because SomeType cannot be inferred. So use address_space_base<> instead

Need to think further about it...

Definition at line 257 of file address_space.hpp.

```
{
  /* Strangely I cannot have it working in the initializer instead, for
     some cases */
  super::variable = SomeType(v);
}
```

8.2.2.9 struct cl::sycl::detail::address_space_object

template<typename T, address_space AS>
struct cl::sycl::detail::address_space_object< T, AS >

Implementation of an object type with an OpenCL address space.

Parameters

T	is the type of the basic object to be created
AS	is the address space to place the object into

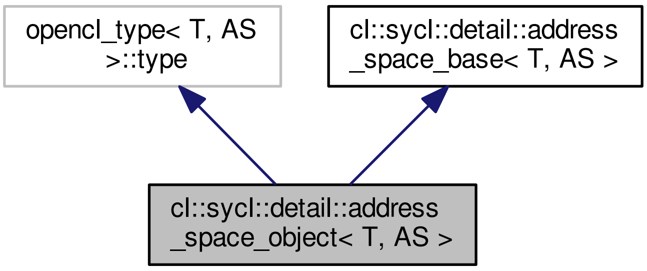
The class implementation is just inheriting of T so that all methods and non-member operators on T work also on address_space_object<T>

Todo Verify/improve to deal with const/volatile?

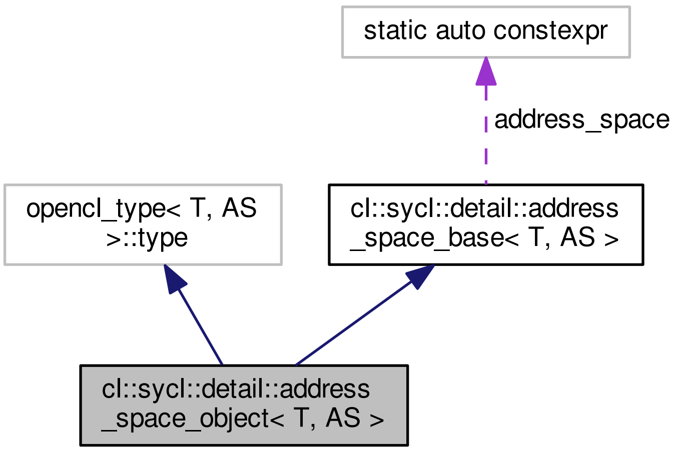
Todo what about T having some final methods?

Definition at line 101 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_object< T, AS >:


opencl_type< T, AS >::type
cl::sycl::detail::address _space_base< T, AS >
cl::sycl::detail::address _space_object< T, AS >

Collaboration diagram for cl::sycl::detail::address_space_object< T, AS >:


static auto constexpr
address_space
opencl_type< T, AS >::type
cl::sycl::detail::address _space_base< T, AS >
cl::sycl::detail::address _space_object< T, AS >

Public Types

- using opencl_type = typename opencl_type< T, AS >::type

  Store the base type of the object with OpenCL address space modifier.

Public Member Functions

- address_space_object (T &&v)

  Allow to create an address space version of an object or to convert one.
- operator opencl_type & ()

  Conversion operator to allow a address_space_object< T > to be used as a T so that all the methods of a T and the built-in operators for T can be used on a address_space_object< T > too.

Additional Inherited Members

8.2.2.9.1 Member Typedef Documentation

8.2.2.9.1.1 template<typename T , address_space AS> using **cl::sycl::detail::address_space_object< T, AS >::opencl_type** = typename **opencl_type**<T, AS>::type

Store the base type of the object with OpenCL address space modifier.

Todo Add to the specification

Definition at line 352 of file address_space.hpp.

8.2.2.9.2 Constructor & Destructor Documentation

8.2.2.9.2.1 template<typename T , address_space AS> cl::sycl::detail::address_space_object< T, AS >::address_space_object (T && *v*) [inline]

Allow to create an address space version of an object or to convert one.

Definition at line 363 of file address_space.hpp.

```
00363 : opencl_type(v) { }
```

8.2.2.9.3 Member Function Documentation

8.2.2.9.3.1 template<typename T , address_space AS> cl::sycl::detail::address_space_object< T, AS >::operator opencl_type & ( ) [inline]

Conversion operator to allow a address_space_object<T> to be used as a T so that all the methods of a T and the built-in operators for T can be used on a address_space_object<T> too.

Use opencl_type so that if we take the address of it, the address space is kept.

Definition at line 371 of file address_space.hpp.

```
00371 { return *this; }
```

8.2.2.10 struct cl::sycl::detail::address_space_ptr

template<typename T, address_space AS>
struct cl::sycl::detail::address_space_ptr< T, AS >

Implementation for an OpenCL address space pointer.

Parameters

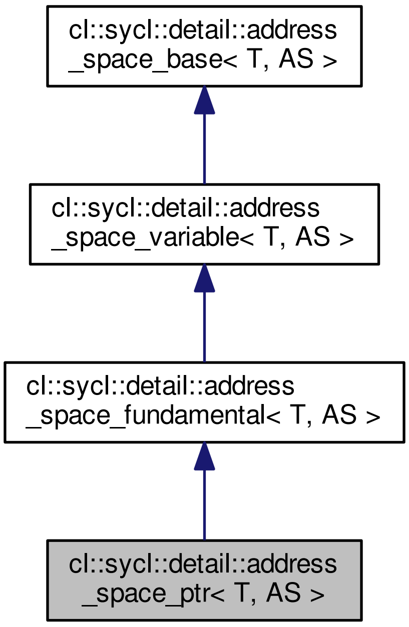
T	is the pointer type

Note that if *T* is not a pointer type, it is an error.

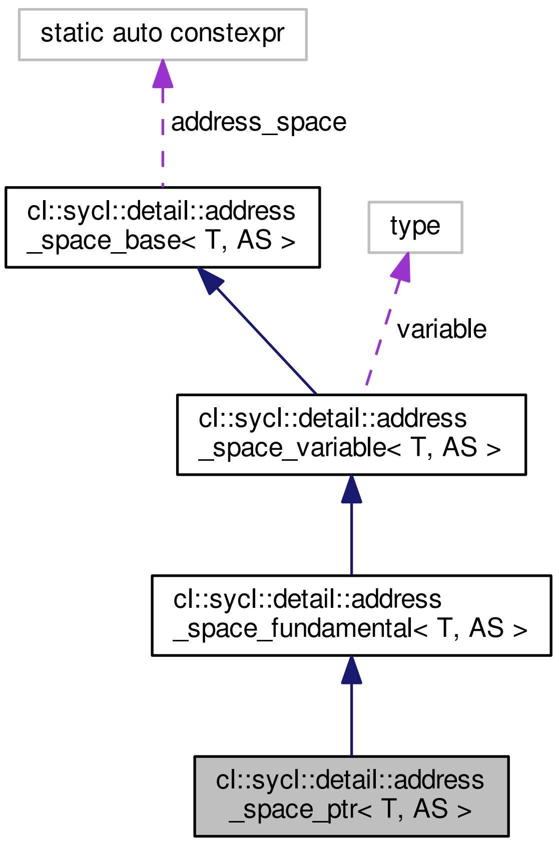
All the address space pointers inherit from it, which makes trivial the implementation of cl::sycl::multi_ptr<T, AS>

Definition at line 104 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_ptr< T, AS >:


cl::sycl::detail::address_space_base< T, AS >
cl::sycl::detail::address_space_variable< T, AS >
cl::sycl::detail::address_space_fundamental< T, AS >
cl::sycl::detail::address_space_ptr< T, AS >

Collaboration diagram for cl::sycl::detail::address_space_ptr< T, AS >:


static auto constexpr
address_space
cl::sycl::detail::address_space_base< T, AS >
type
variable
cl::sycl::detail::address_space_variable< T, AS >
cl::sycl::detail::address_space_fundamental< T, AS >
cl::sycl::detail::address_space_ptr< T, AS >

Public Types

- using super = address_space_fundamental< T, AS >

    Keep track of the base class as a short-cut.

Public Member Functions

- address_space_ptr (address_space_fundamental< typename std::pointer_traits< T >::element_type, AS > ∗p)

    Allow initialization of a pointer type from the address of an element with the same type and address space.

Additional Inherited Members

8.2.2.10.1 Member Typedef Documentation

8.2.2.10.1.1 template<typename T, address_space AS> using **cl::sycl::detail::address_space_ptr**< T, AS >::**super** = **address_space_fundamental**<T, AS>

Keep track of the base class as a short-cut.

Definition at line 283 of file address_space.hpp.

8.2.2.10.2 Constructor & Destructor Documentation

8.2.2.10.2.1 template<typename T, address_space AS> cl::sycl::detail::address_space_ptr< T, AS >::address_space_ptr (address_space_fundamental< typename std::pointer_traits< T >::element_type, AS > * *p*) [inline]

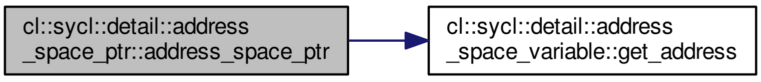
Allow initialization of a pointer type from the address of an element with the same type and address space.

Definition at line 291 of file address_space.hpp.

References cl::sycl::detail::address_space_variable< T, AS >::get_address().

```
00292     : address_space_fundamental<T, AS> { p->get_address() } {}
```

Here is the call graph for this function:


cl::sycl::detail::address
_space_ptr::address_space_ptr
cl::sycl::detail::address
_space_variable::get_address

8.2.2.11 struct cl::sycl::detail::address_space_base

template<typename T, address_space AS>
struct cl::sycl::detail::address_space_base< T, AS >

Implementation of the base infrastructure to wrap something in an OpenCL address space.

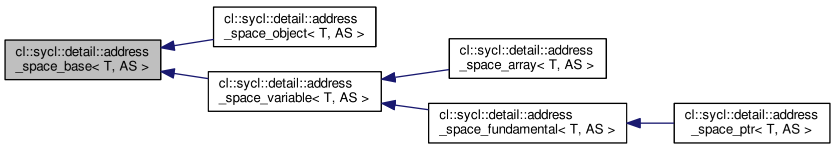
Parameters

T	is the type of the basic stuff to be created
AS	is the address space to place the object into

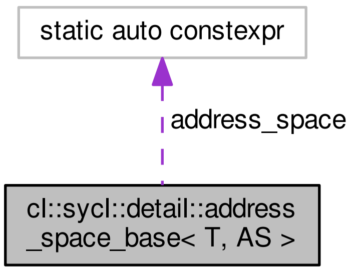
Todo Verify/improve to deal with const/volatile?

Definition at line 135 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_base< T, AS >:


cl::sycl::detail::address_space_base< T, AS >
cl::sycl::detail::address_space_object< T, AS >
cl::sycl::detail::address_space_variable< T, AS >
cl::sycl::detail::address_space_array< T, AS >
cl::sycl::detail::address_space_fundamental< T, AS >
cl::sycl::detail::address_space_ptr< T, AS >

Collaboration diagram for cl::sycl::detail::address_space_base< T, AS >:



Public Types

- using type = T
  Store the base type of the object.
- using opencl_type = typename opencl_type< T, AS >::type
  Store the base type of the object with OpenCL address space modifier.

Static Public Attributes

- static auto constexpr address_space = AS
  Set the address_space identifier that can be queried to know the pointer type.

8.2.2.11.1 Member Typedef Documentation

8.2.2.11.1.1 template<typename T , address_space AS> using cl::sycl::detail::address_space_base< T, AS >::opencl_type = typename opencl_type<T, AS>::type

Store the base type of the object with OpenCL address space modifier.

Todo Add to the specification

Definition at line 146 of file address_space.hpp.

8.2.2.11.1.2 template< typename T , address_space AS > using cl::sycl::detail::address_space_base< T, AS >::type = T

Store the base type of the object.

Todo Add to the specification

Definition at line 140 of file address_space.hpp.

8.2.2.11.2 Member Data Documentation

8.2.2.11.2.1 template< typename T , address_space AS > auto constexpr cl::sycl::detail::address_space_base< T, AS >::address_space = AS [static]

Set the address_space identifier that can be queried to know the pointer type.

Definition at line 150 of file address_space.hpp.

8.2.2.12 struct cl::sycl::detail::address_space_variable

template< typename T, address_space AS >
struct cl::sycl::detail::address_space_variable< T, AS >

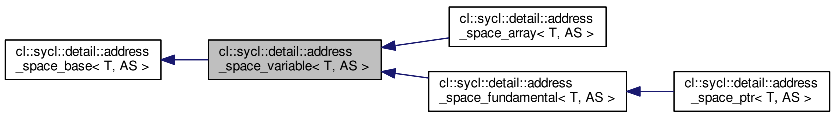
Implementation of a variable with an OpenCL address space.

Parameters

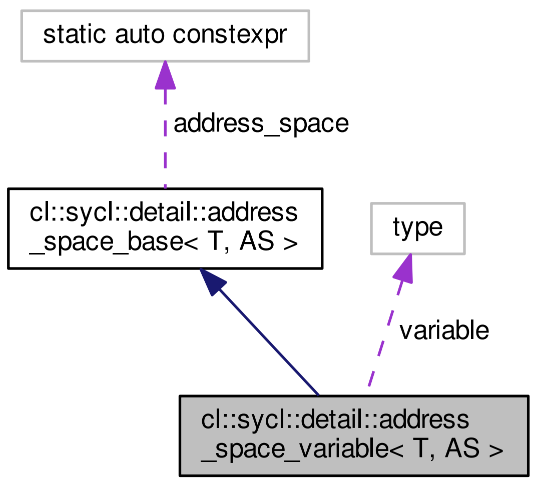
T	is the type of the basic object to be created
AS	is the address space to place the object into

Definition at line 162 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_variable< T, AS >:


cl::sycl::detail::address_space_base< T, AS >
cl::sycl::detail::address_space_variable< T, AS >
cl::sycl::detail::address_space_array< T, AS >
cl::sycl::detail::address_space_fundamental< T, AS >
cl::sycl::detail::address_space_ptr< T, AS >

Collaboration diagram for cl::sycl::detail::address_space_variable< T, AS >:


static auto constexpr
address_space
cl::sycl::detail::address_space_base< T, AS >
type
variable
cl::sycl::detail::address_space_variable< T, AS >

Public Types

- using opencl_type = typename opencl_type< T, AS >::type

  Store the base type of the object with OpenCL address space modifier.
- using super = address_space_base< T, AS >

  Keep track of the base class as a short-cut.

Public Member Functions

- address_space_variable (const T &v)

  Allow to create an address space version of an object or to convert one to be used by the classes inheriting by this one because it is not possible to directly initialize a base class member in C++.
- address_space_variable ()=default

  Put back the default constructors canceled by the previous definition.
- operator opencl_type & ()

  Conversion operator to allow a address_space_object< T > to be used as a T so that all the methods of a T and the built-in operators for T can be used on a address_space_object< T > too.
- opencl_type ∗ get_address ()

  Return the address of the value to implement pointers.

Protected Attributes

- opencl_type variable

Additional Inherited Members

8.2.2.12.1 Member Typedef Documentation

8.2.2.12.1.1 template<typename T , address_space AS> using **cl::sycl::detail::address_space_variable< T, AS >::opencl_type** = typename **opencl_type<T, AS>::type**

Store the base type of the object with OpenCL address space modifier.

Todo Add to the specification

Definition at line 167 of file address_space.hpp.

8.2.2.12.1.2 template<typename T , address_space AS> using **cl::sycl::detail::address_space_variable< T, AS >::super** = **address_space_base<T, AS>**

Keep track of the base class as a short-cut.

Definition at line 170 of file address_space.hpp.

8.2.2.12.2 Constructor & Destructor Documentation

8.2.2.12.2.1 template<typename T , address_space AS> **cl::sycl::detail::address_space_variable< T, AS >::address_space_variable** (const T & *v*) `[inline]`

Allow to create an address space version of an object or to convert one to be used by the classes inheriting by this one because it is not possible to directly initialize a base class member in C++.

Definition at line 186 of file address_space.hpp.

```
00186 : variable(v) { }
```

8.2.2.12.2.2 template<typename T , address_space AS> **cl::sycl::detail::address_space_variable< T, AS >::address_space_variable** ( ) `[default]`

Put back the default constructors canceled by the previous definition.

8.2.2.12.3 Member Function Documentation

8.2.2.12.3.1 template<typename T , address_space AS> **opencl_type∗ cl::sycl::detail::address_space_variable< T, AS >::get_address** ( ) `[inline]`

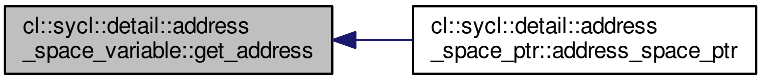
Return the address of the value to implement pointers.

Definition at line 203 of file address_space.hpp.

Referenced by cl::sycl::detail::address_space_ptr< T, AS >::address_space_ptr().

```
00203 { return &variable; }
```

Here is the caller graph for this function:


cl::sycl::detail::address_space_variable::get_address
cl::sycl::detail::address_space_ptr::address_space_ptr

8.2.2.12.3.2 template<typename T , address_space AS> cl::sycl::detail::address_space_variable< T, AS >::operator opencl_type & ( ) [inline]

Conversion operator to allow a address_space_object<T> to be used as a T so that all the methods of a T and the built-in operators for T can be used on a address_space_object<T> too.

Use opencl_type so that if we take the address of it, the address space is kept.

Definition at line 200 of file address_space.hpp.

```
00200 { return variable; }
```

8.2.2.12.4 Member Data Documentation

8.2.2.12.4.1 template<typename T , address_space AS> opencl_type cl::sycl::detail::address_space_variable< T, AS >::variable [protected]

Definition at line 179 of file address_space.hpp.

8.2.3 Typedef Documentation

8.2.3.1 template<typename T , address_space AS> using cl::sycl::detail::addr_space = typedef typename std::conditional<std::is_pointer<T>::value, address_space_ptr<T, AS>, typename std::conditional<std::is_class<T>::value, address_space_object<T, AS>, typename std::conditional<std::is_array<T>::value, address_space_array<T, AS>, address_space_fundamental<T, AS> >::type>::type>::type

```
#include <include/CL/sycl/address_space/detail/address_space.hpp>
```

Dispatch the address space implementation according to the requested type.

Parameters

T	is the type of the object to be created
AS	is the address space to place the object into or to point to in the case of a pointer type

Definition at line 122 of file address_space.hpp.

8.2.3.2 template<typename T > using cl::sycl::constant = typedef detail::addr_space<T, constant_address_space>

```
#include <include/CL/sycl/address_space.hpp>
```

Declare a variable to be in the OpenCL constant address space.

Parameters

T	is the type of the object

Definition at line 55 of file address_space.hpp.

8.2.3.3 template< typename T > using cl::sycl::generic = typedef detail::addr_space<T, generic_address_space>

`#include <include/CL/sycl/address_space.hpp>`

Declare a variable to be in the OpenCL 2 generic address space.

Parameters

T	is the type of the object

Definition at line 63 of file address_space.hpp.

8.2.3.4 template< typename T > using cl::sycl::global = typedef detail::addr_space<T, global_address_space>

`#include <include/CL/sycl/address_space.hpp>`

Declare a variable to be in the OpenCL global address space.

Parameters

T	is the type of the object

Definition at line 71 of file address_space.hpp.

8.2.3.5 template< typename T > using cl::sycl::local = typedef detail::addr_space<T, local_address_space>

`#include <include/CL/sycl/address_space.hpp>`

Declare a variable to be in the OpenCL local address space.

Parameters

T	is the type of the object

Definition at line 79 of file address_space.hpp.

8.2.3.6 template< typename Pointer , address_space AS> using cl::sycl::multi_ptr = typedef detail::address_space_ptr<Pointer, AS>

`#include <include/CL/sycl/address_space.hpp>`

A pointer that can be statically associated to any address-space.

Parameters

Pointer	is the pointer type
AS	is the address space to point to

Note that if *Pointer* is not a pointer type, it is an error.

Definition at line 99 of file address_space.hpp.

8.2.3.7 template<typename T > using cl::sycl::priv = typedef detail::addr_space<T, private_address_space>

```
#include <include/CL/sycl/address_space.hpp>
```

Declare a variable to be in the OpenCL private address space.

Parameters

T	is the type of the object

Definition at line 87 of file address_space.hpp.

8.2.4 Enumeration Type Documentation

8.2.4.1 enum cl::sycl::address_space

```
#include <include/CL/sycl/address_space.hpp>
```

Enumerate the different OpenCL 2 address spaces.

Enumerator

constant_address_space

generic_address_space

global_address_space

local_address_space

private_address_space

Definition at line 27 of file address_space.hpp.

```
00027                    {
00028   constant_address_space,
00029   generic_address_space,
00030   global_address_space,
00031   local_address_space,
00032   private_address_space,
00033 };
```

8.2.5 Function Documentation

8.2.5.1 template<typename T , address_space AS> multi_ptr<T, AS> cl::sycl::make_multi (multi_ptr< T, AS > *pointer*)

```
#include <include/CL/sycl/address_space.hpp>
```

Construct a cl::sycl::multi_ptr<> with the right type.

Parameters

pointer	is the address with its address space to point to

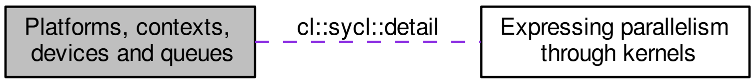
Todo Implement the case with a plain pointer

Definition at line 109 of file address_space.hpp.

```
                                                      {
  return pointer;
}
```

8.3 Platforms, contexts, devices and queues

Collaboration diagram for Platforms, contexts, devices and queues:


Platforms, contexts, devices and queues
cl::sycl::detail
Expressing parallelism through kernels

Namespaces

- cl::sycl::info
- cl::sycl::detail

Classes

- class cl::sycl::context

  SYCL context. More...
- class cl::sycl::detail::device

  An abstract class representing various models of SYCL devices. More...
- class cl::sycl::device

  SYCL device. More...
- class cl::sycl::device_type_selector

  A device selector by device_type. More...
- class cl::sycl::device_typename_selector< DeviceType >

  Select a device by template device_type parameter. More...
- class cl::sycl::device_selector

  The SYCL heuristics to select a device. More...
- class cl::sycl::handler

  Command group handler class. More...
- class cl::sycl::detail::kernel

  Abstract SYCL kernel. More...
- class cl::sycl::kernel

  SYCL kernel. More...
- class cl::sycl::detail::host_platform

  SYCL host platform. More...
- class cl::sycl::detail::opencl_platform

  SYCL OpenCL platform. More...
- class cl::sycl::detail::platform

  An abstract class representing various models of SYCL platforms. More...
- class cl::sycl::platform

  Abstract the OpenCL platform. More...
- class cl::sycl::queue

  SYCL queue, similar to the OpenCL queue concept. More...

Typedefs

- using cl::sycl::default_selector = device_typename_selector< info::device_type::defaults >

  Devices selected by heuristics of the system.
- using cl::sycl::gpu_selector = device_typename_selector< info::device_type::gpu >

  Select devices according to device type info::device::device_type::gpu from all the available OpenCL devices.
- using cl::sycl::cpu_selector = device_typename_selector< info::device_type::cpu >

  Select devices according to device type info::device::device_type::cpu from all the available devices and heuristics.
- using cl::sycl::host_selector = device_typename_selector< info::device_type::host >

  Selects the SYCL host CPU device that does not require an OpenCL runtime.
- using cl::sycl::info::device_fp_config = unsigned int
- using cl::sycl::info::device_exec_capabilities = unsigned int
- using cl::sycl::info::device_queue_properties = unsigned int

Enumerations

- enum cl::sycl::info::device_type : unsigned int {
  cl::sycl::info::device_type::cpu, cl::sycl::info::device_type::gpu, cl::sycl::info::device_type::accelerator, cl↩
  ::sycl::info::device_type::custom,
  cl::sycl::info::device_type::defaults, cl::sycl::info::device_type::host, cl::sycl::info::device_type::opencl, cl↩
  ::sycl::info::device_type::all }

  Type of devices.
- enum cl::sycl::info::device : int {
  cl::sycl::info::device::device_type, cl::sycl::info::device::vendor_id, cl::sycl::info::device::max_compute_units,
  cl::sycl::info::device::max_work_item_dimensions,
  cl::sycl::info::device::max_work_item_sizes, cl::sycl::info::device::max_work_group_size, cl::sycl::info↩
  ::device::preferred_vector_width_char, cl::sycl::info::device::preferred_vector_width_short,
  cl::sycl::info::device::preferred_vector_width_int, cl::sycl::info::device::preferred_vector_width_long_long, cl↩
  ::sycl::info::device::preferred_vector_width_float, cl::sycl::info::device::preferred_vector_width_double,
  cl::sycl::info::device::preferred_vector_width_half, cl::sycl::info::device::native_vector_witdth_char, cl::sycl↩
  ::info::device::native_vector_witdth_short, cl::sycl::info::device::native_vector_witdth_int,
  cl::sycl::info::device::native_vector_witdth_long_long, cl::sycl::info::device::native_vector_witdth_float, cl↩
  ::sycl::info::device::native_vector_witdth_double, cl::sycl::info::device::native_vector_witdth_half,
  cl::sycl::info::device::max_clock_frequency, cl::sycl::info::device::address_bits, cl::sycl::info::device::max_↩
  mem_alloc_size, cl::sycl::info::device::image_support,
  cl::sycl::info::device::max_read_image_args, cl::sycl::info::device::max_write_image_args, cl::sycl::info↩
  ::device::image2d_max_height, cl::sycl::info::device::image2d_max_width,
  cl::sycl::info::device::image3d_max_height, cl::sycl::info::device::image3d_max_widht, cl::sycl::info::device↩
  ::image3d_mas_depth, cl::sycl::info::device::image_max_buffer_size,
  cl::sycl::info::device::image_max_array_size, cl::sycl::info::device::max_samplers, cl::sycl::info::device↩
  ::max_parameter_size, cl::sycl::info::device::mem_base_addr_align,
  cl::sycl::info::device::single_fp_config, cl::sycl::info::device::double_fp_config, cl::sycl::info::device::global_↩
  mem_cache_type, cl::sycl::info::device::global_mem_cache_line_size,
  cl::sycl::info::device::global_mem_cache_size, cl::sycl::info::device::global_mem_size, cl::sycl::info::device↩
  ::max_constant_buffer_size, cl::sycl::info::device::max_constant_args,
  cl::sycl::info::device::local_mem_type, cl::sycl::info::device::local_mem_size, cl::sycl::info::device::error_↩
  correction_support, cl::sycl::info::device::host_unified_memory,
  cl::sycl::info::device::profiling_timer_resolution, cl::sycl::info::device::endian_little, cl::sycl::info::device::is_↩
  available, cl::sycl::info::device::is_compiler_available,
  cl::sycl::info::device::is_linker_available, cl::sycl::info::device::execution_capabilities, cl::sycl::info::device↩
  ::queue_properties, cl::sycl::info::device::built_in_kernels,
  cl::sycl::info::device::platform, cl::sycl::info::device::name, cl::sycl::info::device::vendor, cl::sycl::info::device↩
  ::driver_version,
  cl::sycl::info::device::profile, cl::sycl::info::device::device_version, cl::sycl::info::device::opencl_version, cl↩
  ::sycl::info::device::extensions,
  cl::sycl::info::device::printf_buffer_size, cl::sycl::info::device::preferred_interop_user_sync, cl::sycl::info↩

::device::parent_device, cl::sycl::info::device::partition_max_sub_devices, cl::sycl::info::device::partition_properties, cl::sycl::info::device::partition_affinity_domain, cl::sycl::info↩ ::device::partition_type, cl::sycl::info::device::reference_count }

Device information descriptors.

- enum cl::sycl::info::device_partition_property : int { cl::sycl::info::device_partition_property::unsupported, cl::sycl::info::device_partition_property::partition↩ _equally, cl::sycl::info::device_partition_property::partition_by_counts, cl::sycl::info::device_partition_↩ property::partition_by_affinity_domain, cl::sycl::info::device_partition_property::partition_affinity_domain_next_partitionable }
- enum cl::sycl::info::device_affinity_domain : int { cl::sycl::info::device_affinity_domain::unsupported, cl::sycl::info::device_affinity_domain::numa, cl::sycl↩ ::info::device_affinity_domain::L4_cache, cl::sycl::info::device_affinity_domain::L3_cache, cl::sycl::info::device_affinity_domain::L2_cache, cl::sycl::info::device_affinity_domain::next_partitionable }
- enum cl::sycl::info::device_partition_type : int { cl::sycl::info::device_partition_type::no_partition, cl::sycl::info::device_partition_type::numa, cl::sycl::info↩ ::device_partition_type::L4_cache, cl::sycl::info::device_partition_type::L3_cache, cl::sycl::info::device_partition_type::L2_cache, cl::sycl::info::device_partition_type::L1_cache }
- enum cl::sycl::info::local_mem_type : int { cl::sycl::info::local_mem_type::none, cl::sycl::info::local_mem_↩ type::local, cl::sycl::info::local_mem_type::global }
- enum cl::sycl::info::fp_config : int { cl::sycl::info::fp_config::denorm, cl::sycl::info::fp_config::inf_nan, cl::sycl::info::fp_config::round_to_nearest, cl::sycl::info::fp_config::round_to_zero, cl::sycl::info::fp_config::round_to_inf, cl::sycl::info::fp_config::fma, cl::sycl::info::fp_config::correctly_↩ rounded_divide_sqrt, cl::sycl::info::fp_config::soft_float }
- enum cl::sycl::info::global_mem_cache_type : int { cl::sycl::info::global_mem_cache_type::none, cl::sycl↩ ::info::global_mem_cache_type::read_only, cl::sycl::info::global_mem_cache_type::write_only }
- enum cl::sycl::info::device_execution_capabilities : unsigned int { cl::sycl::info::device_execution_↩ capabilities::exec_kernel, cl::sycl::info::device_execution_capabilities::exec_native_kernel }
- enum cl::sycl::info::platform : unsigned int { cl::sycl::info::platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_PROFILE), cl::sycl::info::platform::↩ TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_VERSION), cl::sycl::info::platform::TRISYCL_SKIP_OPE↩ NCL =(= CL_PLATFORM_NAME), cl::sycl::info::platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM↩ _VENDOR), cl::sycl::info::platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_EXTENSIONS) }

  Platform information descriptors.

Functions

- detail::cache< cl_kernel, detail::opencl_kernel > opencl_kernel::cache cl::sycl::detail::__attribute__ ((weak))
- static vector_class< device > cl::sycl::device::get_devices (info::device_type device_type=info::device_↩ type::all) __attribute__((weak))

  Return a list of all available devices.

Variables

- detail::cache< cl_device_id, detail::opencl_device > opencl_device::cache cl::sycl::detail::__attribute__↩ ((weak))

8.3.1 Detailed Description

8.3.2 Class Documentation

8.3.2.1 class cl::sycl::context

SYCL context.

The context class encapsulates an OpenCL context, which is implicitly created and the lifetime of the context instance defines the lifetime of the underlying OpenCL context instance.

On destruction clReleaseContext is called.

The default context is the SYCL host context containing only the SYCL host device.

Todo The implementation is quite minimal for now.

Definition at line 66 of file context.hpp.

Public Member Functions

- context (async_handler asyncHandler)

  Constructs a context object for SYCL host using an async_handler for handling asynchronous errors.
- context (cl_context clContext, async_handler asyncHandler=nullptr)
- context (const device_selector &deviceSelector, info::gl_context_interop interopFlag, async_handler async↩ Handler=nullptr)

  Constructs a context object using a device_selector object.
- context (const device &dev, info::gl_context_interop interopFlag, async_handler asyncHandler=nullptr)

  Constructs a context object using a device object.
- context (const platform &plt, info::gl_context_interop interopFlag, async_handler asyncHandler=nullptr)

  Constructs a context object using a platform object.
- context (const vector_class< device > &deviceList, info::gl_context_interop interopFlag, async_handler asyncHandler=nullptr)
- context ()=default

  Default constructor that chooses the context according the heuristics of the default selector.
- cl_context get () const
- bool is_host () const

  Specifies whether the context is in SYCL Host Execution Mode.
- platform get_platform ()

  Returns the SYCL platform that the context is initialized for.
- vector_class< device > get_devices () const

  Returns the set of devices that are part of this context.
- template<info::context Param>
  info::param_traits< info::context, Param >::type get_info () const

  Queries OpenCL information for the under-lying cl context.

8.3.2.1.1 Constructor & Destructor Documentation

8.3.2.1.1.1 cl::sycl::context::context (**async_handler** *asyncHandler*) [inline], [explicit]

Constructs a context object for SYCL host using an async_handler for handling asynchronous errors.

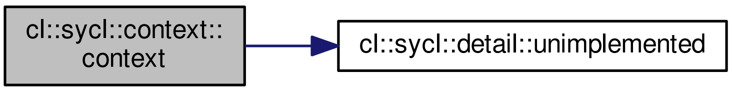
Note that the default case asyncHandler = nullptr is handled by the default constructor.

Definition at line 76 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
                                                {
    detail::unimplemented();
  }
```

Here is the call graph for this function:



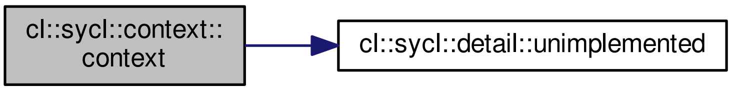
8.3.2.1.1.2 cl::sycl::context::context (cl_context *clContext,* **async_handler** *asyncHandler* = nullptr) [inline]

Definition at line 90 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
                                                                       {
    detail::unimplemented();
  }
```

Here is the call graph for this function:



8.3.2.1.1.3 cl::sycl::context::context (const device_selector & *deviceSelector*, info::gl_context_interop *interopFlag*, async_handler *asyncHandler* = nullptr) [inline]

Constructs a context object using a device_selector object.

The context is constructed with a single device retrieved from the device_selector object provided.

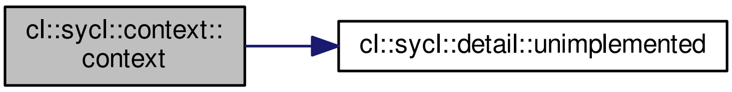
Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the async_↩
handler, if provided.

Definition at line 103 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
                                                {
    detail::unimplemented();
  }
```

Here is the call graph for this function:



8.3.2.1.1.4 cl::sycl::context::context (const device & *dev*, info::gl_context_interop *interopFlag*, async_handler *asyncHandler* = nullptr) [inline]

Constructs a context object using a device object.

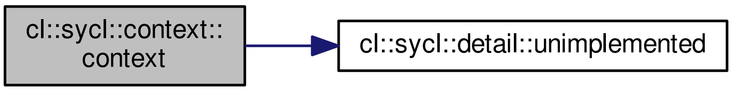
Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the async_↩
handler, if provided.

Definition at line 115 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
                                                {
    detail::unimplemented();
  }
```

Here is the call graph for this function:



8.3.2.1.1.5 cl::sycl::context::context (const platform & *plt*, info::gl_context_interop *interopFlag*, async_handler *asyncHandler* = nullptr) [inline]

Constructs a context object using a platform object.

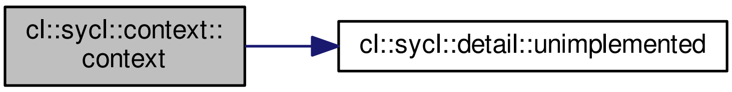
Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the async_↩ handler, if provided.

Definition at line 127 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
                                                 {
        detail::unimplemented();
    }
```

Here is the call graph for this function:



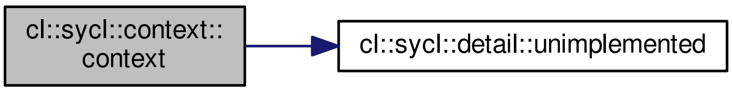
8.3.2.1.1.6 cl::sycl::context::context (const vector_class< device > & *deviceList*, info::gl_context_interop *interopFlag*, async_handler *asyncHandler* = nullptr) [inline]

Definition at line 142 of file context.hpp.

References cl::sycl::info::context, and cl::sycl::detail::unimplemented().

```
                                                 {
        detail::unimplemented();
    }
```

Here is the call graph for this function:



8.3.2.1.1.7 cl::sycl::context::context ( ) [default]

Default constructor that chooses the context according the heuristics of the default selector.

Return synchronous errors via the SYCL exception class.

Get the default constructors back.

8.3.2.1.2 Member Function Documentation

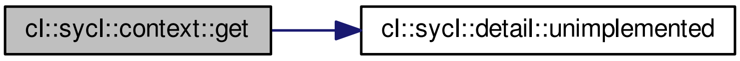
8.3.2.1.2.1 cl_context cl::sycl::context::get ( ) const [inline]

Definition at line 165 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
                           {
    detail::unimplemented();
    return {};
  }
```

Here is the call graph for this function:



8.3.2.1.2.2 vector_class<device> cl::sycl::context::get_devices ( ) const [inline]

Returns the set of devices that are part of this context.

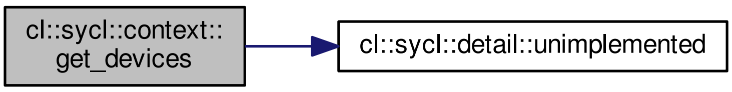
Todo To be implemented

Definition at line 189 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
                                               {
    detail::unimplemented();
    return {};
  }
```

Here is the call graph for this function:



8.3.2.1.2.3 template<info::context Param> **info::param_traits<info::context**, Param>::type cl::sycl::context::get_info ( ) const [inline]

Queries OpenCL information for the under-lying cl context.

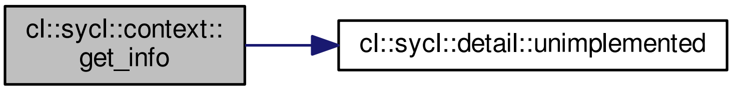
Todo To be implemented

Definition at line 200 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
                                                                   {
    detail::unimplemented();
    return {};
  }
```

Here is the call graph for this function:



8.3.2.1.2.4 **platform** cl::sycl::context::get_platform ( )

Returns the SYCL platform that the context is initialized for.

Todo To be implemented

8.3.2.1.2.5 **bool** cl::sycl::context::is_host ( ) const [inline]

Specifies whether the context is in SYCL Host Execution Mode.

Definition at line 173 of file context.hpp.

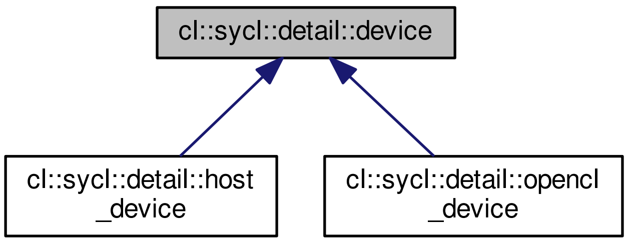
```
                              {
    return true;
  }
```

8.3.2.2 class cl::sycl::detail::device

An abstract class representing various models of SYCL devices.

Definition at line 25 of file device.hpp.

Inheritance diagram for cl::sycl::detail::device:


cl::sycl::detail::device
cl::sycl::detail::host _device
cl::sycl::detail::opencl _device

Public Member Functions

- virtual cl_device_id get () const =0

    Return the cl_device_id of the underlying OpenCL platform.
- virtual bool is_host () const =0

    Return true if the device is a SYCL host device.
- virtual bool is_cpu () const =0

    Return true if the device is an OpenCL CPU device.
- virtual bool is_gpu () const =0

    Return true if the device is an OpenCL GPU device.
- virtual bool is_accelerator () const =0

    Return true if the device is an OpenCL accelerator device.
- virtual cl::sycl::platform get_platform () const =0

    Return the platform of device.
- virtual bool has_extension (const string_class &extension) const =0

    Query the device for OpenCL info::device info.
- virtual ∼device ()

8.3.2.2.1 Constructor & Destructor Documentation

8.3.2.2.1.1 virtual cl::sycl::detail::device::∼device ( ) [inline], [virtual]

Definition at line 67 of file device.hpp.

```
00067 {}
```

8.3.2.2.2 Member Function Documentation

8.3.2.2.2.1 virtual cl_device_id cl::sycl::detail::device::get ( ) const [pure virtual]

Return the cl_device_id of the underlying OpenCL platform.

Implemented in cl::sycl::detail::opencl_device, and cl::sycl::detail::host_device.

8.3.2.2.2.2 virtual **cl::sycl::platform** cl::sycl::detail::device::get_platform ( ) const [pure virtual]

Return the platform of device.

Implemented in cl::sycl::detail::opencl_device, and cl::sycl::detail::host_device.

8.3.2.2.2.3 virtual **bool** cl::sycl::detail::device::has_extension (const **string_class** & *extension*) const [pure virtual]

Query the device for OpenCL info::device info.

Todo virtual cannot be templated template <typename t>=""> virtual T get_info(info::device param) const = 0;

Specify whether a specific extension is supported on the device.

Implemented in cl::sycl::detail::opencl_device, and cl::sycl::detail::host_device.

8.3.2.2.2.4 virtual **bool** cl::sycl::detail::device::is_accelerator ( ) const [pure virtual]

Return true if the device is an OpenCL accelerator device.

Implemented in cl::sycl::detail::opencl_device, and cl::sycl::detail::host_device.

8.3.2.2.2.5 virtual **bool** cl::sycl::detail::device::is_cpu ( ) const [pure virtual]

Return true if the device is an OpenCL CPU device.

Implemented in cl::sycl::detail::opencl_device, and cl::sycl::detail::host_device.

8.3.2.2.2.6 virtual **bool** cl::sycl::detail::device::is_gpu ( ) const [pure virtual]

Return true if the device is an OpenCL GPU device.

Implemented in cl::sycl::detail::opencl_device, and cl::sycl::detail::host_device.

8.3.2.2.2.7 virtual **bool** cl::sycl::detail::device::is_host ( ) const [pure virtual]

Return true if the device is a SYCL host device.

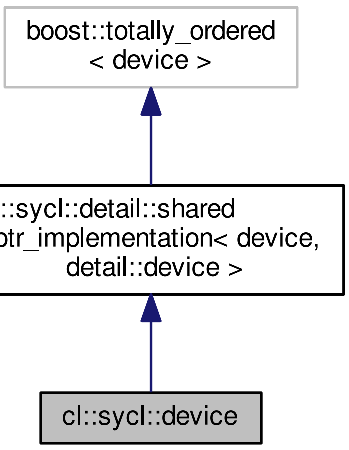
Implemented in cl::sycl::detail::opencl_device, and cl::sycl::detail::host_device.

8.3.2.3 class cl::sycl::device

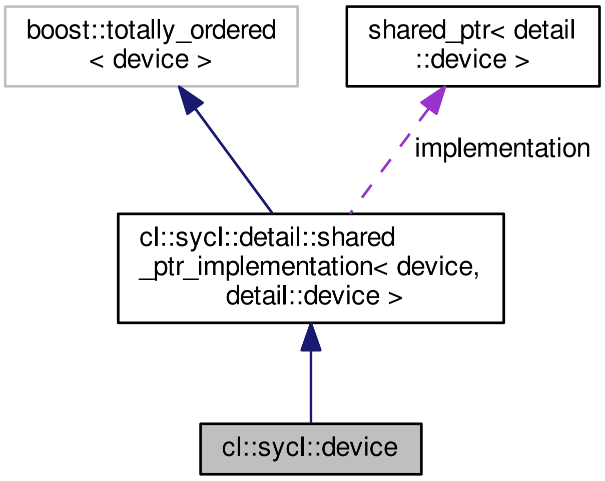
SYCL device.

Definition at line 41 of file device.hpp.

Inheritance diagram for cl::sycl::device:



Collaboration diagram for cl::sycl::device:



Public Member Functions

- device ()

    The default constructor uses the SYCL host device.
- device (cl_device_id device_id)

    Construct a device class instance using cl_device_id of the OpenCL device.
- device (const boost::compute::device &d)

    Construct a device class instance using a boost::compute::device.
- device (const device_selector &ds)

    Construct a device class instance using the device selector provided.
- cl_device_id get () const

    Return the cl_device_id of the underlying OpenCL platform.
- bool is_host () const

    Return true if the device is the SYCL host device.
- bool is_cpu () const

    Return true if the device is an OpenCL CPU device.
- bool is_gpu () const

    Return true if the device is an OpenCL GPU device.
- bool is_accelerator () const

    Return true if the device is an OpenCL accelerator device.
- info::device_type type () const

    Return the device_type of a device.
- platform get_platform () const

    Return the platform of device.
- template< typename T >
  T get_info (info::device param) const

    Query the device for OpenCL info::device info.
- template< info::device Param>
  auto get_info () const

    Query the device for OpenCL info::device info.
- bool has_extension (const string_class &extension) const

    Test if a specific extension is supported on the device.

Static Public Member Functions

- static vector_class< device > get_devices (info::device_type device_type=info::device_type::all) __↩
  attribute__((weak))

    Return a list of all available devices.

Private Types

- using implementation_t = detail::shared_ptr_implementation< device, detail::device >

Additional Inherited Members

8.3.2.3.1 Member Typedef Documentation

8.3.2.3.1.1 using cl::sycl::device::implementation_t = detail::shared_ptr_implementation<device, detail::device> [private]

Definition at line 48 of file device.hpp.

8.3.2.3.2 Constructor & Destructor Documentation

8.3.2.3.2.1 cl::sycl::device::device ( ) [inline]

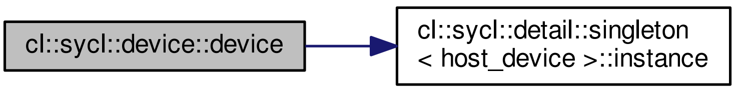
The default constructor uses the SYCL host device.

Definition at line 56 of file device.hpp.

References cl::sycl::detail::singleton< host_device >::instance().

```
00056 : implementation_t { detail::host_device::instance() } {}
```

Here is the call graph for this function:



8.3.2.3.2.2 cl::sycl::device::device (cl_device_id *device_id*) [inline]

Construct a device class instance using cl_device_id of the OpenCL device.

Return synchronous errors via the SYCL exception class.

Retain a reference to the OpenCL device and if this device was an OpenCL subdevice the device should be released by the caller when it is no longer needed.

Definition at line 69 of file device.hpp.

```
00070     : device { boost::compute::device { device_id } } {}
```

8.3.2.3.2.3 cl::sycl::device::device (const boost::compute::device & *d*) [inline]

Construct a device class instance using a boost::compute::device.

This is a triSYCL extension for boost::compute interoperation.

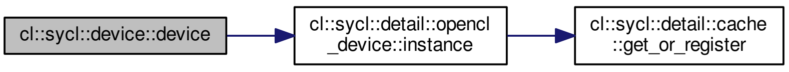
Return synchronous errors via the SYCL exception class.

Definition at line 79 of file device.hpp.

References cl::sycl::detail::opencl_device::instance().

```
00080     : implementation_t { detail::opencl_device::instance(d)
      } {}
```

Here is the call graph for this function:


cl::sycl::device::device
cl::sycl::detail::opencl _device::instance
cl::sycl::detail::cache ::get_or_register

8.3.2.3.2.4 cl::sycl::device::device (const device_selector & *ds*) [inline], [explicit]

Construct a device class instance using the device selector provided.

Return errors via C++ exception class.

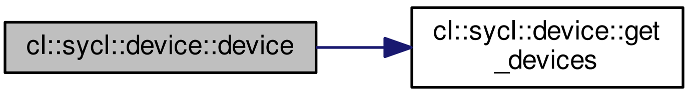
Todo Make it non-explicit in the specification?

Definition at line 91 of file device.hpp.

References get_devices(), and cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

```
                                              {
    auto devices = device::get_devices();
    if (devices.empty())
      // \todo Put a SYCL exception
      throw std::domain_error("No device at all! Internal error...");

    /* Find the device with the best score according to the given
       device_selector */
    auto max = std::max_element(devices.cbegin(), devices.cend(),
                                [&] (const device &d1, const device &d2) {
                                  return ds(d1) < ds(d2);
                                });
    if (ds(*max) < 0)
      // \todo Put a SYCL exception
      throw std::domain_error("No device selected because no positive "
                              "device_selector score found");

    // Create the current device as a shared copy of the selected one
    implementation = max->implementation;
  }
```

Here is the call graph for this function:



8.3.2.3.3 Member Function Documentation

8.3.2.3.3.1 cl_device_id cl::sycl::device::get ( ) const [inline]

Return the cl_device_id of the underlying OpenCL platform.

Return synchronous errors via the SYCL exception class.

Retain a reference to the returned cl_device_id object. Caller should release it when finished.

In the case where this is the SYCL host device it will throw an exception.

Definition at line 124 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

```
                           {
    return implementation->get();
  }
```

8.3.2.3.3.2 template<typename T > T cl::sycl::device::get_info (info::device *param*) const [inline]

Query the device for OpenCL info::device info.

Return synchronous errors via the SYCL exception class.

Todo

Definition at line 199 of file device.hpp.

```
                                    {
    //return implementation->get_info<Param>(param);
  }
```

8.3.2.3.3.3 template<info::device Param> auto cl::sycl::device::get_info ( ) const [inline]

Query the device for OpenCL info::device info.

Return synchronous errors via the SYCL exception class.

Todo

Definition at line 211 of file device.hpp.

```
                        {
    // Forward to the version where the info parameter is not a template
    //return get_info<typename info::param_traits_t<info::device, Param>>(Param);
  }
```

8.3.2.3.3.4 platform cl::sycl::device::get_platform ( ) const [inline]

Return the platform of device.

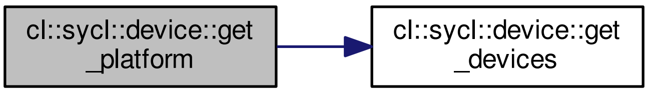
Return synchronous errors via the SYCL exception class.

Definition at line 178 of file device.hpp.

References cl::sycl::detail::__attribute__, cl::sycl::info::all, get_devices(), and cl::sycl::detail::shared_ptr_↩ implementation< device, detail::device >::implementation.

```
                                {
    return implementation->get_platform();
  }
```

Here is the call graph for this function:



8.3.2.3.3.5 bool cl::sycl::device::has_extension (const string_class & *extension*) const [inline]

Test if a specific extension is supported on the device.

Definition at line 218 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

```
                                                     {
    return implementation->has_extension(extension);
  }
```

8.3.2.3.3.6 bool cl::sycl::device::is_accelerator ( ) const [inline]

Return true if the device is an OpenCL accelerator device.

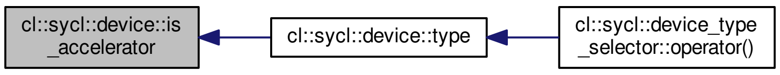
Definition at line 149 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

Referenced by type().

```
                                 {
    return implementation->is_accelerator();
  }
```

Here is the caller graph for this function:



8.3.2.3.3.7 bool cl::sycl::device::is_cpu ( ) const [inline]

Return true if the device is an OpenCL CPU device.

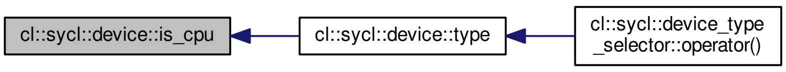
Definition at line 137 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

Referenced by type().

```
                         {
    return implementation->is_cpu();
  }
```

Here is the caller graph for this function:



8.3.2.3.3.8 bool cl::sycl::device::is_gpu ( ) const [inline]

Return true if the device is an OpenCL GPU device.

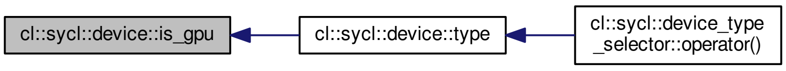
Definition at line 143 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

Referenced by type().

```
                      {
    return implementation->is_gpu();
  }
```

Here is the caller graph for this function:



8.3.2.3.3.9 bool cl::sycl::device::is_host ( ) const [inline]

Return true if the device is the SYCL host device.

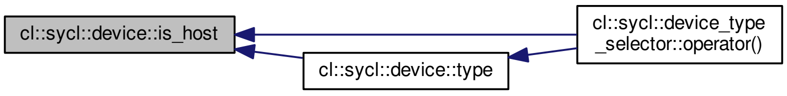
Definition at line 131 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

Referenced by cl::sycl::device_type_selector::operator()(), and type().

```
                       {
    return implementation->is_host();
  }
```

Here is the caller graph for this function:



8.3.2.3.3.10 info::device_type cl::sycl::device::type ( ) const [inline]

Return the device_type of a device.

Todo Present in Boost.Compute, to be added to the specification

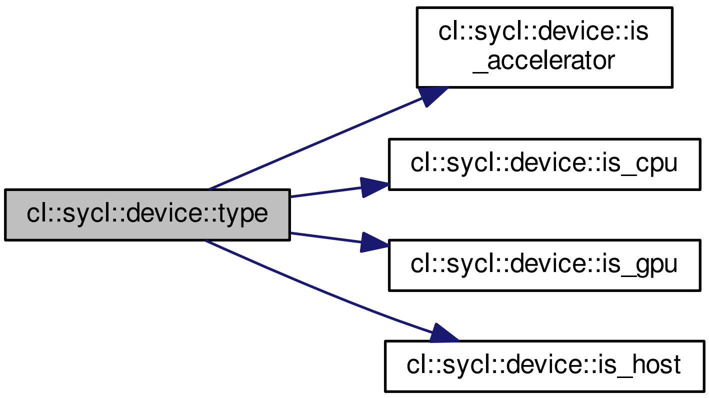
Definition at line 159 of file device.hpp.

References cl::sycl::info::accelerator, cl::sycl::info::cpu, cl::sycl::info::gpu, cl::sycl::info::host, is_accelerator(), is_cpu(), is_gpu(), and is_host().

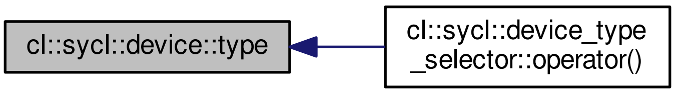
Referenced by cl::sycl::device_type_selector::operator()().

```
00159                                 {
00160     if (is_host())
00161       return info::device_type::host;
00162     else if (is_cpu())
00163       return info::device_type::cpu;
00164     else if (is_gpu())
00165       return info::device_type::gpu;
00166     else if (is_accelerator())
00167       return info::device_type::accelerator;
00168     else
00169       // \todo Put a SYCL exception
00170       throw std::domain_error("Unknown cl::sycl::info::device_type");
00171   }
```

Here is the call graph for this function:



Here is the caller graph for this function:



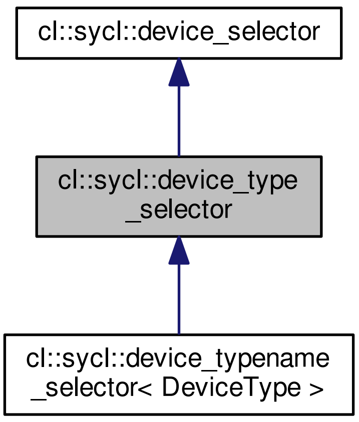
8.3.2.4 class cl::sycl::device_type_selector

A device selector by device_type.

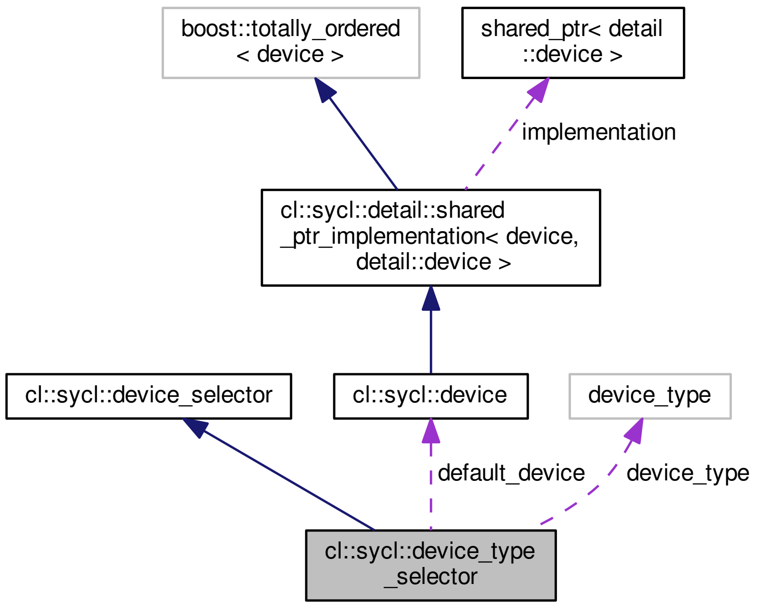
Todo To be added to the specification

Definition at line 28 of file device_selector_tail.hpp.

Inheritance diagram for cl::sycl::device_type_selector:


cl::sycl::device_selector
cl::sycl::device_type _selector
cl::sycl::device_typename _selector< DeviceType >

Collaboration diagram for cl::sycl::device_type_selector:


boost::totally_ordered < device >
shared_ptr< detail ::device >
implementation
cl::sycl::detail::shared _ptr_implementation< device, detail::device >
cl::sycl::device_selector
cl::sycl::device
device_type
default_device
device_type
cl::sycl::device_type _selector

Public Member Functions

- device_type_selector (info::device_type device_type)
- int operator() (const device &dev) const override

  This pure virtual operator allows the customization of device selection.

Private Attributes

- info::device_type device_type

  The device_type to select.
- device default_device

  Cache the default device to select with the default device selector.

8.3.2.4.1 Constructor & Destructor Documentation

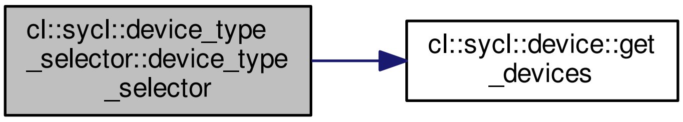
8.3.2.4.1.1 cl::sycl::device_type_selector::device_type_selector (info::device_type *device_type*) [inline]

Definition at line 44 of file device_selector_tail.hpp.

References cl::sycl::info::defaults, cl::sycl::device::get_devices(), and cl::sycl::info::opencl.

```
    : device_type { device_type } {
    // The default device selection heuristic
    if (device_type == info::device_type::defaults) {
      auto devices = device::get_devices(
info::device_type::opencl);
      /* If there is an OpenCL device, pick the first one as the
         default device, other wise it is the host device */
      if (!devices.empty())
        default_device = devices[0];
    }
}
```

Here is the call graph for this function:


cl::sycl::device_type_selector::device_type_selector
cl::sycl::device::get_devices

8.3.2.4.2 Member Function Documentation

8.3.2.4.2.1 int cl::sycl::device_type_selector::operator() (const device & *dev*) const [inline], [override], [virtual]

This pure virtual operator allows the customization of device selection.

It defines the behavior of the device_selector functor called by the SYCL runtime on device selection. It returns a "score" for each device in the system and the highest rated device will be used by the SYCL runtime.

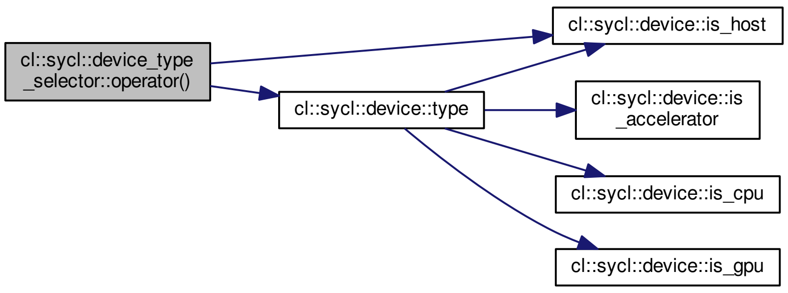
Implements cl::sycl::device_selector.

Definition at line 57 of file device_selector_tail.hpp.

References cl::sycl::info::all, cl::sycl::info::defaults, cl::sycl::device::is_host(), cl::sycl::info::opencl, and cl::sycl↩ ::device::type().

```
                                                    {
    if (device_type == info::device_type::all)
      // All devices fit all
      return 1;

    if (device_type == info::device_type::defaults)
      // Only select the default device
      return dev == default_device ? 1 : -1;

    if (device_type == info::device_type::opencl)
      // For now, any non host device is an OpenCL device
      return dev.is_host() ? -1 : 1;

    return dev.type() == device_type ? 1 : -1;
  }
```

Here is the call graph for this function:


cl::sycl::device_type _selector::operator()
cl::sycl::device::is_host
cl::sycl::device::type
cl::sycl::device::is _accelerator
cl::sycl::device::is_cpu
cl::sycl::device::is_gpu

8.3.2.4.3 Member Data Documentation

8.3.2.4.3.1 device cl::sycl::device_type_selector::default_device [private]

Cache the default device to select with the default device selector.

This is the host device at construction time and remains as is if there is no openCL device

Definition at line 40 of file device_selector_tail.hpp.

8.3.2.4.3.2 info::device_type cl::sycl::device_type_selector::device_type [private]

The device_type to select.

Definition at line 33 of file device_selector_tail.hpp.

8.3.2.5 class cl::sycl::device_typename_selector

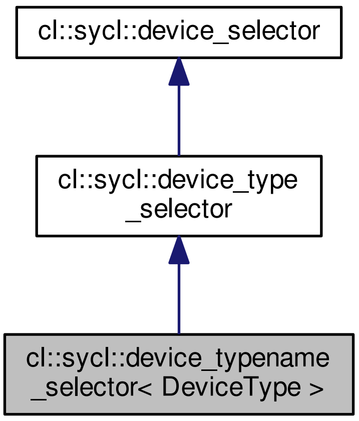
template<info::device_type DeviceType>
class cl::sycl::device_typename_selector< DeviceType >

Select a device by template device_type parameter.

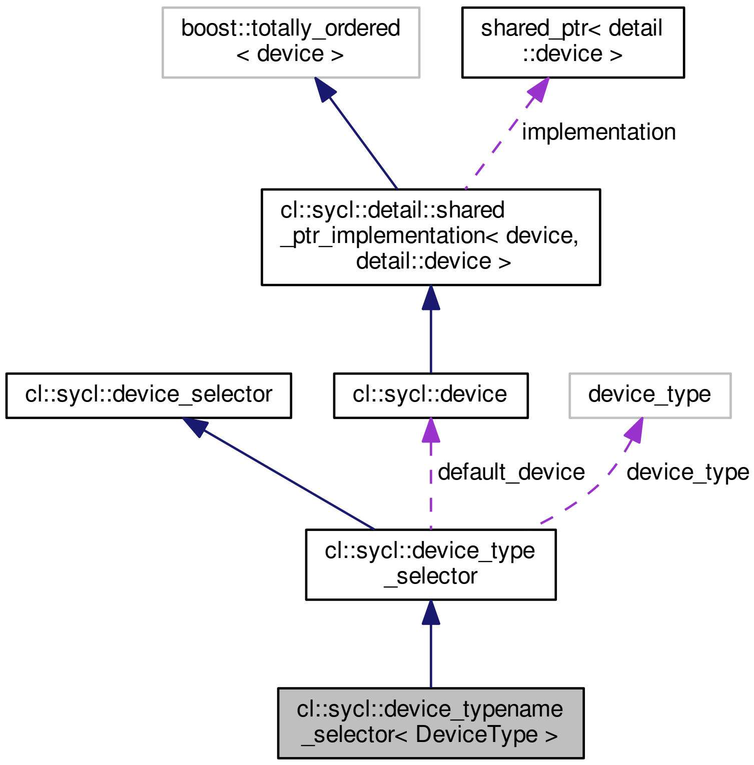
Todo To be added to the specification

Definition at line 81 of file device_selector_tail.hpp.

Inheritance diagram for cl::sycl::device_typename_selector< DeviceType >:


cl::sycl::device_selector
cl::sycl::device_type_selector
cl::sycl::device_typename_selector< DeviceType >

Collaboration diagram for cl::sycl::device_typename_selector< DeviceType >:


boost::totally_ordered < device >
shared_ptr< detail ::device >
implementation
cl::sycl::detail::shared _ptr_implementation< device, detail::device >
cl::sycl::device_selector
cl::sycl::device
device_type
default_device
device_type
cl::sycl::device_type _selector
cl::sycl::device_typename _selector< DeviceType >

Public Member Functions

- device_typename_selector ()

8.3.2.5.1 Constructor & Destructor Documentation

8.3.2.5.1.1 template<info::device_type DeviceType> **cl::sycl::device_typename_selector**< DeviceType >::**device_typename_selector ( )** [inline]

Definition at line 85 of file device_selector_tail.hpp.

```
00085 : device_type_selector { DeviceType } {}
```

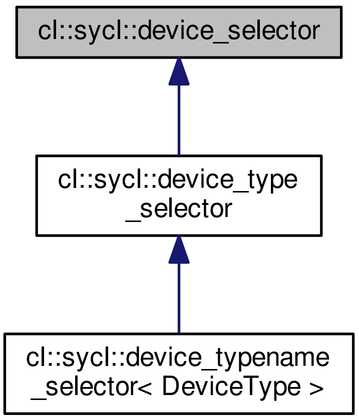
8.3.2.6 class cl::sycl::device_selector

The SYCL heuristics to select a device.

The device with the highest score is selected

Definition at line 26 of file device_selector.hpp.

Inheritance diagram for cl::sycl::device_selector:


cl::sycl::device_selector
cl::sycl::device_type _selector
cl::sycl::device_typename _selector< DeviceType >

Public Member Functions

- void select_device () const

  Returns a selected device using the functor operator defined in sub-classes operator()(const device &dev)
- virtual int operator() (const device &dev) const =0

  This pure virtual operator allows the customization of device selection.
- virtual ∼device_selector ()

  Virtual destructor so the final destructor can be called if any.

8.3.2.6.1 Constructor & Destructor Documentation

8.3.2.6.1.1 virtual cl::sycl::device_selector::∼device_selector ( ) [inline], [virtual]

Virtual destructor so the final destructor can be called if any.

Definition at line 52 of file device_selector.hpp.

```
00052 {}
```

8.3.2.6.2 Member Function Documentation

8.3.2.6.2.1 virtual int cl::sycl::device_selector::operator() (const device & *dev*) const [pure virtual]

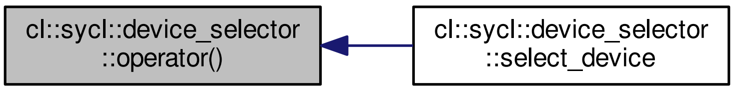
This pure virtual operator allows the customization of device selection.

It defines the behavior of the device_selector functor called by the SYCL runtime on device selection. It returns a "score" for each device in the system and the highest rated device will be used by the SYCL runtime.

Implemented in cl::sycl::device_type_selector.

Referenced by select_device().

Here is the caller graph for this function:


cl::sycl::device_selector ::operator()
cl::sycl::device_selector ::select_device

8.3.2.6.2.2 void cl::sycl::device_selector::select_device ( ) const [inline]

Returns a selected device using the functor operator defined in sub-classes operator()(const device &dev)

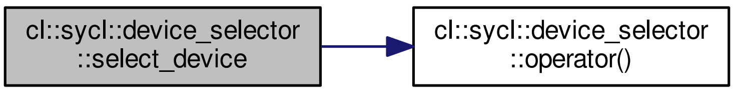
Todo Remove this from specification

Definition at line 35 of file device_selector.hpp.

References operator()().

```
00035                                  {
00036     //    return {};
00037   }
```

Here is the call graph for this function:


cl::sycl::device_selector ::select_device
cl::sycl::device_selector ::operator()

8.3.2.7 class cl::sycl::handler

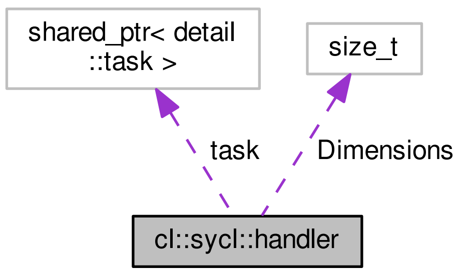
Command group handler class.

A command group handler object can only be constructed by the SYCL runtime.

All of the accessors defined in the command group scope take as a parameter an instance of the command group handler and all the kernel invocation functions are methods of this class.

Definition at line 43 of file handler.hpp.

Collaboration diagram for cl::sycl::handler:



Public Member Functions

- handler (const std::shared_ptr< detail::queue > &q)
- template<typename DataType , std::size_t Dimensions, access::mode Mode, access::target Target = access::target::global_buffer>
  void set_arg (int arg_index, accessor< DataType, Dimensions, Mode, Target > acc_obj)

  Set kernel arg for an OpenCL kernel which is used through the SYCL/OpenCL interop interface.
- template<typename T >
  void set_arg (int arg_index, T scalar_value)

  Set kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interoperability interface.
- template<typename... Ts>
  void set_args (Ts &&...args)

  Set all kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interop interface.
- template<typename KernelName = std::nullptr_t>
  void single_task (std::function< void(void)> F)

  Kernel invocation method of a kernel defined as a lambda or functor.
- TRISYCL_parallel_for_functor_GLOBAL (1) TRISYCL_parallel_for_functor_GLOBAL(2) TRISYCL_parallel↩ _for_functor_GLOBAL(3) TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(1) TRISYCL_ParallelFor↩ Functor_GLOBAL_OFFSET(2) TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(3) template< typename KernelName

  Kernel invocation method of a kernel defined as a lambda or functor, for the specified range and offset and given an id or item for indexing in the indexing space defined by range.
- std::size_t ParallelForFunctor void parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)
- template<typename KernelName = std::nullptr_t, std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_work_group (nd_range< Dimensions > r, ParallelForFunctor f)

Hierarchical kernel invocation method of a kernel defined as a lambda encoding the body of each work-group to launch.

- void single_task (kernel syclKernel)

  Kernel invocation method of a kernel defined as pointer to a kernel object, described in detail in 3.5.3.
- TRISYCL_ParallelForKernel_RANGE (1) TRISYCL_ParallelForKernel_RANGE(2) TRISYCL_ParallelFor←Kernel_RANGE(3) template< std

  Kernel invocation method of a kernel defined as pointer to a kernel object, for the specified nd_range and given an nd_item for indexing in the indexing space defined by the nd_range, described in detail in 3.5.3.

Public Attributes

- std::shared_ptr< detail::task > task

  Attach the task and accessors to it.
- std::size_t Dimensions

Private Member Functions

- template<std::size_t... Is, typename... Ts>
  void dispatch_set_arg (std::index_sequence< Is... >, Ts &&...args)

  Helper to individually call set_arg() for each argument.

8.3.2.7.1 Constructor & Destructor Documentation

8.3.2.7.1.1 cl::sycl::handler::handler (const std::shared_ptr< **detail::queue** > & *q*) [inline]

Definition at line 61 of file handler.hpp.

References Dimensions, and cl::sycl::access::global_buffer.

```
                                                {
    // Create a new task for this command_group
    task = std::make_shared<detail::task>(q);
  }
```

8.3.2.7.2 Member Function Documentation

8.3.2.7.2.1 template<std::size_t... Is, typename... Ts> void cl::sycl::handler::dispatch_set_arg (std::index_sequence< Is... > , Ts &&... *args*) [inline], [private]

Helper to individually call set_arg() for each argument.

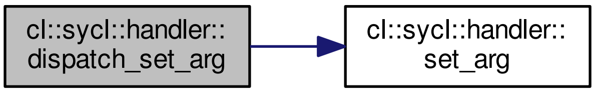
Definition at line 129 of file handler.hpp.

References set_arg().

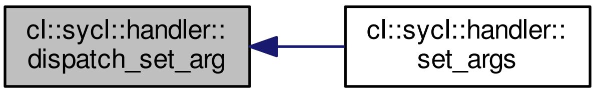
Referenced by set_args().

```
                                                                  {
    // Use an intermediate tuple to ease individual argument access
    auto &&t = std::make_tuple(std::forward<Ts>(args)...);
    // Dispatch individual set_arg() for each argument
    auto just_to_evaluate = {
      0 /*< At least 1 element to deal with empty set_args() */,
      ( set_arg(Is, std::forward<Ts>(std::get<Is>(t))), 0)...
    };
    // Remove the warning about unused variable
    static_cast<void>(just_to_evaluate);
  }
```

Here is the call graph for this function:



Here is the caller graph for this function:



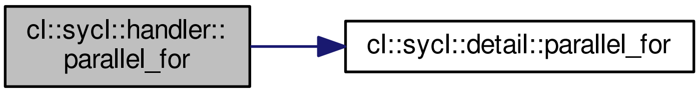
8.3.2.7.2.2 std::size_t ParallelForFunctor void cl::sycl::handler::parallel_for (nd_range< Dimensions > *r*, ParallelForFunctor *f*) [inline]

Definition at line 281 of file handler.hpp.

References cl::sycl::detail::parallel_for().

```
                                                                {
    task->schedule(detail::trace_kernel<KernelName>([=] {
          detail::parallel_for(r, f);
        }));
  }
```

Here is the call graph for this function:



8.3.2.7.2.3 template<typename KernelName = std::nullptr_t, std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::handler::parallel_for_work_group (nd_range< Dimensions > *r*, ParallelForFunctor *f*) [inline]

Hierarchical kernel invocation method of a kernel defined as a lambda encoding the body of each work-group to launch.

May contain multiple kernel built-in parallel_for_work_item functions representing the execution on each work-item.

Launch num_work_groups work-groups of runtime-defined size. Described in detail in 3.5.3.

Parameters

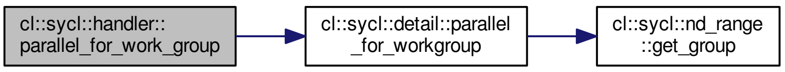
r	defines the iteration space with the work-group layout and offset
Dimensions	dimensionality of the iteration space
f	is the kernel functor to execute
ParallelForFunctor	is the kernel functor type
KernelName	is a class type that defines the name to be used for the underlying kernel

Definition at line 312 of file handler.hpp.

References cl::sycl::detail::parallel_for_workgroup().

```
                                                            {
    task->schedule(detail::trace_kernel<KernelName>([=] {
          detail::parallel_for_workgroup(r, f); }));
  }
```

Here is the call graph for this function:


cl::sycl::handler::parallel_for_work_group
cl::sycl::detail::parallel_for_workgroup
cl::sycl::nd_range::get_group

8.3.2.7.2.4 template<typename DataType , std::size_t Dimensions, access::mode Mode, access::target Target = access::target::global_buffer> void cl::sycl::handler::set_arg (int *arg_index,* accessor< DataType, Dimensions, Mode, Target > *acc_obj*) [inline]

Set kernel arg for an OpenCL kernel which is used through the SYCL/OpenCL interop interface.

The index value specifies which parameter of the OpenCL kernel is being set and the accessor object, which OpenCL buffer or image is going to be given as kernel argument.

Todo Update the specification to use a ref && to the accessor instead?

Todo It is not that clean to have set_arg() associated to a command handler. Rethink the specification?

Todo It seems more logical to have these methods on kernel instead

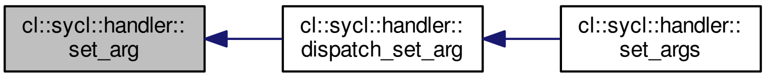
Definition at line 86 of file handler.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

Referenced by dispatch_set_arg().

```
                                                                  {
    /* Before running the kernel, make sure the cl_mem behind this
       accessor is up-to-date on the device if needed and pass it to
       the kernel.

       Explicitly capture task by copy instead of having this captured
       by reference and task by reference by side effect */
    task->add_prelude([=, task = task] {
        acc_obj.implementation->copy_in_cl_buffer();
        task->get_kernel().get_boost_compute()
          .set_arg(arg_index, acc_obj.implementation->get_cl_buffer());
      });
    /* After running the kernel, make sure the cl_mem behind this
       accessor is up-to-date on the host if needed */
    task->add_postlude([=] {
        acc_obj.implementation->copy_back_cl_buffer();
      });
  }
```

Here is the caller graph for this function:


cl::sycl::handler::set_arg
cl::sycl::handler::dispatch_set_arg
cl::sycl::handler::set_args

8.3.2.7.2.5 template<typename T > void cl::sycl::handler::set_arg (int *arg_index*, T *scalar_value*) [inline]

Set kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interoperability interface.

The index value specifies which parameter of the OpenCL kernel is being set and the accessor object, which OpenCL buffer or image is going to be given as kernel argument.

Todo It is not that clean to have set_arg() associated to a command handler. Rethink the specification?

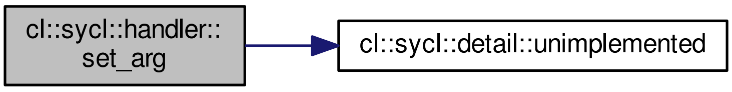
Todo To be implemented

Definition at line 120 of file handler.hpp.

References cl::sycl::detail::unimplemented().

```
                                        {
    detail::unimplemented();
  }
```

Here is the call graph for this function:



8.3.2.7.2.6 template<typename... Ts> void cl::sycl::handler::set_args (Ts &&... *args*) [inline]

Set all kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interop interface.

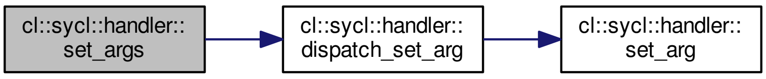
Todo Update the specification to add this function according to `https://cvs.khronos.org/bugzilla/show↩_bug.cgi?id=15978` proposal

Definition at line 150 of file handler.hpp.

References dispatch_set_arg().

```
                             {
    /* Construct a set of increasing argument index to be able to call
       the real set_arg */
    dispatch_set_arg(std::make_index_sequence<sizeof...(Ts)>{},
                           std::forward<Ts>(args)...);
  }
```

Here is the call graph for this function:



8.3.2.7.2.7 template<typename KernelName = std::nullptr_t> void cl::sycl::handler::single_task (std::function< void(void)> *F*) [inline]

Kernel invocation method of a kernel defined as a lambda or functor.

If it is a lambda function or the functor type is globally visible there is no need for the developer to provide a kernel name type (typename KernelName) for it, as described in 3.5.3

SYCL single_task launches a computation without parallelism at launch time.

Parameters

F	specify the kernel to be launched as a single_task
KernelName	is a class type that defines the name to be used for the underlying kernel

Definition at line 173 of file handler.hpp.

```
                                                {
    task->schedule(detail::trace_kernel<KernelName>(F));
  }
```

8.3.2.7.2.8 void cl::sycl::handler::single_task (kernel *syclKernel*) [inline]

Kernel invocation method of a kernel defined as pointer to a kernel object, described in detail in 3.5.3.

Todo Add in the spec a version taking a kernel and a functor, to have host fall-back

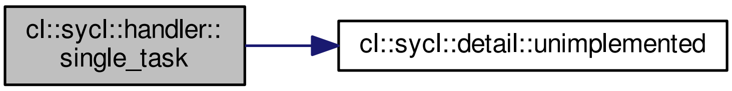
Todo To be implemented

Definition at line 327 of file handler.hpp.

References cl::sycl::detail::unimplemented().

```
                                      {
    detail::unimplemented();
  }
```

Here is the call graph for this function:



8.3.2.7.2.9 cl::sycl::handler::TRISYCL_parallel_for_functor_GLOBAL (1)

Kernel invocation method of a kernel defined as a lambda or functor, for the specified range and offset and given an id or item for indexing in the indexing space defined by range.

If it is a lambda function or the if the functor type is globally visible there is no need for the developer to provide a kernel name type (typename KernelName) for it, as described in detail in 3.5.3

Parameters

global_size	is the global size of the range<>
offset	is the offset to be add to the id<> during iteration
f	is the kernel functor to execute
ParallelForFunctor	is the kernel functor type
KernelName	is a class type that defines the name to be used for the underlying kernel

Unfortunately, to have implicit conversion to work on the range, the function can not be templated, so instantiate it for all the dimensionsKernel invocation method of a kernel defined as a lambda or functor, for the specified nd_range and given an nd_item for indexing in the indexing space defined by the nd_range

If it is a lambda function or the if the functor type is globally visible there is no need for the developer to provide a kernel name type (typename KernelName) for it, as described in detail in 3.5.3

Parameters

r	defines the iteration space with the work-group layout and offset
Dimensions	dimensionality of the iteration space
f	is the kernel functor to execute
ParallelForFunctor	is the kernel functor type
KernelName	is a class type that defines the name to be used for the underlying kernel

8.3.2.7.2.10 cl::sycl::handler::TRISYCL_ParallelForKernel_RANGE (1) [inline]

Kernel invocation method of a kernel defined as pointer to a kernel object, for the specified nd_range and given an nd_item for indexing in the indexing space defined by the nd_range, described in detail in 3.5.3.

Todo Add in the spec a version taking a kernel and a functor, to have host fall-back

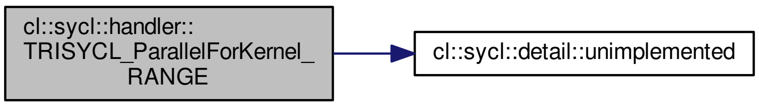
Todo To be implemented

Definition at line 365 of file handler.hpp.

References cl::sycl::detail::unimplemented().

```
                                                                            {
            detail::unimplemented();
          }
```

Here is the call graph for this function:



8.3.2.7.3 Member Data Documentation

8.3.2.7.3.1 std::size_t cl::sycl::handler::Dimensions

Definition at line 279 of file handler.hpp.

Referenced by handler().

8.3.2.7.3.2 std::shared_ptr< detail::task > cl::sycl::handler::task

Attach the task and accessors to it.

Definition at line 49 of file handler.hpp.

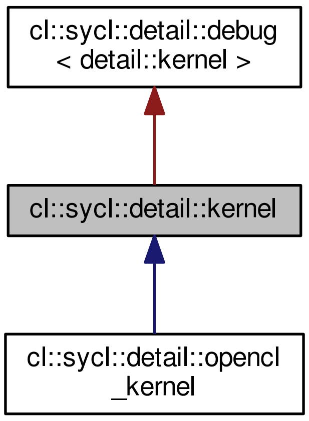
Referenced by cl::sycl::detail::add_buffer_to_task().

8.3.2.8 class cl::sycl::detail::kernel

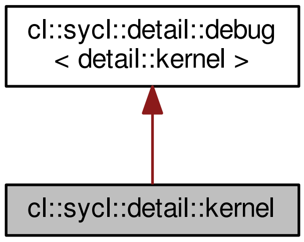
Abstract SYCL kernel.

Definition at line 31 of file kernel.hpp.

Inheritance diagram for cl::sycl::detail::kernel:


cl::sycl::detail::debug < detail::kernel >
cl::sycl::detail::kernel
cl::sycl::detail::opencl _kernel

Collaboration diagram for cl::sycl::detail::kernel:



Public Member Functions

- virtual cl_kernel get () const =0

    Return the OpenCL kernel object for this kernel.
- virtual boost::compute::kernel get_boost_compute () const =0

    Return the Boost.Compute OpenCL kernel object for this kernel.
- TRISYCL_ParallelForKernel_RANGE (1) TRISYCL_ParallelForKernel_RANGE(2) TRISYCL_ParallelFor←Kernel_RANGE(3) virtual ∼kernel()

    Return the context that this kernel is defined for.

8.3.2.8.1 Member Function Documentation

8.3.2.8.1.1 virtual cl_kernel cl::sycl::detail::kernel::get ( ) const [pure virtual]

Return the OpenCL kernel object for this kernel.

Retains a reference to the returned cl_kernel object. Caller should release it when finished.

Implemented in cl::sycl::detail::opencl_kernel.

8.3.2.8.1.2 virtual boost::compute::kernel cl::sycl::detail::kernel::get_boost_compute ( ) const [pure virtual]

Return the Boost.Compute OpenCL kernel object for this kernel.

This is an extension.

Implemented in cl::sycl::detail::opencl_kernel.

8.3.2.8.1.3 cl::sycl::detail::kernel::TRISYCL_ParallelForKernel_RANGE (1) [inline]

Return the context that this kernel is defined for.

Return the program that this kernel is part of

Definition at line 62 of file kernel.hpp.

```
00075                           {}
```

8.3.2.9 class cl::sycl::kernel

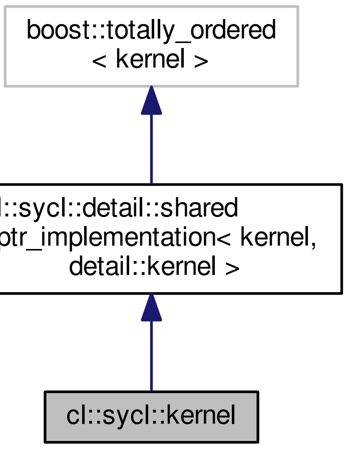
SYCL kernel.

Todo To be implemented

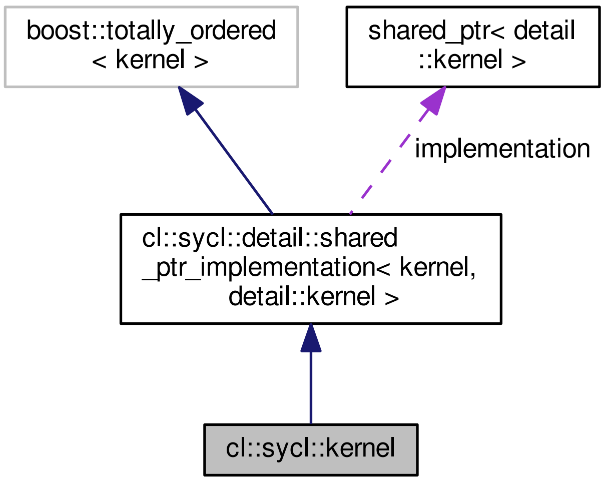
Todo Check specification

Definition at line 38 of file kernel.hpp.

Inheritance diagram for cl::sycl::kernel:


boost::totally_ordered < kernel >
cl::sycl::detail::shared _ptr_implementation< kernel, detail::kernel >
cl::sycl::kernel

Collaboration diagram for cl::sycl::kernel:


boost::totally_ordered < kernel >
shared_ptr< detail ::kernel >
implementation
cl::sycl::detail::shared _ptr_implementation< kernel, detail::kernel >
cl::sycl::kernel

Public Member Functions

- kernel ()=delete

  The default object is not valid because there is no program or.
- kernel (cl_kernel k)

  Constructor for SYCL kernel class given an OpenCL kernel object with set arguments, valid for enqueuing.
- kernel (const boost::compute::kernel &k)

  Construct a kernel class instance using a boost::compute::kernel.
- cl_kernel get () const

  Return the OpenCL kernel object for this kernel.

Private Types

- using implementation_t = detail::shared_ptr_implementation< kernel, detail::kernel >

Friends

- class handler

Additional Inherited Members

8.3.2.9.1 Member Typedef Documentation

8.3.2.9.1.1 using cl::sycl::kernel::implementation_t = detail::shared_ptr_implementation<kernel, detail::kernel> [private]

Definition at line 45 of file kernel.hpp.

8.3.2.9.2 Constructor & Destructor Documentation

8.3.2.9.2.1 cl::sycl::kernel::kernel ( ) [delete]

The default object is not valid because there is no program or.

```
cl_kernel
```

associated with it

8.3.2.9.2.2 cl::sycl::kernel::kernel (cl_kernel *k*) [inline]

Constructor for SYCL kernel class given an OpenCL kernel object with set arguments, valid for enqueuing.

Retains a reference to the `cl_kernel` object. The Caller should release the passed cl_kernel object when it is no longer needed.

Definition at line 67 of file kernel.hpp.

```
00067 : kernel { boost::compute::kernel { k } } {}
```

8.3.2.9.2.3 cl::sycl::kernel::kernel (const boost::compute::kernel & *k*) [inline]

Construct a kernel class instance using a boost::compute::kernel.

This is a triSYCL extension for boost::compute interoperation.

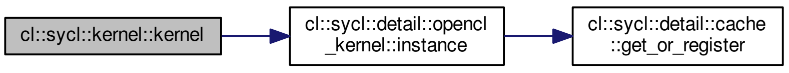
Return synchronous errors via the SYCL exception class.

Definition at line 76 of file kernel.hpp.

References cl::sycl::detail::opencl_kernel::instance().

```
00077     : implementation_t { detail::opencl_kernel::instance(k)
      } {}
```

Here is the call graph for this function:


cl::sycl::kernel::kernel
cl::sycl::detail::opencl_kernel::instance
cl::sycl::detail::cache::get_or_register

8.3.2.9.3 Member Function Documentation

8.3.2.9.3.1 cl_kernel cl::sycl::kernel::get ( ) const [inline]

Return the OpenCL kernel object for this kernel.

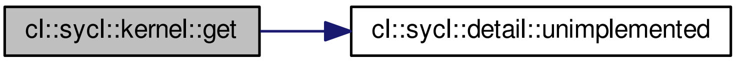
Retains a reference to the returned cl_kernel object. Caller should release it when finished.

Definition at line 85 of file kernel.hpp.

References cl::sycl::detail::shared_ptr_implementation< kernel, detail::kernel >::implementation, and cl::sycl↩ ::detail::unimplemented().

```
                            {
    return implementation->get();
  }
```

Here is the call graph for this function:



8.3.2.9.4 Friends And Related Function Documentation

8.3.2.9.4.1 friend class handler [friend]

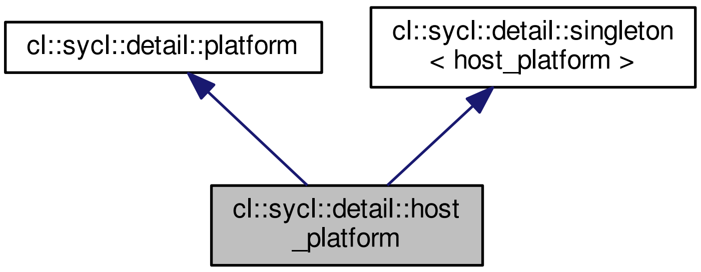
Definition at line 51 of file kernel.hpp.

8.3.2.10 class cl::sycl::detail::host_platform

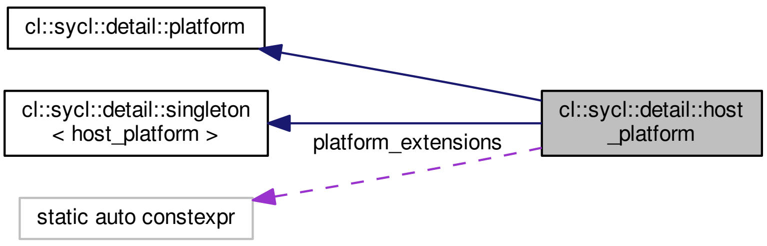
SYCL host platform.

Definition at line 31 of file host_platform.hpp.

Inheritance diagram for cl::sycl::detail::host_platform:


cl::sycl::detail::platform
cl::sycl::detail::singleton < host_platform >
cl::sycl::detail::host _platform

Collaboration diagram for cl::sycl::detail::host_platform:



Public Member Functions

- cl_platform_id get () const override

  Return the cl_platform_id of the underlying OpenCL platform.
- bool is_host () const override

  Return true since this platform is the SYCL host platform.
- string_class get_info_string (info::platform param) const override

  Returning the information parameters for the host platform implementation.
- bool has_extension (const string_class &extension) const override

  Specify whether a specific extension is supported on the platform.

Static Private Attributes

- static auto constexpr platform_extensions = "Xilinx_blocking_pipes"

Additional Inherited Members

8.3.2.10.1 Member Function Documentation

8.3.2.10.1.1 cl_platform_id cl::sycl::detail::host_platform::get ( ) const [inline], [override], [virtual]

Return the cl_platform_id of the underlying OpenCL platform.

This throws an error since there is no OpenCL platform associated to the host platform.

Implements cl::sycl::detail::platform.

Definition at line 45 of file host_platform.hpp.

```
                                          {
    throw non_cl_error("The host platform has no OpenCL platform");
  }
```

8.3.2.10.1.2 string_class cl::sycl::detail::host_platform::get_info_string (info::platform *param*) const [inline], [override], [virtual]

Returning the information parameters for the host platform implementation.

Implements cl::sycl::detail::platform.

Definition at line 79 of file host_platform.hpp.

References cl::sycl::info::extensions, cl::sycl::info::name, platform_extensions, cl::sycl::info::profile, and cl::sycl↩ ::info::vendor.

```
                                                                    {
    switch (param) {
    case info::platform::profile:
      /* Well... Is the host platform really a full profile whereas it
         is not really OpenCL? */
      return "FULL_PROFILE";

    case info::platform::version:
      // \todo I guess it should include the software version too...
      return "2.2";

    case info::platform::name:
      return "triSYCL host platform";

    case info::platform::vendor:
      return "triSYCL Open Source project";

    case info::platform::extensions:
      return platform_extensions;

    default:
      // \todo Define some SYCL exception type for this type of errors
      throw std::invalid_argument {
        "Unknown parameter value for SYCL platform information" };
    }
  }
```

8.3.2.10.1.3 bool cl::sycl::detail::host_platform::has_extension (const string_class & *extension*) const [inline], [override], [virtual]

Specify whether a specific extension is supported on the platform.

Todo To be implemented

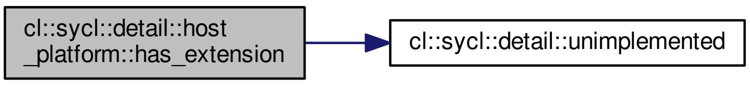
Implements cl::sycl::detail::platform.

Definition at line 111 of file host_platform.hpp.

References cl::sycl::detail::unimplemented().

```
                                                                     {
    detail::unimplemented();
    return {};
  }
```

Here is the call graph for this function:



8.3.2.10.1.4 bool cl::sycl::detail::host_platform::is_host ( ) const [inline], [override], [virtual]

Return true since this platform is the SYCL host platform.

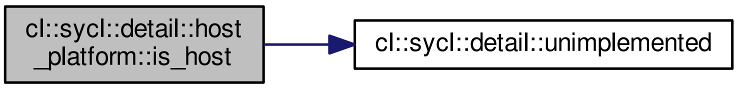
Implements cl::sycl::detail::platform.

Definition at line 52 of file host_platform.hpp.

References cl::sycl::info::all, and cl::sycl::detail::unimplemented().

```
                              {
    return true;
  }
```

Here is the call graph for this function:


cl::sycl::detail::host _platform::is_host
cl::sycl::detail::unimplemented

8.3.2.10.2 Member Data Documentation

8.3.2.10.2.1 auto constexpr cl::sycl::detail::host_platform::platform_extensions = "Xilinx_blocking_pipes" [static], [private]

Definition at line 35 of file host_platform.hpp.

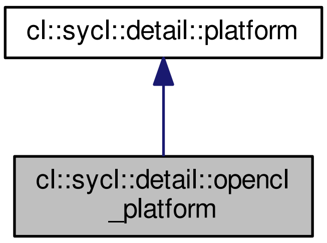
Referenced by get_info_string().

8.3.2.11 class cl::sycl::detail::opencl_platform

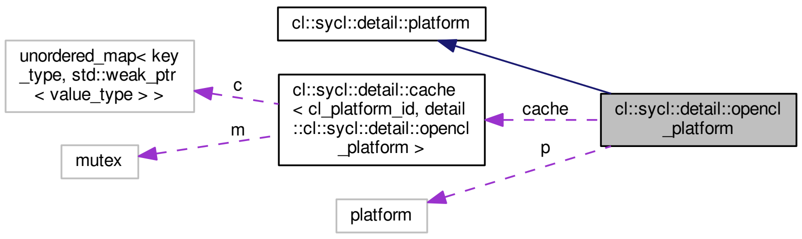
SYCL OpenCL platform.

Definition at line 36 of file opencl_platform.hpp.

Inheritance diagram for cl::sycl::detail::opencl_platform:


cl::sycl::detail::platform
cl::sycl::detail::opencl _platform

Collaboration diagram for cl::sycl::detail::opencl_platform:


cl::sycl::detail::platform
unordered_map< key _type, std::weak_ptr < value_type > >
c
m
mutex
cl::sycl::detail::cache < cl_platform_id, detail ::cl::sycl::detail::opencl _platform >
cache
cl::sycl::detail::opencl _platform
p
platform

Public Member Functions

- cl_platform_id get () const override

  Return the cl_platform_id of the underlying OpenCL platform.
- bool is_host () const override

  Return false since an OpenCL platform is not the SYCL host platform.
- string_class get_info_string (info::platform param) const override

  Returning the information string parameters for the OpenCL platform.
- bool has_extension (const string_class &extension) const override

  Specify whether a specific extension is supported on the platform.
- ∼opencl_platform () override

  Unregister from the cache on destruction.

Static Public Member Functions

- static std::shared_ptr< opencl_platform > instance (const boost::compute::platform &p)

Private Member Functions

- opencl_platform (const boost::compute::platform &p)

  Only the instance factory can built it.

Private Attributes

- boost::compute::platform p

  Use the Boost Compute abstraction of the OpenCL platform.

Static Private Attributes

- static detail::cache< cl_platform_id, detail::opencl_platform > cache

  A cache to always return the same live platform for a given OpenCL platform.

8.3.2.11.1 Constructor & Destructor Documentation

8.3.2.11.1.1 cl::sycl::detail::opencl_platform::opencl_platform (const boost::compute::platform & *p*) [inline], [private]

Only the instance factory can built it.

Definition at line 106 of file opencl_platform.hpp.

```
: p { p } {}
```

8.3.2.11.1.2 cl::sycl::detail::opencl_platform::∼opencl_platform ( ) [inline], [override]

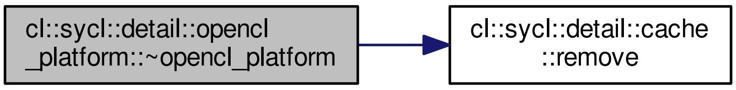
Unregister from the cache on destruction.

Definition at line 111 of file opencl_platform.hpp.

References cl::sycl::detail::__attribute__, cache, and cl::sycl::detail::cache< Key, Value >::remove().

```
                                   {
    cache.remove(p.id());
  }
```

Here is the call graph for this function:



8.3.2.11.2 Member Function Documentation

8.3.2.11.2.1 cl_platform_id cl::sycl::detail::opencl_platform::get ( ) const [inline], [override], [virtual]

Return the cl_platform_id of the underlying OpenCL platform.

Implements cl::sycl::detail::platform.

Definition at line 51 of file opencl_platform.hpp.

```
                                           {
    return p.id();
  }
```

8.3.2.11.2.2 string_class cl::sycl::detail::opencl_platform::get_info_string (info::platform *param*) const [inline], [override], [virtual]

Returning the information string parameters for the OpenCL platform.

Implements cl::sycl::detail::platform.

Definition at line 82 of file opencl_platform.hpp.

```
                                                                      {
      /* Use the fact that the triSYCL info values are the same as the
         OpenCL ones used in Boost.Compute to just cast the enum class
         to the int value */
      return p.get_info<std::string>(static_cast<cl_platform_info>(param));
    }
```

8.3.2.11.2.3 bool cl::sycl::detail::opencl_platform::has_extension (const string_class & *extension*) const [inline], [override], [virtual]

Specify whether a specific extension is supported on the platform.

Implements cl::sycl::detail::platform.

Definition at line 91 of file opencl_platform.hpp.

```
                                                                       {
      return p.supports_extension(extension);
    }
```

8.3.2.11.2.4 static std::shared_ptr< opencl_platform > cl::sycl::detail::opencl_platform::instance (const boost::compute::platform & *p*) [inline], [static]

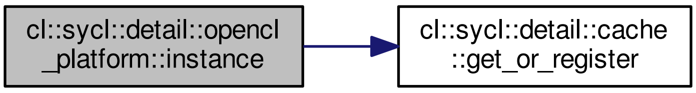
Definition at line 98 of file opencl_platform.hpp.

References cl::sycl::detail::cache< Key, Value >::get_or_register().

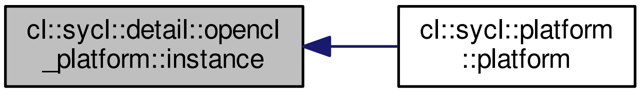
Referenced by cl::sycl::platform::platform().

```
                                                   {
      return cache.get_or_register(p.id(),
                                  [&] { return new opencl_platform {
 p }; });
    }
```

Here is the call graph for this function:



Here is the caller graph for this function:



8.3.2.11.2.5 bool cl::sycl::detail::opencl_platform::is_host ( ) const [inline], [override], [virtual]

Return false since an OpenCL platform is not the SYCL host platform.

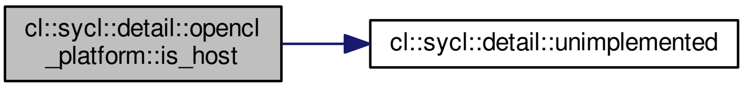
Implements cl::sycl::detail::platform.

Definition at line 57 of file opencl_platform.hpp.

References cl::sycl::info::all, and cl::sycl::detail::unimplemented().

```
                                     {
    return false;
  }
```

Here is the call graph for this function:



8.3.2.11.3 Member Data Documentation

8.3.2.11.3.1 detail::cache<cl_platform_id, detail::opencl_platform> cl::sycl::detail::opencl_platform::cache [static], [private]

A cache to always return the same live platform for a given OpenCL platform.

C++11 guaranties the static construction is thread-safe

Definition at line 46 of file opencl_platform.hpp.

Referenced by ∼opencl_platform().

8.3.2.11.3.2 boost::compute::platform cl::sycl::detail::opencl_platform::p [private]

Use the Boost Compute abstraction of the OpenCL platform.

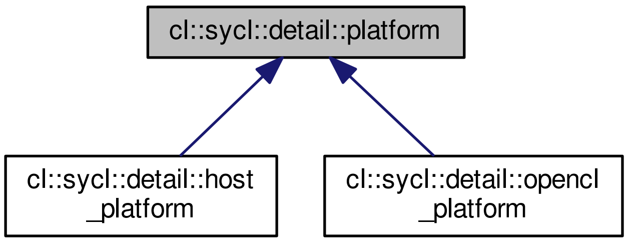
Definition at line 39 of file opencl_platform.hpp.

8.3.2.12 class cl::sycl::detail::platform

An abstract class representing various models of SYCL platforms.

Definition at line 25 of file platform.hpp.

Inheritance diagram for cl::sycl::detail::platform:


cl::sycl::detail::platform
cl::sycl::detail::host _platform
cl::sycl::detail::opencl _platform

Public Member Functions

- virtual cl_platform_id get () const =0

  Return the cl_platform_id of the underlying OpenCL platform.
- virtual bool is_host () const =0

  Return true if the platform is a SYCL host platform.
- virtual string_class get_info_string (info::platform param) const =0

  Query the platform for OpenCL string info::platform info.
- virtual bool has_extension (const string_class &extension) const =0

  Specify whether a specific extension is supported on the platform.
- virtual ∼platform ()

8.3.2.12.1 Constructor & Destructor Documentation

8.3.2.12.1.1 virtual cl::sycl::detail::platform::∼platform ( ) [inline], [virtual]

Definition at line 48 of file platform.hpp.

```
00048 {}
```

8.3.2.12.2 Member Function Documentation

8.3.2.12.2.1 virtual cl_platform_id cl::sycl::detail::platform::get ( ) const [pure virtual]

Return the cl_platform_id of the underlying OpenCL platform.

Implemented in cl::sycl::detail::opencl_platform, and cl::sycl::detail::host_platform.

8.3.2.12.2.2 virtual string_class cl::sycl::detail::platform::get_info_string (info::platform *param*) const [pure virtual]

Query the platform for OpenCL string info::platform info.

Implemented in cl::sycl::detail::opencl_platform, and cl::sycl::detail::host_platform.

8.3.2.12.2.3 virtual bool cl::sycl::detail::platform::has_extension (const string_class & *extension*) const [pure virtual]

Specify whether a specific extension is supported on the platform.

Implemented in cl::sycl::detail::host_platform, and cl::sycl::detail::opencl_platform.

8.3.2.12.2.4 virtual bool cl::sycl::detail::platform::is_host ( ) const [pure virtual]

Return true if the platform is a SYCL host platform.

Implemented in cl::sycl::detail::opencl_platform, and cl::sycl::detail::host_platform.

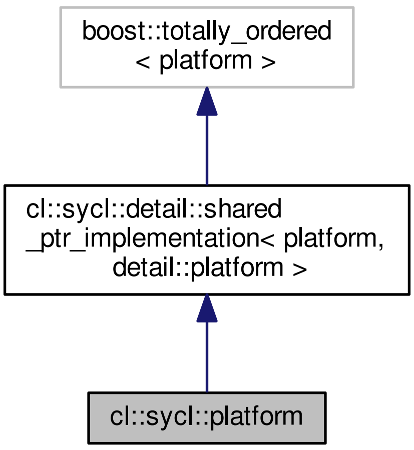
8.3.2.13 class cl::sycl::platform

Abstract the OpenCL platform.

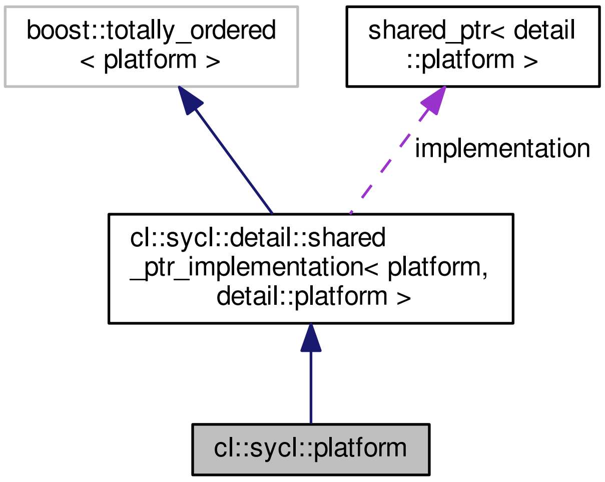
Todo triSYCL Implementation

Definition at line 43 of file platform.hpp.

Inheritance diagram for cl::sycl::platform:


boost::totally_ordered < platform >
cl::sycl::detail::shared _ptr_implementation< platform, detail::platform >
cl::sycl::platform

Collaboration diagram for cl::sycl::platform:


boost::totally_ordered < platform >
shared_ptr< detail ::platform >
implementation
cl::sycl::detail::shared _ptr_implementation< platform, detail::platform >
cl::sycl::platform

Public Member Functions

- platform ()
  Default constructor for platform which is the host platform.
- platform (cl_platform_id platform_id)
  Construct a platform class instance using cl_platform_id of the OpenCL device.
- platform (const boost::compute::platform &p)
  Construct a platform class instance using a boost::compute::platform.
- platform (const device_selector &dev_selector)
  Construct a platform object from the device selected by a device selector of the user's choice.
- cl_platform_id get () const
  Returns the cl_platform_id of the underlying OpenCL platform.
- template<typename ReturnT >
  ReturnT get_info (info::platform param) const
  Get the OpenCL information about the requested parameter.
- template<info::platform Param>
  info::param_traits< info::platform, Param >::type get_info () const
  Get the OpenCL information about the requested template parameter.
- bool has_extension (const string_class &extension) const
  Test if an extension is available on the platform.
- bool is_host () const
  Test if this platform is a host platform.

Static Public Member Functions

- static vector_class< platform > get_platforms ()
  Get the list of all the platforms available to the application.

Private Types

- using implementation_t = detail::shared_ptr_implementation< platform, detail::platform >

Additional Inherited Members

8.3.2.13.1 Member Typedef Documentation

8.3.2.13.1.1 using cl::sycl::platform::implementation_t = detail::shared_ptr_implementation<platform, detail::platform> [private]

Definition at line 50 of file platform.hpp.

8.3.2.13.2 Constructor & Destructor Documentation

8.3.2.13.2.1 cl::sycl::platform::platform ( ) [inline]

Default constructor for platform which is the host platform.

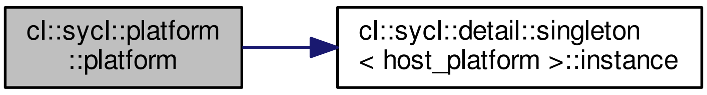
Returns errors via the SYCL exception class.

Definition at line 62 of file platform.hpp.

References cl::sycl::detail::singleton< host_platform >::instance().

```
00062 : implementation_t { detail::host_platform::instance() } {}
```

Here is the call graph for this function:


cl::sycl::platform ::platform
cl::sycl::detail::singleton < host_platform >::instance

8.3.2.13.2.2 cl::sycl::platform::platform (cl_platform_id *platform_id*) [inline]

Construct a platform class instance using cl_platform_id of the OpenCL device.

Return synchronous errors via the SYCL exception class.

Retain a reference to the OpenCL platform.

Definition at line 73 of file platform.hpp.

```
00074     : platform { boost::compute::platform { platform_id } } {}
```

8.3.2.13.2.3 cl::sycl::platform::platform (const boost::compute::platform & *p*) [inline]

Construct a platform class instance using a boost::compute::platform.

This is a triSYCL extension for boost::compute interoperation.

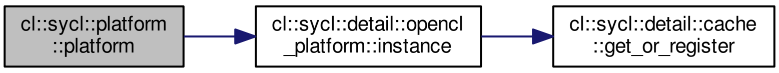
Return synchronous errors via the SYCL exception class.

Definition at line 83 of file platform.hpp.

References cl::sycl::detail::opencl_platform::instance().

```
00084     : implementation_t { detail::opencl_platform::instance
      (p) } {}
```

Here is the call graph for this function:


cl::sycl::platform ::platform
cl::sycl::detail::opencl _platform::instance
cl::sycl::detail::cache ::get_or_register

8.3.2.13.2.4 cl::sycl::platform::platform (const **device_selector** & *dev_selector*) [inline], [explicit]

Construct a platform object from the device selected by a device selector of the user's choice.

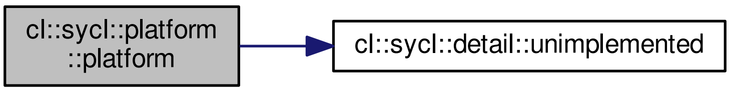
Returns errors via the SYCL exception class.

Definition at line 93 of file platform.hpp.

References cl::sycl::detail::unimplemented().

```
00093                                                          {
00094     detail::unimplemented();
00095   }
```

Here is the call graph for this function:


cl::sycl::platform ::platform
cl::sycl::detail::unimplemented

8.3.2.13.3 Member Function Documentation

8.3.2.13.3.1 cl_platform_id cl::sycl::platform::get ( ) const [inline]

Returns the cl_platform_id of the underlying OpenCL platform.

If the platform is not a valid OpenCL platform, for example if it is the SYCL host, an exception is thrown

Todo Define a SYCL exception for this

Definition at line 106 of file platform.hpp.

References cl::sycl::detail::shared_ptr_implementation< platform, detail::platform >::implementation.

```
00106                              {
00107     return implementation->get();
00108   }
```

8.3.2.13.3.2 template<typename ReturnT > ReturnT cl::sycl::platform::get_info (info::platform *param*) const [inline]

Get the OpenCL information about the requested parameter.

Todo Add to the specification

Definition at line 147 of file platform.hpp.

References cl::sycl::detail::shared_ptr_implementation< platform, detail::platform >::implementation.

```
00147                                               {
00148     // Only strings are needed here
00149     return implementation->get_info_string(param);
00150   }
```

8.3.2.13.3.3 template<info::platform Param> info::param_traits<info::platform, Param>::type cl::sycl::platform::get_info ( ) const [inline]

Get the OpenCL information about the requested template parameter.

Definition at line 156 of file platform.hpp.

```
00156                   {
00157     /* Forward to the implementation without using template parameter
00158        but with a parameter instead, since it is incompatible with
00159        virtual function and because fortunately only strings are
00160        needed here */
00161     return get_info<typename info::param_traits<info::platform,
00162                                                  Param>::type>(Param);
00163   }
```

8.3.2.13.3.4 static vector_class< platform > cl::sycl::platform::get_platforms ( ) [inline], [static]

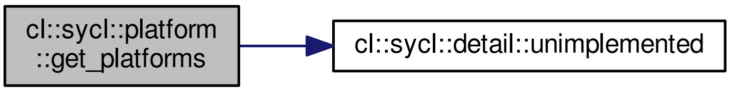
Get the list of all the platforms available to the application.

Definition at line 113 of file platform.hpp.

References cl::sycl::info::all, and cl::sycl::detail::unimplemented().

```
                                              {
    // Start with the default platform
    vector_class<platform> platforms { {} };

#ifdef TRISYCL_OPENCL
    // Then add all the OpenCL platforms
    for (const auto &d : boost::compute::system::platforms())
      platforms.emplace_back(d);
#endif

    return platforms;
  }
```

Here is the call graph for this function:


cl::sycl::platform ::get_platforms
cl::sycl::detail::unimplemented

8.3.2.13.3.5 bool cl::sycl::platform::has_extension (const string_class & *extension*) const [inline]

Test if an extension is available on the platform.

Definition at line 167 of file platform.hpp.

References cl::sycl::detail::shared_ptr_implementation< platform, detail::platform >::implementation.

```
                                                        {
    return implementation->has_extension(extension);
  }
```

8.3.2.13.3.6 bool cl::sycl::platform::is_host ( ) const [inline]

Test if this platform is a host platform.

Definition at line 173 of file platform.hpp.

References cl::sycl::detail::shared_ptr_implementation< platform, detail::platform >::implementation.

```
                          {
    return implementation->is_host();
  }
```

8.3.2.14 class cl::sycl::queue

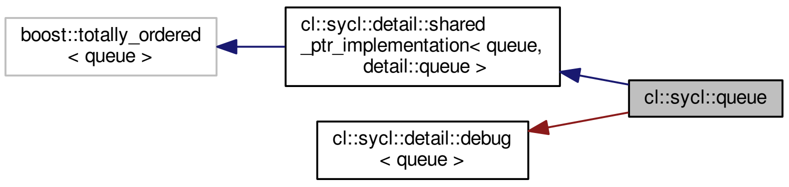
SYCL queue, similar to the OpenCL queue concept.

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

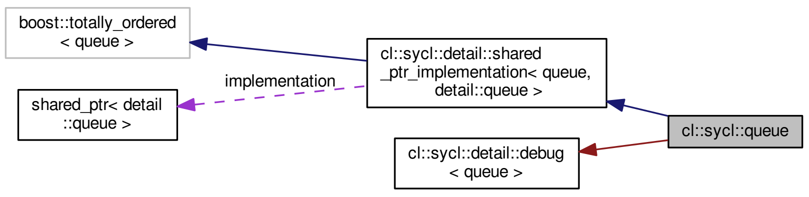
Todo All the queue methods should return a queue& instead of void to it is possible to chain opoerations

Definition at line 80 of file queue.hpp.

Inheritance diagram for cl::sycl::queue:


boost::totally_ordered < queue >
cl::sycl::detail::shared _ptr_implementation< queue, detail::queue >
cl::sycl::queue
cl::sycl::detail::debug < queue >

Collaboration diagram for cl::sycl::queue:


boost::totally_ordered < queue >
shared_ptr< detail ::queue >
implementation
cl::sycl::detail::shared _ptr_implementation< queue, detail::queue >
cl::sycl::queue
cl::sycl::detail::debug < queue >

Public Member Functions

- queue ()

  Default constructor for platform which is the host platform.
- queue (async_handler asyncHandler)

  This constructor creates a SYCL queue from an OpenCL queue.
- queue (const device_selector &deviceSelector, async_handler asyncHandler=nullptr)

  Creates a queue for the device provided by the device selector.
- queue (const device &syclDevice, async_handler asyncHandler=nullptr)

  A queue is created for syclDevice.
- queue (const context &syclContext, const device_selector &deviceSelector, async_handler async↩ Handler=nullptr)

  This constructor chooses a device based on the provided device_selector, which needs to be in the given context.
- queue (const context &syclContext, const device &syclDevice, async_handler asyncHandler=nullptr)

Creates a command queue using clCreateCommandQueue from a context and a device.
- queue (const context &syclContext, const device &syclDevice, info::queue_profiling profilingFlag, async_↩ handler asyncHandler=nullptr)

  Creates a command queue using clCreateCommandQueue from a context and a device.
- queue (const cl_command_queue &q, async_handler ah=nullptr)

  This constructor creates a SYCL queue from an OpenCL queue.
- queue (const boost::compute::command_queue &q, async_handler ah=nullptr)

  Construct a queue instance using a boost::compute::command_queue.
- cl_command_queue get () const

  Return the underlying OpenCL command queue after doing a retain.
- context get_context () const

  Return the SYCL queue's context.
- device get_device () const

  Return the SYCL device the queue is associated with.
- bool is_host () const

  Return whether the queue is executing on a SYCL host device.
- void wait ()

  Performs a blocking wait for the completion all enqueued tasks in the queue.
- void wait_and_throw ()

  Perform a blocking wait for the completion all enqueued tasks in the queue.
- void throw_asynchronous ()

  Checks to see if any asynchronous errors have been produced by the queue and if so reports them by passing them to the async_handler passed to the queue on construction.
- template<info::queue param>
  info::param_traits< info::queue, param >::type get_info () const

  Queries the platform for cl_command_queue info.
- handler_event submit (std::function< void(handler &)> cgf)

  Submit a command group functor to the queue, in order to be scheduled for execution on the device.
- handler_event submit (std::function< void(handler &)> cgf, queue &secondaryQueue)

  Submit a command group functor to the queue, in order to be scheduled for execution on the device.

Private Types

- using implementation_t = detail::shared_ptr_implementation< queue, detail::queue >

Additional Inherited Members

8.3.2.14.1 Member Typedef Documentation

8.3.2.14.1.1 using cl::sycl::queue::implementation_t = detail::shared_ptr_implementation<queue, detail::queue> [private]

Definition at line 87 of file queue.hpp.

8.3.2.14.2 Constructor & Destructor Documentation

8.3.2.14.2.1 cl::sycl::queue::queue ( ) [inline]

Default constructor for platform which is the host platform.

Returns errors via the SYCL exception class.

Definition at line 98 of file queue.hpp.

```
00098 : implementation_t { new detail::host_queue } {}
```

8.3.2.14.2.2 cl::sycl::queue::queue (async_handler *asyncHandler*) [inline], [explicit]

This constructor creates a SYCL queue from an OpenCL queue.

At construction it does a retain on the queue memory object.

Retain a reference to the cl_command_queue object. Caller should release the passed cl_command_queue object when it is no longer needed.

Return synchronous errors regarding the creation of the queue and report asynchronous errors via the async_↩
handler callback function in conjunction with the synchronization and throw methods.

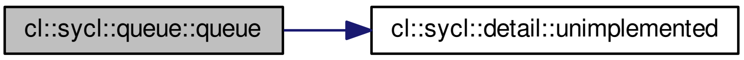
Note that the default case asyncHandler = nullptr is handled by the default constructor.

Definition at line 117 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
                                                : queue { } {
    detail::unimplemented();
  }
```

Here is the call graph for this function:



8.3.2.14.2.3 cl::sycl::queue::queue (const device_selector & *deviceSelector*, async_handler *asyncHandler* = nullptr) [inline]

Creates a queue for the device provided by the device selector.

If no device is selected, an error is reported.

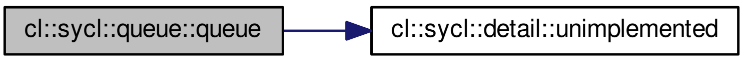
Return synchronous errors regarding the creation of the queue and report asynchronous errors via the async_↩
handler callback function if and only if there is an async_handler provided.

Definition at line 130 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
                                                 : queue { } {
    detail::unimplemented();
  }
```

Here is the call graph for this function:



8.3.2.14.2.4 cl::sycl::queue::queue (const device & *syclDevice,* async_handler *asyncHandler* = `nullptr`) `[inline]`

A queue is created for syclDevice.

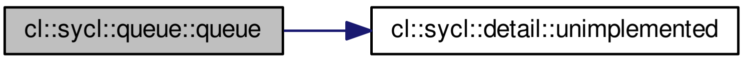
Return asynchronous errors via the async_handler callback function.

Definition at line 140 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
                                                          : queue { } {
    detail::unimplemented();
  };
```

Here is the call graph for this function:



8.3.2.14.2.5 cl::sycl::queue::queue (const context & *syclContext,* const device_selector & *deviceSelector,* async_handler *asyncHandler* = `nullptr`) `[inline]`

This constructor chooses a device based on the provided device_selector, which needs to be in the given context.

If no device is selected, an error is reported.

Return synchronous errors regarding the creation of the queue.

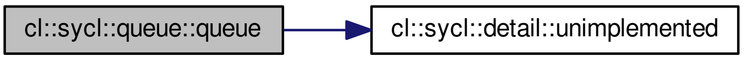
If and only if there is an asyncHandler provided, it reports asynchronous errors via the async_handler callback function in conjunction with the synchronization and throw methods.

Definition at line 157 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
                                                          : queue { } {
    detail::unimplemented();
  }
```

Here is the call graph for this function:



8.3.2.14.2.6 cl::sycl::queue::queue (const context & *syclContext,* const device & *syclDevice,* async_handler *asyncHandler* = nullptr) [inline]

Creates a command queue using clCreateCommandQueue from a context and a device.

Return synchronous errors regarding the creation of the queue.

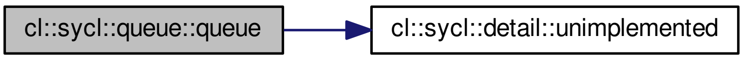
If and only if there is an asyncHandler provided, it reports asynchronous errors via the async_handler callback function in conjunction with the synchronization and throw methods.

Definition at line 173 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
                                                  : queue { } {
    detail::unimplemented();
  }
```

Here is the call graph for this function:



8.3.2.14.2.7 cl::sycl::queue::queue (const context & *syclContext,* const device & *syclDevice,* info::queue_profiling *profilingFlag,* async_handler *asyncHandler* = nullptr) [inline]

Creates a command queue using clCreateCommandQueue from a context and a device.

It enables profiling on the queue if the profilingFlag is set to true.

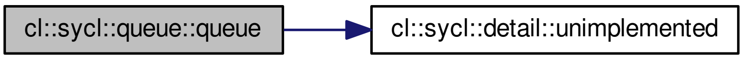
Return synchronous errors regarding the creation of the queue. If and only if there is an asyncHandler provided, it reports asynchronous errors via the async_handler callback function in conjunction with the synchronization and throw methods.

Definition at line 191 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
                                                  : queue { } {
    detail::unimplemented();
  }
```

Here is the call graph for this function:



8.3.2.14.2.8 cl::sycl::queue::queue (const cl_command_queue & *q*, async_handler *ah* = nullptr) [inline]

This constructor creates a SYCL queue from an OpenCL queue.

At construction it does a retain on the queue memory object.

Return synchronous errors regarding the creation of the queue. If and only if there is an async_handler provided, it reports asynchronous errors via the async_handler callback function in conjunction with the synchronization and throw methods.

Definition at line 209 of file queue.hpp.

```
00210     : queue { boost::compute::command_queue { q }, ah } {}
```

8.3.2.14.2.9 cl::sycl::queue::queue (const boost::compute::command_queue & *q*, async_handler *ah* = nullptr) [inline]

Construct a queue instance using a boost::compute::command_queue.

This is a triSYCL extension for boost::compute interoperation.

Return synchronous errors via the SYCL exception class.

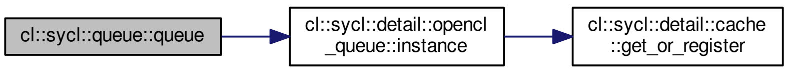
Todo Deal with handler

Definition at line 221 of file queue.hpp.

References cl::sycl::detail::opencl_queue::instance().

```
00222     : implementation_t { detail::opencl_queue::instance(q) }
      {}
```

Here is the call graph for this function:


cl::sycl::queue::queue
cl::sycl::detail::opencl_queue::instance
cl::sycl::detail::cache::get_or_register

8.3.2.14.3 Member Function Documentation

8.3.2.14.3.1 cl_command_queue cl::sycl::queue::get ( ) const [inline]

Return the underlying OpenCL command queue after doing a retain.

This memory object is expected to be released by the developer.

Retain a reference to the returned cl_command_queue object.

Caller should release it when finished.

If the queue is a SYCL host queue then an exception is thrown.

Definition at line 237 of file queue.hpp.

```
00237                           {
00238     return implementation->get();
00239   }
```

8.3.2.14.3.2 context cl::sycl::queue::get_context ( ) const [inline]

Return the SYCL queue's context.

Report errors using SYCL exception classes.

Definition at line 247 of file queue.hpp.

```
                                {
    return implementation->get_context();
  }
```

8.3.2.14.3.3 device cl::sycl::queue::get_device ( ) const [inline]

Return the SYCL device the queue is associated with.

Report errors using SYCL exception classes.

Definition at line 256 of file queue.hpp.

```
                              {
    return implementation->get_device();
  }
```

8.3.2.14.3.4 template<info::queue param> info::param_traits<info::queue, param>::type cl::sycl::queue::get_info ( ) const [inline]

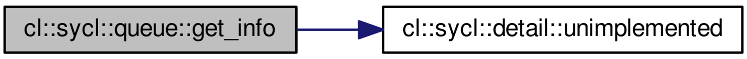
Queries the platform for cl_command_queue info.

Definition at line 306 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
                                                                          {
    detail::unimplemented();
    return {};
  }
```

Here is the call graph for this function:



8.3.2.14.3.5 bool cl::sycl::queue::is_host ( ) const [inline]

Return whether the queue is executing on a SYCL host device.

Definition at line 262 of file queue.hpp.

```
                          {
    return implementation->is_host();
  }
```

8.3.2.14.3.6 handler_event cl::sycl::queue::submit (std::function< void(handler &)> *cgf*) [inline]

Submit a command group functor to the queue, in order to be scheduled for execution on the device.

Use an explicit functor parameter taking a handler& so we can use "auto" in submit() lambda parameter.

Todo Add in the spec an implicit conversion of handler_event to queue& so it is possible to chain operations on the queue

Todo Update the spec to replace std::function by a templated type to avoid memory allocation

Definition at line 324 of file queue.hpp.

```
                                                     {
    handler command_group_handler { implementation };
    cgf(command_group_handler);
    return {};
  }
```

8.3.2.14.3.7 handler_event cl::sycl::queue::submit (std::function< void(handler &)> *cgf,* queue & *secondaryQueue*) [inline]

Submit a command group functor to the queue, in order to be scheduled for execution on the device.

On kernel error, this command group functor, then it is scheduled for execution on the secondary queue.

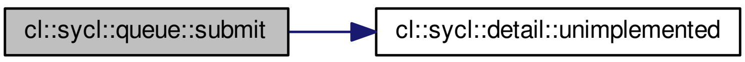
Return a command group functor event, which is corresponds to the queue the command group functor is being enqueued on.

Definition at line 340 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
                                                                               {
    detail::unimplemented();
    // Since it is not implemented, always submit on the main queue
    return submit(cgf);
  }
```

Here is the call graph for this function:



8.3.2.14.3.8 void cl::sycl::queue::throw_asynchronous ( ) [inline]

Checks to see if any asynchronous errors have been produced by the queue and if so reports them by passing them to the async_handler passed to the queue on construction.

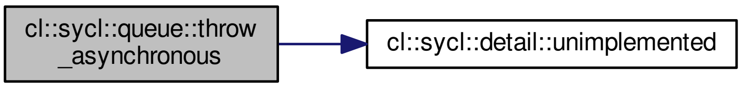
If no async_handler was provided then asynchronous exceptions will be lost.

Definition at line 299 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00299                            {
00300     detail::unimplemented();
00301   }
```

Here is the call graph for this function:


cl::sycl::queue::throw_asynchronous
cl::sycl::detail::unimplemented

8.3.2.14.3.9 void cl::sycl::queue::wait ( ) [inline]

Performs a blocking wait for the completion all enqueued tasks in the queue.

Synchronous errors will be reported through SYCL exceptions.

Definition at line 272 of file queue.hpp.

```
00272              {
00273     implementation->wait_for_kernel_execution();
00274   }
```

8.3.2.14.3.10 void cl::sycl::queue::wait_and_throw ( ) [inline]

Perform a blocking wait for the completion all enqueued tasks in the queue.

Synchronous errors will be reported via SYCL exceptions.

Asynchronous errors will be passed to the async_handler passed to the queue on construction.

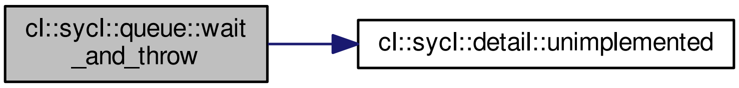
If no async_handler was provided then asynchronous exceptions will be lost.

Definition at line 287 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00287                        {
00288     detail::unimplemented();
00289   }
```

Here is the call graph for this function:


cl::sycl::queue::wait_and_throw
cl::sycl::detail::unimplemented

8.3.3 Typedef Documentation

8.3.3.1 using cl::sycl::cpu_selector = typedef device_typename_selector<info::device_type::cpu>

`#include <include/CL/sycl/device_selector/detail/device_selector_tail.hpp>`

Select devices according to device type info::device::device_type::cpu from all the available devices and heuristics.

If no OpenCL CPU device is found the selector fails.

Definition at line 112 of file device_selector_tail.hpp.

8.3.3.2 using cl::sycl::default_selector = typedef device_typename_selector<info::device_type::defaults>

`#include <include/CL/sycl/device_selector/detail/device_selector_tail.hpp>`

Devices selected by heuristics of the system.

If no OpenCL device is found then it defaults to the SYCL host device.

Definition at line 94 of file device_selector_tail.hpp.

8.3.3.3 using cl::sycl::info::device_exec_capabilities = typedef unsigned int

`#include <include/CL/sycl/info/device.hpp>`

Definition at line 183 of file device.hpp.

8.3.3.4 using cl::sycl::info::device_fp_config = typedef unsigned int

`#include <include/CL/sycl/info/device.hpp>`

Definition at line 182 of file device.hpp.

8.3.3.5 using cl::sycl::info::device_queue_properties = typedef unsigned int

`#include <include/CL/sycl/info/device.hpp>`

Definition at line 184 of file device.hpp.

8.3.3.6 using cl::sycl::gpu_selector = typedef device_typename_selector<info::device_type::gpu>

`#include <include/CL/sycl/device_selector/detail/device_selector_tail.hpp>`

Select devices according to device type info::device::device_type::gpu from all the available OpenCL devices.

If no OpenCL GPU device is found the selector fails.

Select the best GPU, if any.

Definition at line 104 of file device_selector_tail.hpp.

8.3.3.7 using cl::sycl::host_selector = typedef device_typename_selector<info::device_type::host>

```
#include <include/CL/sycl/device_selector/detail/device_selector_tail.hpp>
```

Selects the SYCL host CPU device that does not require an OpenCL runtime.

Definition at line 118 of file device_selector_tail.hpp.

8.3.4 Enumeration Type Documentation

8.3.4.1 enum cl::sycl::info::device : int [strong]

```
#include <include/CL/sycl/info/device.hpp>
```

Device information descriptors.

From specs/latex/headers/deviceInfo.h in the specification

Todo Should be unsigned int?

Enumerator

device_type

vendor_id

max_compute_units

max_work_item_dimensions

max_work_item_sizes

max_work_group_size

preferred_vector_width_char

preferred_vector_width_short

preferred_vector_width_int

preferred_vector_width_long_long

preferred_vector_width_float

preferred_vector_width_double

preferred_vector_width_half

native_vector_witdth_char

native_vector_witdth_short

native_vector_witdth_int

native_vector_witdth_long_long

native_vector_witdth_float

native_vector_witdth_double

native_vector_witdth_half

max_clock_frequency

address_bits

max_mem_alloc_size

image_support

max_read_image_args

max_write_image_args

image2d_max_height

image2d_max_width

image3d_max_height

image3d_max_widht

image3d_mas_depth

image_max_buffer_size

image_max_array_size

max_samplers

max_parameter_size

mem_base_addr_align

single_fp_config

double_fp_config

global_mem_cache_type

global_mem_cache_line_size

global_mem_cache_size

global_mem_size

max_constant_buffer_size

max_constant_args

local_mem_type

local_mem_size

error_correction_support

host_unified_memory

profiling_timer_resolution

endian_little

is_available

is_compiler_available

is_linker_available

execution_capabilities

queue_properties

built_in_kernels

platform

name

vendor

driver_version

profile

device_version

opencl_version

extensions

printf_buffer_size

preferred_interop_user_sync

parent_device

partition_max_sub_devices

partition_properties

partition_affinity_domain

partition_type

reference_count

Definition at line 52 of file device.hpp.

```
                   : int {
  device_type,
  vendor_id,
  max_compute_units,
  max_work_item_dimensions,
  max_work_item_sizes,
  max_work_group_size,
  preferred_vector_width_char,
  preferred_vector_width_short,
  preferred_vector_width_int,
  preferred_vector_width_long_long,
  preferred_vector_width_float,
  preferred_vector_width_double,
  preferred_vector_width_half,
  native_vector_witdth_char,
  native_vector_witdth_short,
  native_vector_witdth_int,
  native_vector_witdth_long_long,
  native_vector_witdth_float,
  native_vector_witdth_double,
  native_vector_witdth_half,
  max_clock_frequency,
  address_bits,
  max_mem_alloc_size,
  image_support,
  max_read_image_args,
  max_write_image_args,
  image2d_max_height,
  image2d_max_width,
  image3d_max_height,
  image3d_max_widht,
  image3d_mas_depth,
  image_max_buffer_size,
  image_max_array_size,
  max_samplers,
  max_parameter_size,
  mem_base_addr_align,
  single_fp_config,
  double_fp_config,
  global_mem_cache_type,
  global_mem_cache_line_size,
  global_mem_cache_size,
  global_mem_size,
  max_constant_buffer_size,
  max_constant_args,
  local_mem_type,
  local_mem_size,
  error_correction_support,
  host_unified_memory,
  profiling_timer_resolution,
  endian_little,
  is_available,
  is_compiler_available,
  is_linker_available,
  execution_capabilities,
  queue_properties,
  built_in_kernels,
  platform,
  name,
  vendor,
  driver_version,
  profile,
  device_version,
  opencl_version,
  extensions,
  printf_buffer_size,
  preferred_interop_user_sync,
  parent_device,
  partition_max_sub_devices,
  partition_properties,
  partition_affinity_domain,
  partition_type,
  reference_count
};
```

8.3.4.2 enum cl::sycl::info::device_affinity_domain : int [strong]

`#include <include/CL/sycl/info/device.hpp>`

Enumerator

unsupported

numa

L4_cache

L3_cache

L2_cache

next_partitionable

Definition at line 135 of file device.hpp.

```
                                    : int {
  unsupported,
  numa,
  L4_cache,
  L3_cache,
  L2_cache,
  next_partitionable
};
```

8.3.4.3 enum cl::sycl::info::device_execution_capabilities : unsigned int [strong]

`#include <include/CL/sycl/info/device.hpp>`

Enumerator

exec_kernel

exec_native_kernel

Definition at line 176 of file device.hpp.

```
                                           : unsigned int {
  exec_kernel,
  exec_native_kernel
};
```

8.3.4.4 enum cl::sycl::info::device_partition_property : int [strong]

`#include <include/CL/sycl/info/device.hpp>`

Enumerator

unsupported

partition_equally

partition_by_counts

partition_by_affinity_domain

partition_affinity_domain_next_partitionable

Definition at line 127 of file device.hpp.

```
                                       : int {
  unsupported,
  partition_equally,
  partition_by_counts,
  partition_by_affinity_domain,
  partition_affinity_domain_next_partitionable
};
```

8.3.4.5 enum cl::sycl::info::device_partition_type : int [strong]

`#include <include/CL/sycl/info/device.hpp>`

Enumerator

no_partition

numa

L4_cache

L3_cache

L2_cache

L1_cache

Definition at line 144 of file device.hpp.

```
                                       : int {
  no_partition,
  numa,
  L4_cache,
  L3_cache,
  L2_cache,
  L1_cache
};
```

8.3.4.6 enum cl::sycl::info::device_type : unsigned int [strong]

`#include <include/CL/sycl/info/device.hpp>`

Type of devices.

To be used either to define a device type or to select more broadly a kind of device

Todo To be moved in the specification from platform to device

Todo Add opencl to the specification

Todo there is no accelerator_selector and custom_accelerator

Enumerator

cpu

gpu

accelerator

custom

defaults

host

opencl

all

Definition at line 34 of file device.hpp.

```
                         : unsigned int {
  cpu,
  gpu,
  accelerator,
  custom,
  defaults,
  host,
  opencl,
  all
};
```

8.3.4.7 enum cl::sycl::info::fp_config : int [strong]

`#include <include/CL/sycl/info/device.hpp>`

Enumerator

denorm

inf_nan

round_to_nearest

round_to_zero

round_to_inf

fma

correctly_rounded_divide_sqrt

soft_float

Definition at line 159 of file device.hpp.

```
                      : int {
  denorm,
  inf_nan,
  round_to_nearest,
  round_to_zero,
  round_to_inf,
  fma,
  correctly_rounded_divide_sqrt,
  soft_float
};
```

8.3.4.8 enum cl::sycl::info::global_mem_cache_type : int [strong]

`#include <include/CL/sycl/info/device.hpp>`

Enumerator

none

read_only

write_only

Definition at line 170 of file device.hpp.

```
                                  : int {
  none,
  read_only,
  write_only
};
```

8.3.4.9 enum cl::sycl::info::local_mem_type : int [strong]

```
#include <include/CL/sycl/info/device.hpp>
```

Enumerator

none

local

global

Definition at line 153 of file device.hpp.

```
                           : int {
  none,
  local,
  global
};
```

8.3.4.10 enum cl::sycl::info::platform : unsigned int [strong]

```
#include <include/CL/sycl/info/platform.hpp>
```

Platform information descriptors.

A SYCL platform can be queried for all of the following information using the get_info function.

In this implementation, the values are mapped to OpenCL values to avoid further remapping later when OpenCL is used

Enumerator

TRISYCL_SKIP_OPENCL Returns the profile name (as a string_class) supported by the implementation. Can be either FULL PROFILE or EMBEDDED PROFILE.

TRISYCL_SKIP_OPENCL Returns the OpenCL software driver version string in the form major number.↩ minor number (as a string_class)

TRISYCL_SKIP_OPENCL Returns the name of the platform (as a string_class)

TRISYCL_SKIP_OPENCL Returns the string provided by the platform vendor (as a string_class)

TRISYCL_SKIP_OPENCL Returns a space-separated list of extension names supported by the platform (as a string_class)

Definition at line 31 of file platform.hpp.

```
                    : unsigned int {
  /** Returns the profile name (as a string_class) supported by the
      implementation.

      Can be either FULL PROFILE or EMBEDDED PROFILE.
  */
  profile TRISYCL_SKIP_OPENCL(= CL_PLATFORM_PROFILE),

  /** Returns the OpenCL software driver version string in the form major
      number.minor number (as a string_class)
  */
  version TRISYCL_SKIP_OPENCL(= CL_PLATFORM_VERSION),

  /** Returns the name of the platform (as a string_class)
  */
  name TRISYCL_SKIP_OPENCL(= CL_PLATFORM_NAME),

  /** Returns the string provided by the platform vendor (as a string_class)
  */
  vendor TRISYCL_SKIP_OPENCL(= CL_PLATFORM_VENDOR),

  /** Returns a space-separated list of extension names supported by the
      platform (as a string_class)
  */
  extensions TRISYCL_SKIP_OPENCL(= CL_PLATFORM_EXTENSIONS),

#if CL_SYCL_LANGUAGE_VERSION >= 220 && defined(CL_VERSION_2_1)
  /** Returns the resolution of the host timer in nanoseconds as used by
      clGetDeviceAndHostTimer
  */
  host_timer_resolution
    TRISYCL_SKIP_OPENCL(= CL_PLATFORM_HOST_TIMER_RESOLUTION)
#endif
};
```

8.3.5 Function Documentation

8.3.5.1 detail::cache<cl_kernel, detail::opencl_kernel> opencl_kernel::cache cl::sycl::detail::__attribute__ ((weak))

```
#include <include/CL/sycl/kernel/detail/opencl_kernel.hpp>
```

8.3.5.2 vector_class< device > cl::sycl::device::get_devices (info::device_type *device_type* = info::device_type::all) [static]

```
#include <include/CL/sycl/device.hpp>
```

Return a list of all available devices.

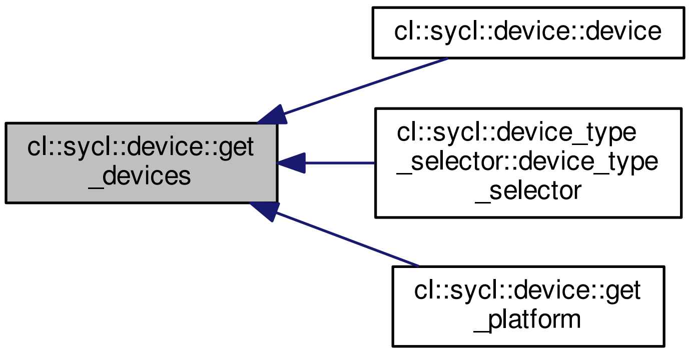
Return synchronous errors via SYCL exception classes.

Definition at line 26 of file device_tail.hpp.

Referenced by cl::sycl::device::device(), cl::sycl::device_type_selector::device_type_selector(), and cl::sycl::device::get_platform().

```
                                               {
  // Start with the default device
  vector_class<device> devices = { {} };

#ifdef TRISYCL_OPENCL
  // Then add all the OpenCL devices
  for (const auto &d : boost::compute::system::devices())
    devices.emplace_back(d);
#endif

  // The selected devices
  vector_class<device> sd;
  device_type_selector s { device_type };

  // Return the devices with the good criterion according to the selector
  std::copy_if(devices.begin(), devices.end(), std::back_inserter(sd),
               [&](const device &e ) { return s(e) >= 0; });
  return sd;
}
```

Here is the caller graph for this function:


cl::sycl::device::get _devices
cl::sycl::device::device
cl::sycl::device_type _selector::device_type _selector
cl::sycl::device::get _platform

8.3.6 Variable Documentation

8.3.6.1 detail::cache< cl_command_queue, detail::opencl_queue > opencl_queue::cache cl::sycl::detail::__attribute__ ((weak))

```
#include <include/CL/sycl/device/detail/opencl_device.hpp>
```

Referenced by cl::sycl::device::get_platform(), cl::sycl::detail::opencl_kernel::TRISYCL_ParallelForKernel_RANG←E(), cl::sycl::detail::opencl_device::∼opencl_device(), cl::sycl::detail::opencl_platform::∼opencl_platform(), and cl←::sycl::detail::opencl_queue::∼opencl_queue().

8.4 Helpers to do array and tuple conversion

Classes

- struct cl::sycl::detail::expand_to_vector< V, Tuple, expansion >

  Allows optional expansion of a 1-element tuple to a V::dimension tuple to replicate scalar values in vector initialization. More...
- struct cl::sycl::detail::expand_to_vector< V, Tuple, true >

  Specialization in the case we ask for expansion. More...

Functions

- template<typename V , typename Tuple , size_t... Is>
  std::array< typename V::element_type, V::dimension > cl::sycl::detail::tuple_to_array_iterate (Tuple t, std↩ ::index_sequence< Is... >)

  Helper to construct an array from initializer elements provided as a tuple.
- template<typename V , typename Tuple >
  auto cl::sycl::detail::tuple_to_array (Tuple t)

  Construct an array from initializer elements provided as a tuple.
- static auto cl::sycl::detail::expand_to_vector< V, Tuple, expansion >::expand (Tuple t)
- template<typename Value , size_t... Is>
  static auto cl::sycl::detail::expand_to_vector< V, Tuple, true >::fill_tuple (Value e, std::index_sequence< Is... >)

  Construct a tuple from a value.
- static auto cl::sycl::detail::expand_to_vector< V, Tuple, true >::expand (Tuple t)

  We expand the 1-element tuple by replicating into a tuple with the size of the vector.
- template<typename V , typename Tuple >
  auto cl::sycl::detail::expand (Tuple t)

  Create the array data of V from a tuple of initializer.

8.4.1 Detailed Description

8.4.2 Class Documentation

8.4.2.1 struct cl::sycl::detail::expand_to_vector

template<typename V, typename Tuple, bool expansion = false>
struct cl::sycl::detail::expand_to_vector< V, Tuple, expansion >

Allows optional expansion of a 1-element tuple to a V::dimension tuple to replicate scalar values in vector initialization.

Definition at line 65 of file array_tuple_helpers.hpp.

Static Public Member Functions

- static auto expand (Tuple t)

8.4.2.2 struct cl::sycl::detail::expand_to_vector< V, Tuple, true >

template<typename V, typename Tuple>
struct cl::sycl::detail::expand_to_vector< V, Tuple, true >

Specialization in the case we ask for expansion.

Definition at line 77 of file array_tuple_helpers.hpp.

Static Public Member Functions

- template<typename Value , size_t... Is>
  static auto fill_tuple (Value e, std::index_sequence< Is... >)

  Construct a tuple from a value.
- static auto expand (Tuple t)

  We expand the 1-element tuple by replicating into a tuple with the size of the vector.

8.4.3 Function Documentation

8.4.3.1 template<typename V , typename Tuple , bool expansion = false> static auto cl::sycl::detail::expand_to_vector< V, Tuple, expansion >::expand (Tuple *t*) [inline], [static]

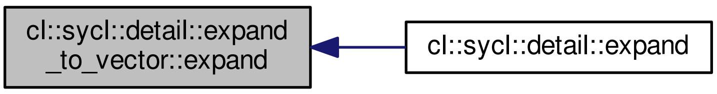
`#include <include/CL/sycl/detail/array_tuple_helpers.hpp>`

Definition at line 70 of file array_tuple_helpers.hpp.

Referenced by cl::sycl::detail::expand().

```
00070 { return t; }
```

Here is the caller graph for this function:



8.4.3.2 template<typename V , typename Tuple > static auto cl::sycl::detail::expand_to_vector< V, Tuple, true >::expand (Tuple *t*) [inline], [static]

`#include <include/CL/sycl/detail/array_tuple_helpers.hpp>`

We expand the 1-element tuple by replicating into a tuple with the size of the vector.

Definition at line 109 of file array_tuple_helpers.hpp.

```
00109                              {
00110     return fill_tuple(std::get<0>(t),
00111                       std::make_index_sequence<V::dimension>{});
00112   }
```

8.4.3.3 template<typename V , typename Tuple > auto cl::sycl::detail::expand (Tuple *t*)

```
#include <include/CL/sycl/detail/array_tuple_helpers.hpp>
```

Create the array data of V from a tuple of initializer.

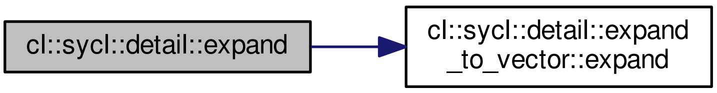
If there is only 1 initializer, this is a scalar initialization of a vector and the value is expanded to all the vector elements first.

Definition at line 123 of file array_tuple_helpers.hpp.

References cl::sycl::detail::expand_to_vector< V, Tuple, expansion >::expand().

```
00123                          {
00124   return tuple_to_array<V>(expand_to_vector<V,
00125                               decltype(t),
00126                               /* Only ask the expansion to all vector
00127                                  element if there only a scalar
00128                                  initializer */
00129                               std::tuple_size<Tuple>::value == 1>{}.expand(t));
00130 }
```

Here is the call graph for this function:



8.4.3.4 template<typename V , typename Tuple > template<typename Value , size_t... Is> static auto cl::sycl::detail::expand_to_vector< V, Tuple, true >::fill_tuple (Value *e*, std::index_sequence< Is... >) [inline], [static]

```
#include <include/CL/sycl/detail/array_tuple_helpers.hpp>
```

Construct a tuple from a value.

Parameters

value	is used to initialize each tuple element
size	is the number of elements of the tuple to be generated

The trick is to get the std::index_sequence<> that represent 0, 1,..., dimension-1 as a variadic template pack Is that we can iterate on, in this function.

Definition at line 93 of file array_tuple_helpers.hpp.

```
                                                          {
    /* The effect is like a static for-loop with Is counting from 0 to
       dimension-1 and thus replicating the pattern to have
       make_tuple( (0, e), (1, e), ... (n - 1, e) )

       Since the "," operator is just here to throw away the Is value
       (which is needed for the pack expansion...), at the end this is
       equivalent to:
       make_tuple( e, e, ..., e )
    */
    return std::make_tuple(((void)Is, e)...);
  }
```

8.4.3.5 template<typename V , typename Tuple > auto cl::sycl::detail::tuple_to_array (Tuple *t*)

`#include <include/CL/sycl/detail/array_tuple_helpers.hpp>`

Construct an array from initializer elements provided as a tuple.

Definition at line 53 of file array_tuple_helpers.hpp.

```
                               {
  /* Construct an index_sequence with 0, 1, ..., (size of the tuple-1)
     so that tuple_to_array_iterate can statically iterate on it */
  return tuple_to_array_iterate<V>(t,
                                    std::make_index_sequence<std::tuple_size<Tuple>::value>{});
}
```

8.4.3.6 template<typename V , typename Tuple , size_t... Is> std::array<typename V::element_type, V::dimension> cl::sycl::detail::tuple_to_array_iterate (Tuple *t*, std::index_sequence< Is... >)

`#include <include/CL/sycl/detail/array_tuple_helpers.hpp>`

Helper to construct an array from initializer elements provided as a tuple.

The trick is to get the std::index_sequence<> that represent 0, 1,..., dimension-1 as a variadic template pack Is that we can iterate on, in this function.

Definition at line 37 of file array_tuple_helpers.hpp.

```
                                                        {
  /* The effect is like a static for-loop with Is counting from 0 to
     dimension-1 and thus constructing a uniform initialization { }
     construction from each tuple element:
     { std::get<0>(t), std::get<1>(t), ..., std::get<dimension-1>(t) }

     The static cast is here to avoid the warning when there is a loss
     of precision, for example when initializing an int from a float.
  */
  return { { static_cast<typename V::element_type>(std::get<Is>(t))...} };
}
```

8.5 Debugging and tracing support

Classes

- struct cl::sycl::detail::debug< T >

  Class used to trace the construction, copy-construction, move-construction and destruction of classes that inherit from it. More...
- struct cl::sycl::detail::display_vector< T >

  Class used to display a vector-like type of classes that inherit from it. More...

Functions

- template<typename KernelName , typename Functor >
  auto cl::sycl::detail::trace_kernel (const Functor &f)

  Wrap a kernel functor in some tracing messages to have start/stop information when TRISYCL_TRACE_KERNEL macro is defined.

8.5.1 Detailed Description

8.5.2 Class Documentation

8.5.2.1 struct cl::sycl::detail::debug

template<typename T>
struct cl::sycl::detail::debug< T >

Class used to trace the construction, copy-construction, move-construction and destruction of classes that inherit from it.

Parameters

T	is the real type name to be used in the debug output.

Definition at line 68 of file debug.hpp.

8.5.2.2 struct cl::sycl::detail::display_vector

template<typename T>
struct cl::sycl::detail::display_vector< T >

Class used to display a vector-like type of classes that inherit from it.

Parameters

T	is the real type name to be used in the debug output.

Calling the display() method dump the values on std::cout

Definition at line 160 of file debug.hpp.

Public Member Functions

- void display () const
  To debug and test.

8.5.2.2.1 Member Function Documentation

8.5.2.2.1.1 template<typename T> void cl::sycl::detail::display_vector< T >::display ( ) const [inline]

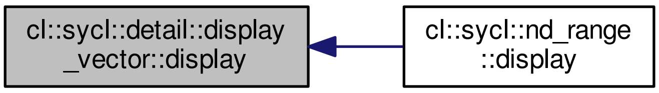
To debug and test.

Definition at line 163 of file debug.hpp.

Referenced by cl::sycl::nd_range< dims >::display().

```
                           {
#ifdef TRISYCL_DEBUG
    std::cout << boost::typeindex::type_id<T>().pretty_name() << ":";
#endif
    // Get a pointer to the real object
    for (auto e : *static_cast<const T *>(this))
      std::cout << " " << e;
    std::cout << std::endl;
  }
```

Here is the caller graph for this function:


cl::sycl::detail::display_vector::display
cl::sycl::nd_range::display

8.5.3 Function Documentation

8.5.3.1 template<typename KernelName , typename Functor > auto cl::sycl::detail::trace_kernel (const Functor & *f*)

`#include <include/CL/sycl/detail/debug.hpp>`

Wrap a kernel functor in some tracing messages to have start/stop information when TRISYCL_TRACE_KERNEL macro is defined.

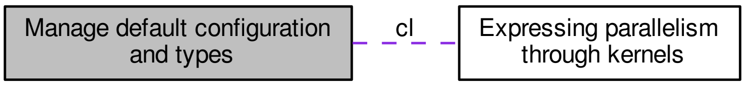
Definition at line 130 of file debug.hpp.

References TRISYCL_INTERNAL_DUMP.

```
                                     {
#ifdef TRISYCL_TRACE_KERNEL
  // Inject tracing message around the kernel
  return [=] {
    /* Since the class KernelName may just be declared and not really
       defined, just use it through a class pointer to have
       typeid().name() not complaining */
    TRISYCL_INTERNAL_DUMP(
      "Kernel started "
      << boost::typeindex::type_id<KernelName *>().pretty_name());
    f();
    TRISYCL_INTERNAL_DUMP(
      "Kernel stopped "
      << boost::typeindex::type_id<KernelName *>().pretty_name());
  };
#else
  // Identity by default
  return f;
#endif
}
```

8.6 Manage default configuration and types

Collaboration diagram for Manage default configuration and types:


Manage default configuration and types
cl
Expressing parallelism through kernels

Namespaces

- cl

  The vector type to be used as SYCL vector.

Macros

- #define CL_SYCL_LANGUAGE_VERSION 220

  This implement SYCL 2.2.
- #define CL_TRISYCL_LANGUAGE_VERSION 220

  This implement triSYCL 2.2.
- #define __SYCL_SINGLE_SOURCE__

  This source is compiled by a single source compiler.
- #define TRISYCL_SKIP_OPENCL(x) x

  Define TRISYCL_OPENCL to add OpenCL.
- #define TRISYCL_ASYNC 0

  Allow the asynchronous implementation of tasks.

8.6.1 Detailed Description

8.6.2 Macro Definition Documentation

8.6.2.1 #define __SYCL_SINGLE_SOURCE__

```
#include <include/CL/sycl/detail/global_config.hpp>
```

This source is compiled by a single source compiler.

Definition at line 28 of file global_config.hpp.

8.6.2.2 #define CL_SYCL_LANGUAGE_VERSION 220

```
#include <include/CL/sycl/detail/global_config.hpp>
```

This implement SYCL 2.2.

Definition at line 19 of file global_config.hpp.

8.6.2.3 #define CL_TRISYCL_LANGUAGE_VERSION 220

```
#include <include/CL/sycl/detail/global_config.hpp>
```

This implement triSYCL 2.2.

Definition at line 24 of file global_config.hpp.

8.6.2.4 #define TRISYCL_ASYNC 0

```
#include <include/CL/sycl/detail/global_config.hpp>
```

Allow the asynchronous implementation of tasks.

Use asynchronous tasks by default.

Is set to 0, the functors are executed synchronously.

Definition at line 58 of file global_config.hpp.

8.6.2.5 #define TRISYCL_SKIP_OPENCL(*x*) x

```
#include <include/CL/sycl/detail/global_config.hpp>
```

Define TRISYCL_OPENCL to add OpenCL.

triSYCL can indeed work without OpenCL if only host support is needed.A macro to keep some stuff in OpenCL mode

Definition at line 46 of file global_config.hpp.

8.7 Some helpers for the implementation

Classes

- struct cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >

    Define a multi-dimensional index, used for example to locate a work item or a buffer element. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >

    A small array of 1, 2 or 3 elements with the implicit constructors. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >

    Use some specializations so that some function overloads can be determined according to some implicit constructors and to have an implicit conversion from/to BasicType (such as an int typically) if dims = 1. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >

Macros

- #define TRISYCL_BOOST_OPERATOR_VECTOR_OP(op)

    Helper macro to declare a vector operation with the given side-effect operator.

Functions

- template<typename Range , typename Id >
  size_t constexpr cl::sycl::detail::linear_id (Range range, Id id, Id offset={})

    Compute a linearized array access used in the OpenCL 2 world.
- void cl::sycl::detail::unimplemented ()

    Display an "unimplemented" message.

8.7.1 Detailed Description

8.7.2 Class Documentation

8.7.2.1 struct cl::sycl::detail::small_array

template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false>
struct cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >

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

Unfortunately, even if std::array is an aggregate class allowing native list initialization, it is no longer an aggregate if we derive from an aggregate. Thus we have to redeclare the constructors.

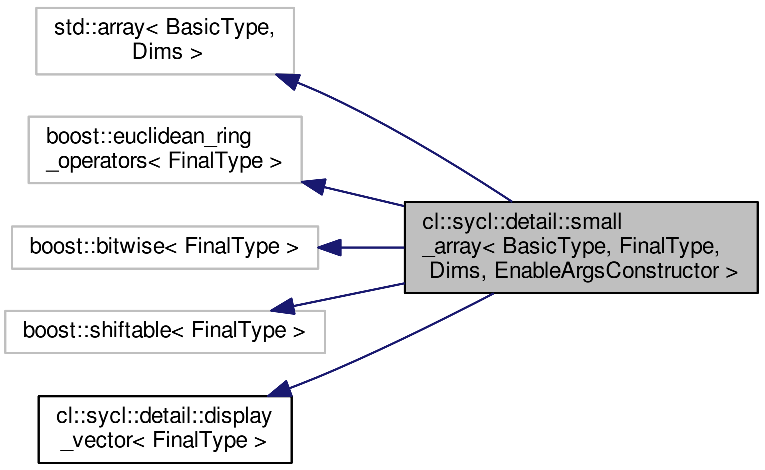
Parameters

BasicType	is the type element, such as int
Dims	is the dimension number, typically between 1 and 3
FinalType	is the final type, such as range<> or id<>, so that boost::operator can return the right type
EnableArgsConstructor	adds a constructors from Dims variadic elements when true. It is false by default.

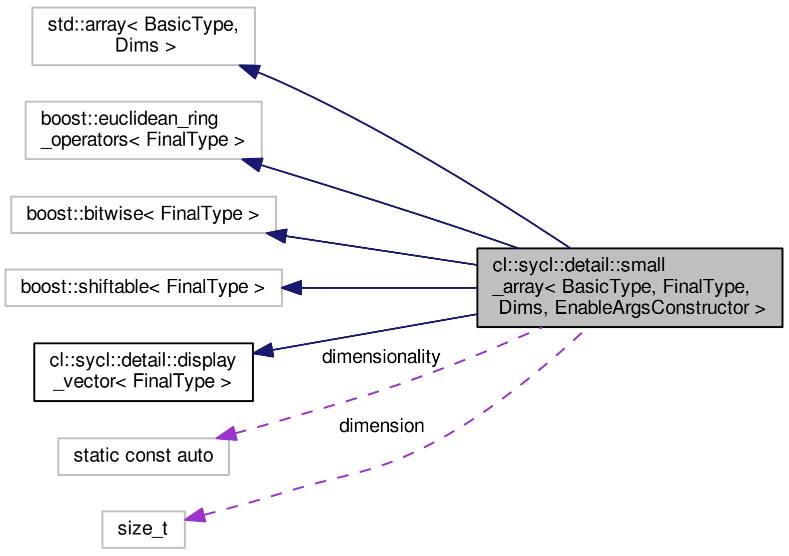
std::array<> provides the collection concept, with .size(), == and != too.

Definition at line 65 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >:


std::array< BasicType, Dims >
boost::euclidean_ring _operators< FinalType >
boost::bitwise< FinalType >
boost::shiftable< FinalType >
cl::sycl::detail::display _vector< FinalType >
cl::sycl::detail::small _array< BasicType, FinalType, Dims, EnableArgsConstructor >

Collaboration diagram for cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >:


std::array< BasicType, Dims >
boost::euclidean_ring _operators< FinalType >
boost::bitwise< FinalType >
boost::shiftable< FinalType >
cl::sycl::detail::display _vector< FinalType >
static const auto
size_t
dimensionality
dimension
cl::sycl::detail::small _array< BasicType, FinalType, Dims, EnableArgsConstructor >

Public Types

- using element_type = BasicType

Public Member Functions

- template<typename SourceType >
  small_array (const SourceType src[Dims])

  A constructor from another array.
- template<typename SourceBasicType , typename SourceFinalType , bool SourceEnableArgsConstructor>
  small_array (const small_array< SourceBasicType, SourceFinalType, Dims, SourceEnableArgsConstructor > &src)

  A constructor from another small_array of the same size.
- template<typename... Types, bool Depend = true, typename = typename std::enable_if_t<EnableArgsConstructor && Depend>>
  small_array (const Types &...args)

  Initialize the array from a list of elements.
- template<typename SourceBasicType >
  small_array (const std::array< SourceBasicType, Dims > &src)

  Construct a small_array from a std::array.
- small_array ()=default

  Keep the synthesized constructors.
- auto get (std::size_t index) const

  Return the element of the array.
- operator FinalType ()

  Add + like operations on the id<> and others.

Static Public Attributes

- static const auto dimensionality = Dims
- static const size_t dimension = Dims

8.7.2.1.1 Member Typedef Documentation

8.7.2.1.1.1 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> using **cl::sycl::detail::small_array**< BasicType, FinalType, Dims, EnableArgsConstructor >**::element_type** = BasicType

Definition at line 85 of file small_array.hpp.

8.7.2.1.2 Constructor & Destructor Documentation

8.7.2.1.2.1 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> template<typename SourceType > **cl::sycl::detail::small_array**< BasicType, FinalType, Dims, EnableArgsConstructor >**::small_array** (const SourceType *src[Dims]*) [inline]

A constructor from another array.

Make it explicit to avoid spurious range<> constructions from int ∗ for example

Definition at line 94 of file small_array.hpp.

```
                                          {
    // (*this)[0] is the first element of the underlying array
    std::copy_n(src, Dims, &(*this)[0]);
  }
```

8.7.2.1.2.2 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> template<typename SourceBasicType , typename SourceFinalType , bool SourceEnableArgsConstructor> **cl::sycl::detail::small_array**< BasicType, FinalType, Dims, EnableArgsConstructor >**::small_array** (const **small_array**< SourceBasicType, SourceFinalType, Dims, SourceEnableArgsConstructor > & *src*) [inline]

A constructor from another small_array of the same size.

Definition at line 104 of file small_array.hpp.

```
00107                                                         {
00108     std::copy_n(&src[0], Dims, &(*this)[0]);
00109   }
```

8.7.2.1.2.3 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> template<typename... Types, bool Depend = true, typename = typename std::enable_if_t<EnableArgsConstructor && Depend>> **cl::sycl::detail::small_array**< BasicType, FinalType, Dims, EnableArgsConstructor >**::small_array** (const Types &... *args*) [inline]

Initialize the array from a list of elements.

Strangely, even when using the array constructors, the initialization of the aggregate is not available. So recreate an equivalent here.

Since there are inherited types that defines some constructors with some conflicts, make it optional here, according to EnableArgsConstructor template parameter.

Definition at line 127 of file small_array.hpp.

```
00128     : std::array<BasicType, Dims> {
00129     // Allow a loss of precision in initialization with the static_cast
00130     { static_cast<BasicType>(args)... }
00131   }
```

8.7.2.1.2.4 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> template<typename SourceBasicType > **cl::sycl::detail::small_array**< BasicType, FinalType, Dims, EnableArgsConstructor >**::small_array** (const std::array< SourceBasicType, Dims > & *src*) [inline]

Construct a small_array from a std::array.

Definition at line 141 of file small_array.hpp.

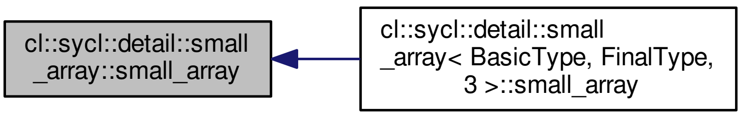
```
00142   : std::array<BasicType, Dims>(src) {}
```

8.7.2.1.2.5 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> **cl::sycl::detail::small_array**< BasicType, FinalType, Dims, EnableArgsConstructor >**::small_array** ( ) [default]

Keep the synthesized constructors.

Referenced by cl::sycl::detail::small_array< BasicType, FinalType, 3 >::small_array().

Here is the caller graph for this function:



8.7.2.1.3 Member Function Documentation

8.7.2.1.3.1 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> auto cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::get (std::size_t *index*) const [inline]

Return the element of the array.

Definition at line 152 of file small_array.hpp.

```
                              {
    return (*this)[index];
  }
```

8.7.2.1.3.2 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::operator FinalType ( ) [inline]

Add + like operations on the id<> and others.

Add - like operations on the id<> and others Add ∗ like operations on the id<> and others Add / like operations on the id<> and others Add % like operations on the id<> and others Add << like operations on the id<> and others Add >> like operations on the id<> and others Add & like operations on the id<> and others Add ^ like operations on the id<> and others Add | like operations on the id<> and others Since the boost::operator work on the small_array, add an implicit conversion to produce the expected type

Definition at line 191 of file small_array.hpp.

```
                         {
    return *static_cast<FinalType *>(this);
  }
```

8.7.2.1.4 Member Data Documentation

8.7.2.1.4.1 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> const size_t cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::dimension = Dims [static]

Definition at line 84 of file small_array.hpp.

8.7.2.1.4.2 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> const auto cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::dimensionality = Dims [static]

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

Definition at line 80 of file small_array.hpp.

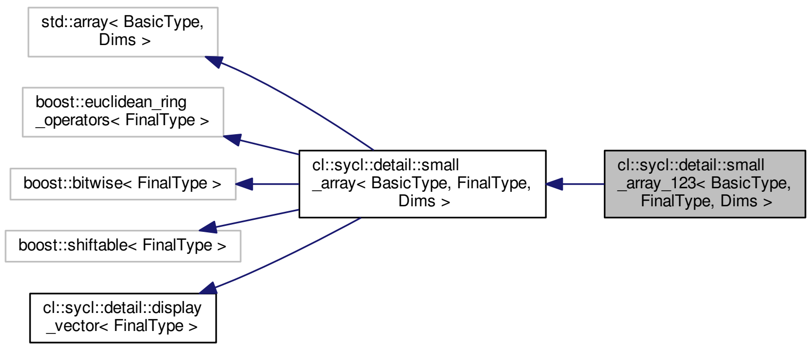
8.7.2.2 struct cl::sycl::detail::small_array_123

template<typename BasicType, typename FinalType, std::size_t Dims>
struct cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >

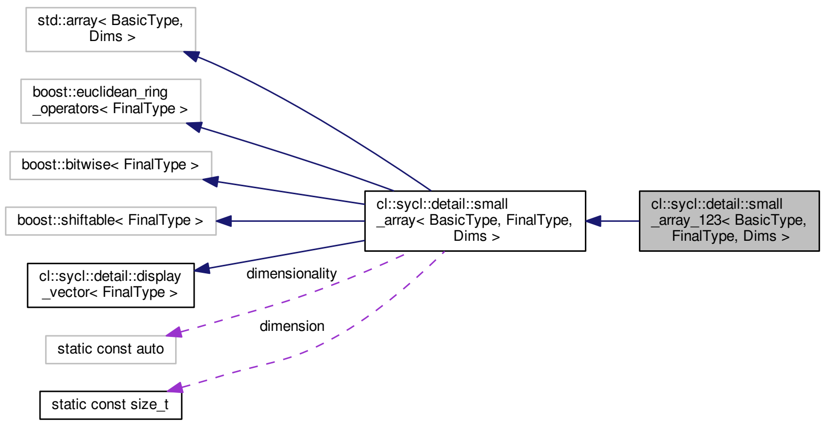
A small array of 1, 2 or 3 elements with the implicit constructors.

Definition at line 200 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >:


std::array< BasicType, Dims >
boost::euclidean_ring _operators< FinalType >
boost::bitwise< FinalType >
boost::shiftable< FinalType >
cl::sycl::detail::display _vector< FinalType >
cl::sycl::detail::small _array< BasicType, FinalType, Dims >
cl::sycl::detail::small _array_123< BasicType, FinalType, Dims >

Collaboration diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >:


std::array< BasicType, Dims >
boost::euclidean_ring _operators< FinalType >
boost::bitwise< FinalType >
boost::shiftable< FinalType >
cl::sycl::detail::display _vector< FinalType >
dimensionality
static const auto
dimension
static const size_t
cl::sycl::detail::small _array< BasicType, FinalType, Dims >
cl::sycl::detail::small _array_123< BasicType, FinalType, Dims >

Additional Inherited Members

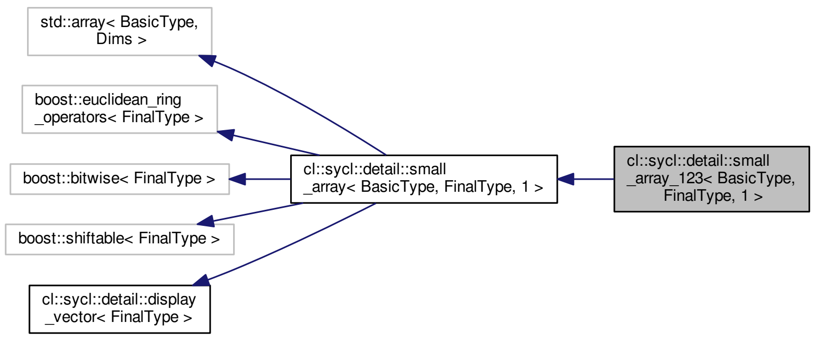
8.7.2.3 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >

template<typename BasicType, typename FinalType>
struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >

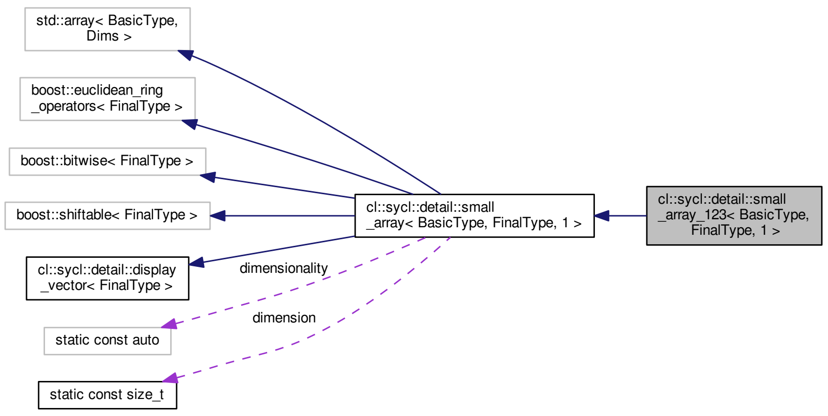
Use some specializations so that some function overloads can be determined according to some implicit constructors and to have an implicit conversion from/to BasicType (such as an int typically) if dims = 1.

Definition at line 212 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >:


std::array< BasicType, Dims >
boost::euclidean_ring _operators< FinalType >
boost::bitwise< FinalType >
boost::shiftable< FinalType >
cl::sycl::detail::display _vector< FinalType >
cl::sycl::detail::small _array< BasicType, FinalType, 1 >
cl::sycl::detail::small _array_123< BasicType, FinalType, 1 >

Collaboration diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >:


std::array< BasicType, Dims >
boost::euclidean_ring _operators< FinalType >
boost::bitwise< FinalType >
boost::shiftable< FinalType >
cl::sycl::detail::display _vector< FinalType >
dimensionality
static const auto
dimension
static const size_t
cl::sycl::detail::small _array< BasicType, FinalType, 1 >
cl::sycl::detail::small _array_123< BasicType, FinalType, 1 >

Public Member Functions

- small_array_123 (BasicType x)

  A 1-D constructor to have implicit conversion from from 1 integer and automatic inference of the dimensionality.
- small_array_123 ()=default

  Keep other constructors.
- operator BasicType () const

  Conversion so that an for example an id<1> can basically be used like an integer.

Additional Inherited Members

8.7.2.3.1 Constructor & Destructor Documentation

8.7.2.3.1.1 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >::small_array_123 (BasicType *x*) [inline]

A 1-D constructor to have implicit conversion from from 1 integer and automatic inference of the dimensionality.

Definition at line 216 of file small_array.hpp.

```
                                {
    (*this)[0] = x;
  }
```

8.7.2.3.1.2 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >::small_array_123 ( ) [default]

Keep other constructors.

8.7.2.3.2 Member Function Documentation

8.7.2.3.2.1 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >::operator BasicType ( ) const [inline]

Conversion so that an for example an id<1> can basically be used like an integer.

Definition at line 228 of file small_array.hpp.

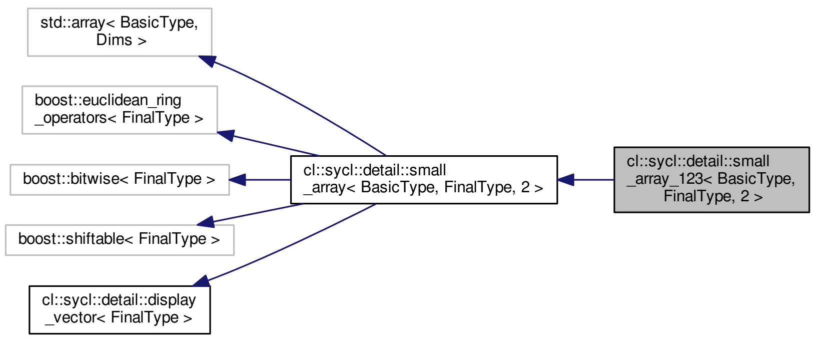
```
                              {
    return (*this)[0];
  }
```

8.7.2.4 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >

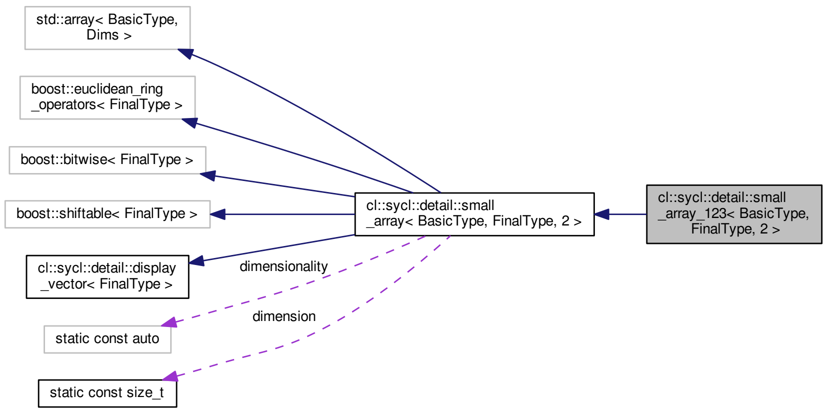
template<typename BasicType, typename FinalType>
struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >

Definition at line 235 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >:


std::array< BasicType, Dims >
boost::euclidean_ring _operators< FinalType >
boost::bitwise< FinalType >
boost::shiftable< FinalType >
cl::sycl::detail::display _vector< FinalType >
cl::sycl::detail::small _array< BasicType, FinalType, 2 >
cl::sycl::detail::small _array_123< BasicType, FinalType, 2 >

Collaboration diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >:


std::array< BasicType, Dims >
boost::euclidean_ring _operators< FinalType >
boost::bitwise< FinalType >
boost::shiftable< FinalType >
cl::sycl::detail::display _vector< FinalType >
static const auto
static const size_t
dimensionality
dimension
cl::sycl::detail::small _array< BasicType, FinalType, 2 >
cl::sycl::detail::small _array_123< BasicType, FinalType, 2 >

Public Member Functions

- small_array_123 (BasicType x, BasicType y)

  A 2-D constructor to have implicit conversion from from 2 integers and automatic inference of the dimensionality.
- small_array_123 (BasicType e)

  Broadcasting constructor initializing all the elements with the same value.
- small_array_123 ()=default

  Keep other constructors.

Additional Inherited Members

8.7.2.4.1 Constructor & Destructor Documentation

8.7.2.4.1.1 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >::small_array_123 (BasicType *x*, BasicType *y*) [inline]

A 2-D constructor to have implicit conversion from from 2 integers and automatic inference of the dimensionality.

Definition at line 239 of file small_array.hpp.

```
00239                                          {
00240     (*this)[0] = x;
00241     (*this)[1] = y;
00242   }
```

8.7.2.4.1.2 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >::small_array_123 (BasicType *e*) [inline], [explicit]

Broadcasting constructor initializing all the elements with the same value.

Todo Add to the specification of the range, id...

Definition at line 250 of file small_array.hpp.

```
00250 : small_array_123 { e, e } { }
```

8.7.2.4.1.3 template< typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >::small_array_123 ( ) [default]

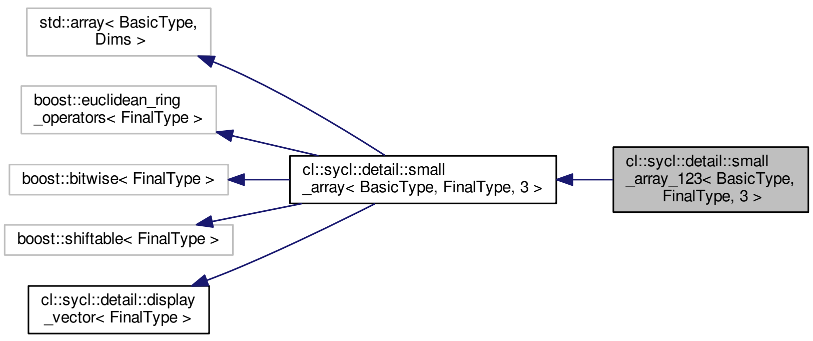
Keep other constructors.

8.7.2.5 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >

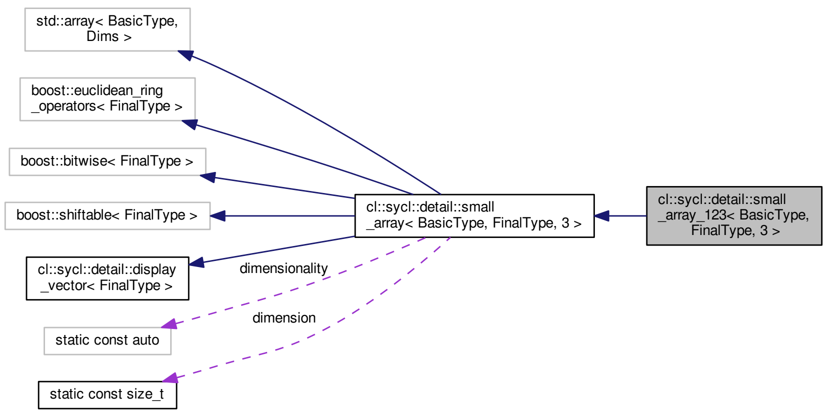
template<typename BasicType, typename FinalType>
struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >

Definition at line 261 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >:


std::array< BasicType, Dims >
boost::euclidean_ring _operators< FinalType >
boost::bitwise< FinalType >
boost::shiftable< FinalType >
cl::sycl::detail::display _vector< FinalType >
cl::sycl::detail::small _array< BasicType, FinalType, 3 >
cl::sycl::detail::small _array_123< BasicType, FinalType, 3 >

Collaboration diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >:


std::array< BasicType, Dims >
boost::euclidean_ring _operators< FinalType >
boost::bitwise< FinalType >
boost::shiftable< FinalType >
cl::sycl::detail::display _vector< FinalType >
static const auto
static const size_t
dimensionality
dimension
cl::sycl::detail::small _array< BasicType, FinalType, 3 >
cl::sycl::detail::small _array_123< BasicType, FinalType, 3 >

Public Member Functions

- small_array_123 (BasicType x, BasicType y, BasicType z)

  A 3-D constructor to have implicit conversion from from 3 integers and automatic inference of the dimensionality.
- small_array_123 (BasicType e)

  Broadcasting constructor initializing all the elements with the same value.
- small_array_123 ()=default

  Keep other constructors.

Additional Inherited Members

8.7.2.5.1 Constructor & Destructor Documentation

8.7.2.5.1.1 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >::small_array_123 (BasicType *x*, BasicType *y*, BasicType *z*) [inline]

A 3-D constructor to have implicit conversion from from 3 integers and automatic inference of the dimensionality.

Definition at line 265 of file small_array.hpp.

```
00265                                                          {
00266     (*this)[0] = x;
00267     (*this)[1] = y;
00268     (*this)[2] = z;
00269   }
```

8.7.2.5.1.2 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >::small_array_123 (BasicType *e*) [inline], [explicit]

Broadcasting constructor initializing all the elements with the same value.

Todo Add to the specification of the range, id...

Definition at line 277 of file small_array.hpp.

```
00277 : small_array_123 { e, e, e } { }
```

8.7.2.5.1.3 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >::small_array_123 ( ) [default]

Keep other constructors.

8.7.3 Macro Definition Documentation

8.7.3.1 #define TRISYCL_BOOST_OPERATOR_VECTOR_OP(*op*)

```
#include <include/CL/sycl/detail/small_array.hpp>
```

Value:

```
FinalType operator op(const FinalType &rhs) {          \
    for (std::size_t i = 0; i != Dims; ++i)              \
      (*this)[i] op rhs[i];                              \
    return *this;                                        \
  }
```

Helper macro to declare a vector operation with the given side-effect operator.

Definition at line 33 of file small_array.hpp.

Referenced by cl::sycl::detail::small_array< BasicType, FinalType, 3 >::get().

8.7.4 Function Documentation

8.7.4.1 template<typename Range , typename Id > size_t constexpr cl::sycl::detail::linear_id (Range *range,* Id *id,* Id *offset =* {}) [inline]

```
#include <include/CL/sycl/detail/linear_id.hpp>
```

Compute a linearized array access used in the OpenCL 2 world.

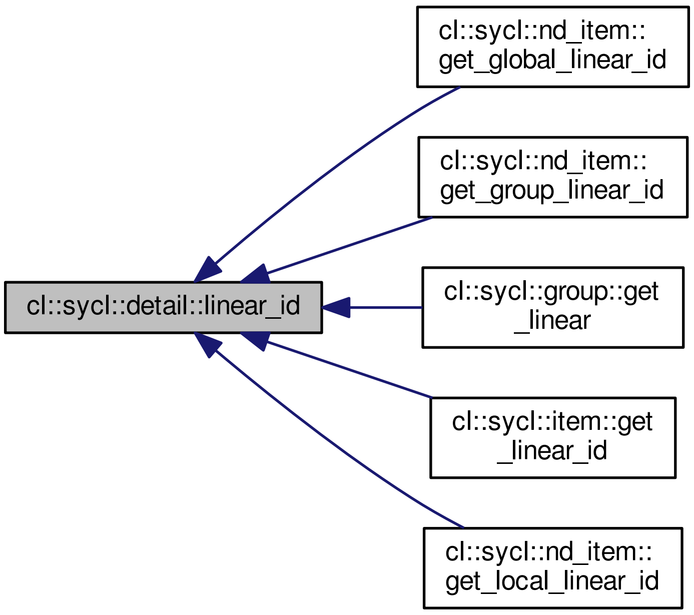
Typically for the get_global_linear_id() and get_local_linear_id() functions.

Definition at line 28 of file linear_id.hpp.

Referenced by cl::sycl::nd_item< dims >::get_global_linear_id(), cl::sycl::nd_item< dims >::get_group_linear_id(), cl::sycl::group< dims >::get_linear(), cl::sycl::item< dims >::get_linear_id(), and cl::sycl::nd_item< dims >::get←_local_linear_id().

```
                                                                  {}) {
  auto dims = std::distance(std::begin(range), std::end(range));

  size_t linear_id = 0;
  /* A good compiler should unroll this and do partial evaluation to
     remove the first multiplication by 0 of this Horner evaluation and
     remove the 0 offset evaluation */
    for (int i = dims - 1; i >= 0; --i)
      linear_id = linear_id*range[i] + id[i] - offset[i];

    return linear_id;
  }
```

Here is the caller graph for this function:



8.7.4.2 void cl::sycl::detail::unimplemented ( ) [inline]

#include <include/CL/sycl/detail/unimplemented.hpp>

Display an "unimplemented" message.

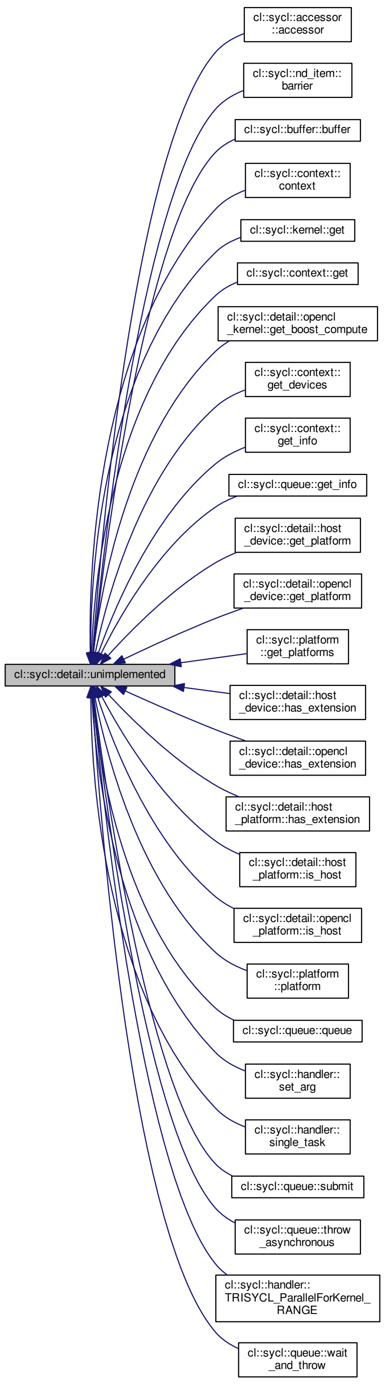
Can be changed to call assert(0) or whatever.

Definition at line 25 of file unimplemented.hpp.

Referenced by cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor(), cl::sycl::nd_item< dims >::barrier(), cl::sycl::buffer< T, Dimensions, Allocator >::buffer(), cl::sycl::context::context(), cl::sycl::kernel::get(), cl::sycl::context::get(), cl::sycl::detail::opencl_kernel::get_boost_compute(), cl::sycl::context::get_devices(), cl::sycl::context::get_info(), cl::sycl::queue::get_info(), cl::sycl::detail::host_device::get_platform(), cl::sycl::detail::opencl_device::get_platform(), cl::sycl::platform::get_platforms(), cl::sycl::detail::host_device::has_extension(), cl::sycl::detail::opencl_device::has_extension(), cl::sycl::detail::host_platform::has_extension(), cl::sycl::detail::host_platform::is_host(), cl::sycl::detail::opencl_platform::is_host(), cl::sycl::platform::platform(), cl::sycl::queue::queue(), cl::sycl::handler::set_arg(), cl::sycl::handler::single_task(), cl::sycl::queue::submit(), cl::sycl::queue::throw_asynchronous(), cl::sycl::handler::TRISYCL_ParallelForKernel_RANGE(), and cl::sycl::queue::wait_and_throw().

```
                          {
  std::cerr << "Error: using a non implemented feature!!!" << std::endl
            << "Please contribute to the open source implementation. :-)"
            << std::endl;
}
```

Here is the caller graph for this function:


cl::sycl::detail::unimplemented
cl::sycl::accessor ::accessor
cl::sycl::nd_item:: barrier
cl::sycl::buffer::buffer
cl::sycl::context:: context
cl::sycl::kernel::get
cl::sycl::context::get
cl::sycl::detail::opencl _kernel::get_boost_compute
cl::sycl::context:: get_devices
cl::sycl::context:: get_info
cl::sycl::queue::get_info
cl::sycl::detail::host _device::get_platform
cl::sycl::detail::opencl _device::get_platform
cl::sycl::platform ::get_platforms
cl::sycl::detail::host _device::has_extension
cl::sycl::detail::opencl _device::has_extension
cl::sycl::detail::host _platform::has_extension
cl::sycl::detail::host _platform::is_host
cl::sycl::detail::opencl _platform::is_host
cl::sycl::platform ::platform
cl::sycl::queue::queue
cl::sycl::handler:: set_arg
cl::sycl::handler:: single_task
cl::sycl::queue::submit
cl::sycl::queue::throw _asynchronous
cl::sycl::handler:: TRISYCL_ParallelForKernel_ RANGE
cl::sycl::queue::wait _and_throw

8.8 Error handling

Namespaces

- cl::sycl::trisycl

Classes

- 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::exception_list

  Exception list to store several exceptions. More...
- class cl::sycl::exception

  Encapsulate a SYCL error information. More...
- class cl::sycl::cl_exception

  Returns the OpenCL error code encapsulated in the exception. More...
- struct cl::sycl::async_exception

  An error stored in an exception_list for asynchronous errors. More...
- class cl::sycl::runtime_error
- class cl::sycl::kernel_error

  Error that occurred before or while enqueuing the SYCL kernel. More...
- class cl::sycl::accessor_error

  Error regarding the cl::sycl::accessor objects defined. More...
- class cl::sycl::nd_range_error

  Error regarding the cl::sycl::nd_range specified for the SYCL kernel. More...
- class cl::sycl::event_error

  Error regarding associated cl::sycl::event objects. More...
- class cl::sycl::invalid_parameter_error

  Error regarding parameters to the SYCL kernel, it may apply to any captured parameters to the kernel lambda. More...
- class cl::sycl::device_error

  The SYCL device will trigger this exception on error. More...
- class cl::sycl::compile_program_error

  Error while compiling the SYCL kernel to a SYCL device. More...
- class cl::sycl::link_program_error

  Error while linking the SYCL kernel to a SYCL device. More...
- class cl::sycl::invalid_object_error

  Error regarding any memory objects being used inside the kernel. More...
- class cl::sycl::memory_allocation_error

  Error on memory allocation on the SYCL device for a SYCL kernel. More...
- class cl::sycl::pipe_error

  A failing pipe error will trigger this exception on error. More...
- class cl::sycl::platform_error

  The SYCL platform will trigger this exception on error. More...
- class cl::sycl::profiling_error

  The SYCL runtime will trigger this error if there is an error when profiling info is enabled. More...
- class cl::sycl::feature_not_supported

  Exception thrown when an optional feature or extension is used in a kernel but its not available on the device the SYCL kernel is being enqueued on. More...
- class cl::sycl::non_cl_error

  Exception for an OpenCL operation requested in a non OpenCL area. More...

Typedefs

- using cl::sycl::exception_ptr = std::exception_ptr

    A shared pointer to an exception as in C++ specification.
- using cl::sycl::async_handler = function_class< void, exception_list >

8.8.1 Detailed Description

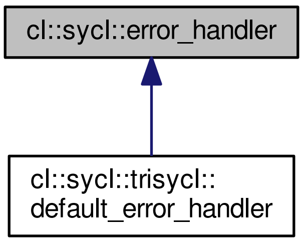
8.8.2 Class Documentation

8.8.2.1 struct cl::sycl::error_handler

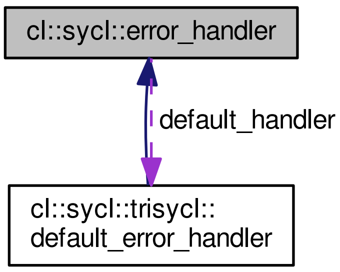
User supplied error handler to call a user-provided function when an error happens from a SYCL object that was constructed with this error handler.

Definition at line 32 of file error_handler.hpp.

Inheritance diagram for cl::sycl::error_handler:



Collaboration 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.

8.8.2.1.1 Member Function Documentation

8.8.2.1.1.1 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.

8.8.2.1.2 Member Data Documentation

8.8.2.1.2.1 **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?

Definition at line 43 of file error_handler.hpp.

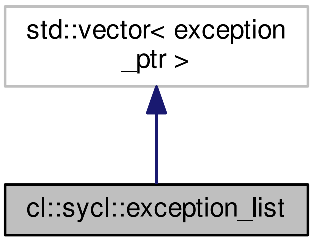
8.8.2.2 struct cl::sycl::exception_list

Exception list to store several exceptions.

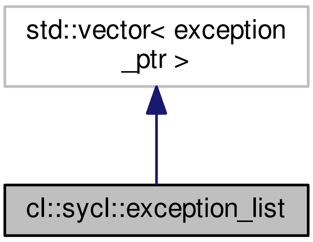
Todo Do we need to define it in SYCL or can we rely on plain C++17 one?

Definition at line 33 of file exception.hpp.

Inheritance diagram for cl::sycl::exception_list:



Collaboration diagram for cl::sycl::exception_list:

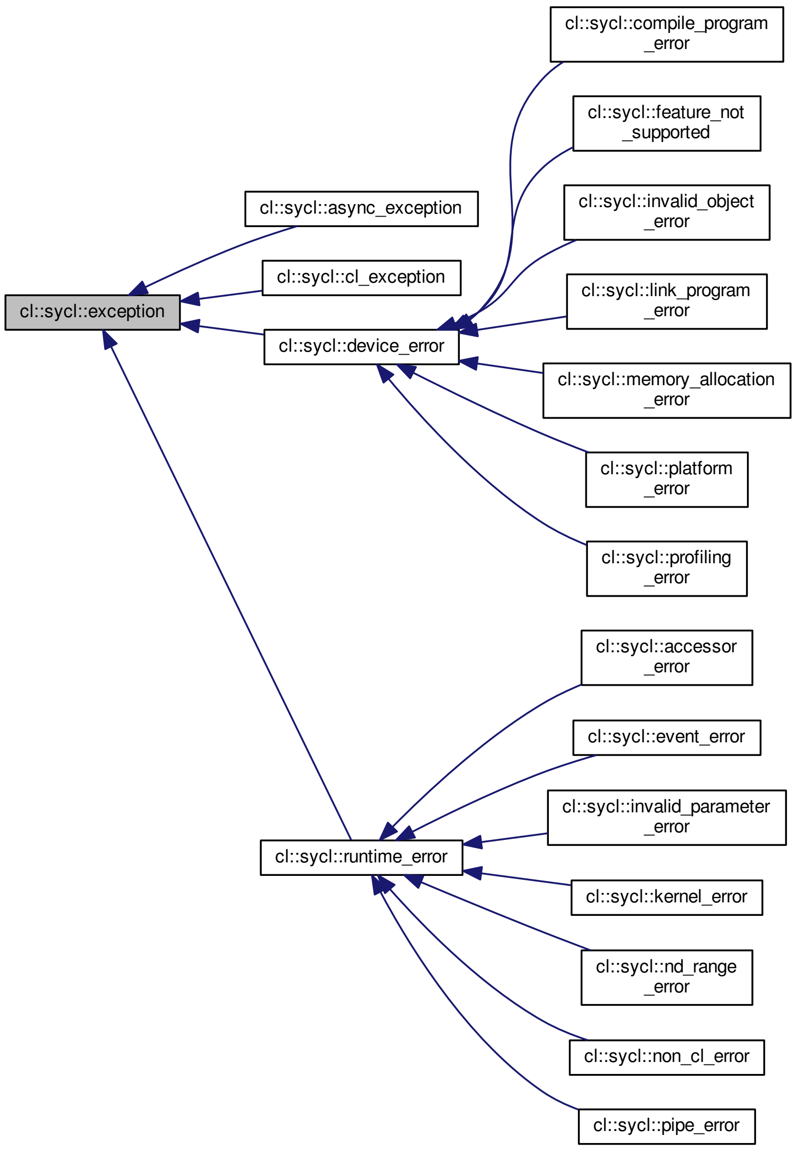


8.8.2.3 class cl::sycl::exception

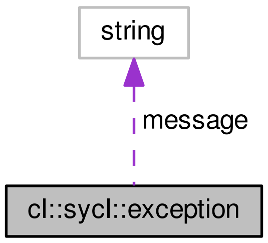
Encapsulate a SYCL error information.

Definition at line 41 of file exception.hpp.

Inheritance diagram for cl::sycl::exception:



Collaboration diagram for cl::sycl::exception:



Public Member Functions

- exception (const string_class &message)

  Construct an exception with a message for internal use.
- string_class what () const

  Returns a descriptive string for the error, if available.

Private Attributes

- string_class message

  The error message to return.

8.8.2.3.1 Constructor & Destructor Documentation

8.8.2.3.1.1 cl::sycl::exception::exception (const **string_class** & *message*) [inline]

Construct an exception with a message for internal use.

Definition at line 49 of file exception.hpp.

```
00049 : message { message } {}
```

8.8.2.3.2 Member Function Documentation

8.8.2.3.2.1 **string_class** cl::sycl::exception::what ( ) const [inline]

Returns a descriptive string for the error, if available.

Definition at line 52 of file exception.hpp.

```
                           {
      return message;
    }
```

8.8.2.3.3 Member Data Documentation

8.8.2.3.3.1 string_class cl::sycl::exception::message [private]

The error message to return.

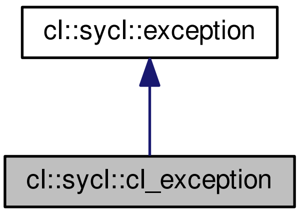
Definition at line 44 of file exception.hpp.

8.8.2.4 class cl::sycl::cl_exception

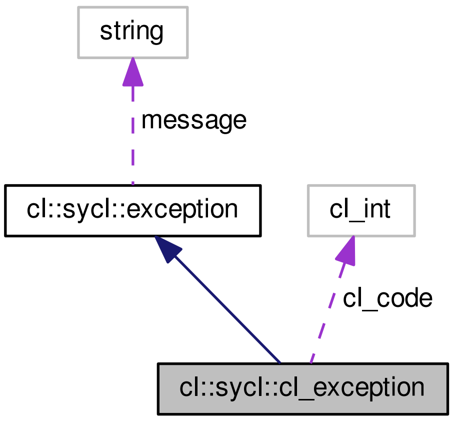
Returns the OpenCL error code encapsulated in the exception.

Definition at line 69 of file exception.hpp.

Inheritance diagram for cl::sycl::cl_exception:



Collaboration diagram for cl::sycl::cl_exception:



Public Member Functions

- cl_exception (const string_class &message, cl_int cl_code)

  Construct an exception with a message and OpenCL error code for internal use.
- cl_int get_cl_code () const

Private Attributes

- cl_int cl_code

  The OpenCL error code to return.

8.8.2.4.1 Constructor & Destructor Documentation

8.8.2.4.1.1 cl::sycl::cl_exception::cl_exception (const string_class & *message,* cl_int *cl_code*) [inline]

Construct an exception with a message and OpenCL error code for internal use.

Definition at line 80 of file exception.hpp.

```
00081     : exception { message }, cl_code { cl_code } {}
```

8.8.2.4.2 Member Function Documentation

8.8.2.4.2.1 cl_int cl::sycl::cl_exception::get_cl_code ( ) const [inline]

Definition at line 84 of file exception.hpp.

```
00084                                 {
00085     return cl_code;
00086   }
```

8.8.2.4.3 Member Data Documentation

8.8.2.4.3.1 cl_int cl::sycl::cl_exception::cl_code [private]

The OpenCL error code to return.

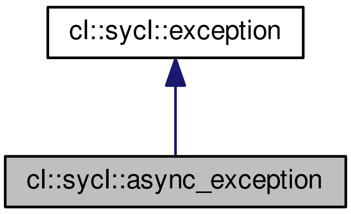
Definition at line 74 of file exception.hpp.

8.8.2.5 struct cl::sycl::async_exception

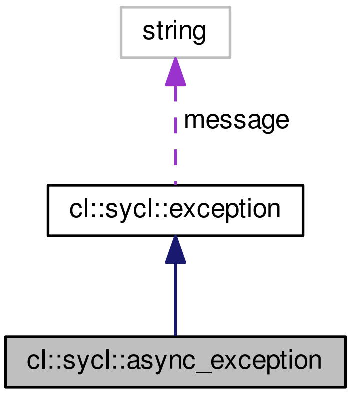
An error stored in an exception_list for asynchronous errors.

Definition at line 93 of file exception.hpp.

Inheritance diagram for cl::sycl::async_exception:



Collaboration diagram for cl::sycl::async_exception:

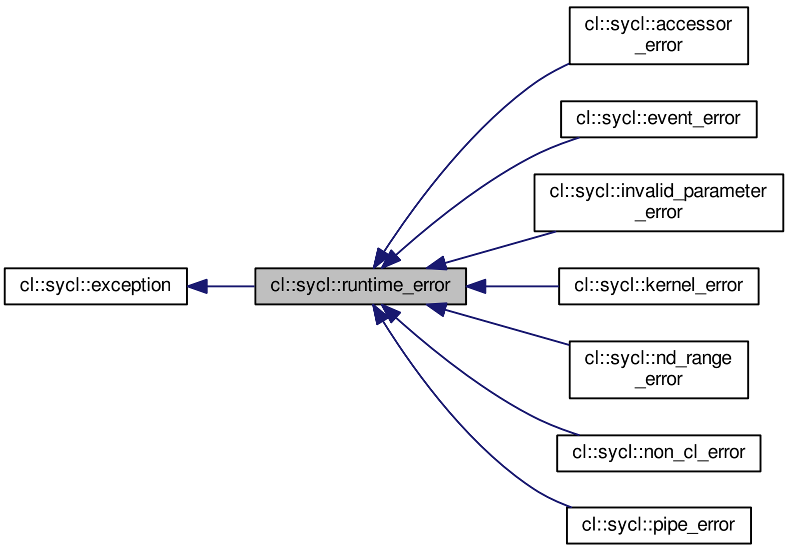


Additional Inherited Members

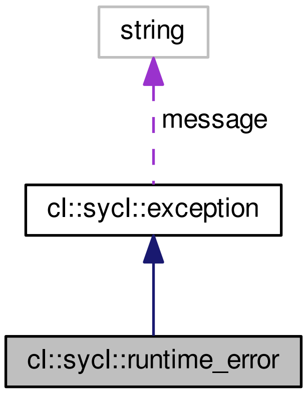
8.8.2.6 class cl::sycl::runtime_error

Definition at line 98 of file exception.hpp.

Inheritance diagram for cl::sycl::runtime_error:



Collaboration diagram for cl::sycl::runtime_error:



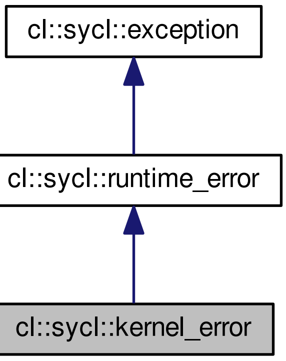
Additional Inherited Members

8.8.2.7 class cl::sycl::kernel_error

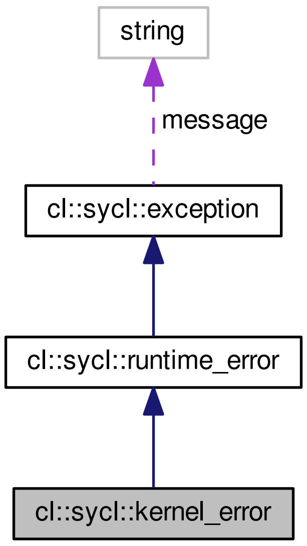
Error that occurred before or while enqueuing the SYCL kernel.

Definition at line 104 of file exception.hpp.

Inheritance diagram for cl::sycl::kernel_error:



Collaboration diagram for cl::sycl::kernel_error:



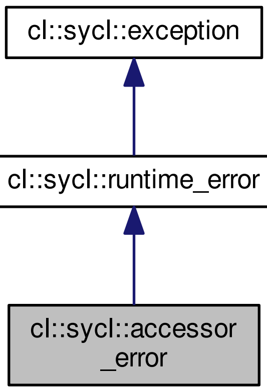
Additional Inherited Members

8.8.2.8 class cl::sycl::accessor_error

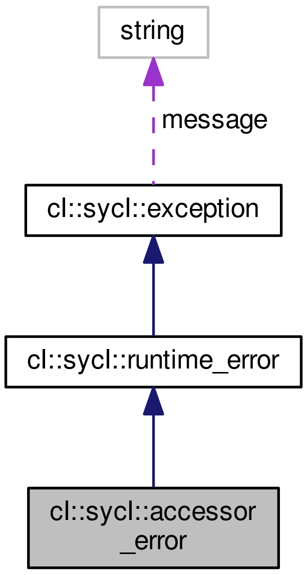
Error regarding the cl::sycl::accessor objects defined.

Definition at line 110 of file exception.hpp.

Inheritance diagram for cl::sycl::accessor_error:



Collaboration diagram for cl::sycl::accessor_error:



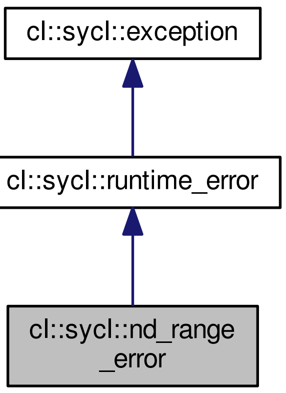
Additional Inherited Members

8.8.2.9 class cl::sycl::nd_range_error

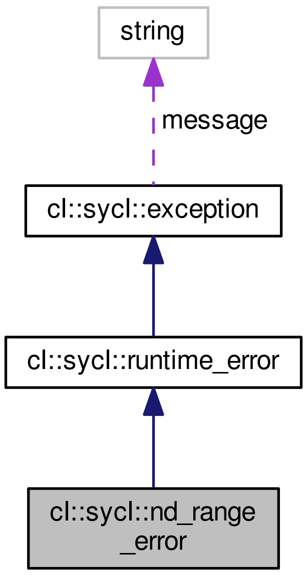
Error regarding the cl::sycl::nd_range specified for the SYCL kernel.

Definition at line 116 of file exception.hpp.

Inheritance diagram for cl::sycl::nd_range_error:



Collaboration diagram for cl::sycl::nd_range_error:



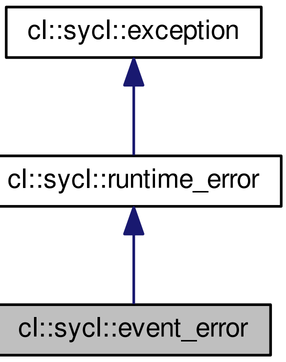
Additional Inherited Members

8.8.2.10 class cl::sycl::event_error

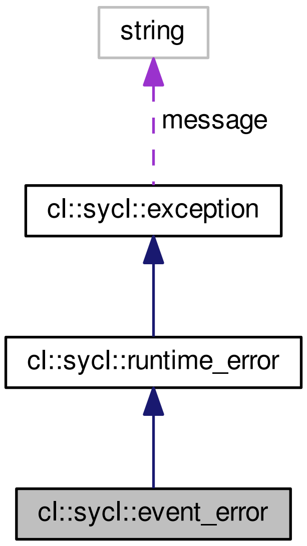
Error regarding associated cl::sycl::event objects.

Definition at line 122 of file exception.hpp.

Inheritance diagram for cl::sycl::event_error:



Collaboration diagram for cl::sycl::event_error:



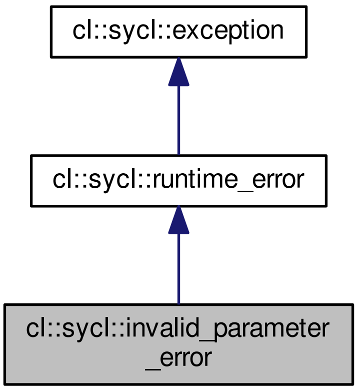
Additional Inherited Members

8.8.2.11 class cl::sycl::invalid_parameter_error

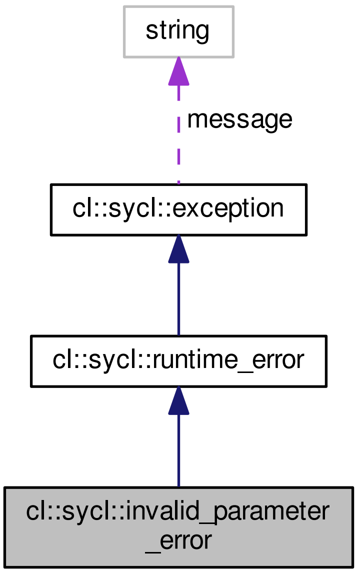
Error regarding parameters to the SYCL kernel, it may apply to any captured parameters to the kernel lambda.

Definition at line 130 of file exception.hpp.

Inheritance diagram for cl::sycl::invalid_parameter_error:



Collaboration diagram for cl::sycl::invalid_parameter_error:



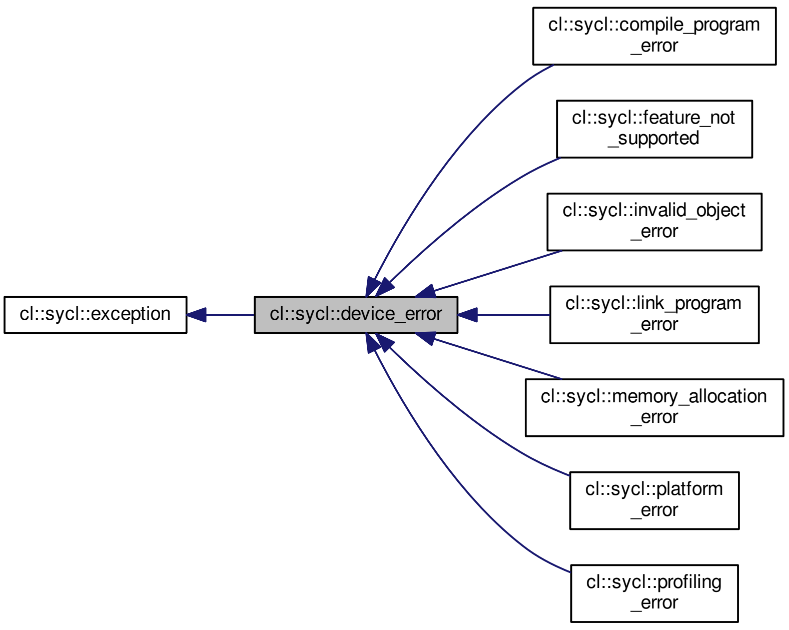
Additional Inherited Members

8.8.2.12 class cl::sycl::device_error

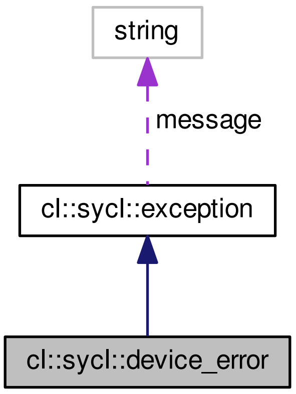
The SYCL device will trigger this exception on error.

Definition at line 136 of file exception.hpp.

Inheritance diagram for cl::sycl::device_error:


cl::sycl::compile_program_error
cl::sycl::feature_not_supported
cl::sycl::invalid_object_error
cl::sycl::exception
cl::sycl::device_error
cl::sycl::link_program_error
cl::sycl::memory_allocation_error
cl::sycl::platform_error
cl::sycl::profiling_error

Collaboration diagram for cl::sycl::device_error:



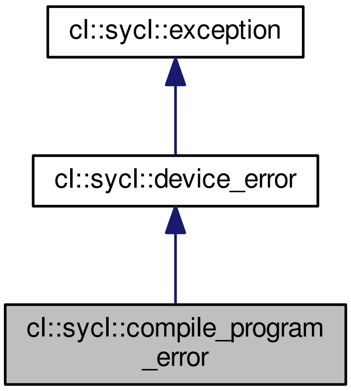
Additional Inherited Members

8.8.2.13 class cl::sycl::compile_program_error

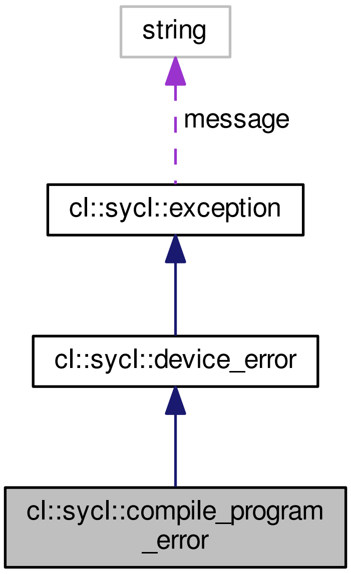
Error while compiling the SYCL kernel to a SYCL device.

Definition at line 142 of file exception.hpp.

Inheritance diagram for cl::sycl::compile_program_error:



Collaboration diagram for cl::sycl::compile_program_error:



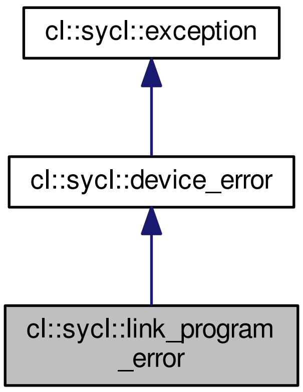
Additional Inherited Members

8.8.2.14 class cl::sycl::link_program_error

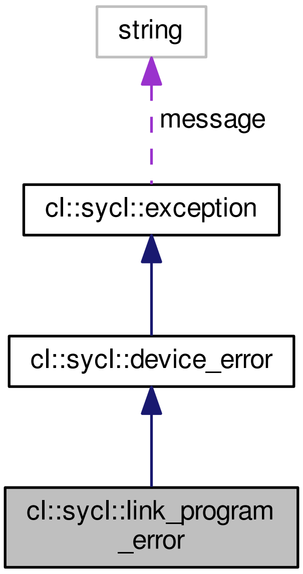
Error while linking the SYCL kernel to a SYCL device.

Definition at line 148 of file exception.hpp.

Inheritance diagram for cl::sycl::link_program_error:



Collaboration diagram for cl::sycl::link_program_error:



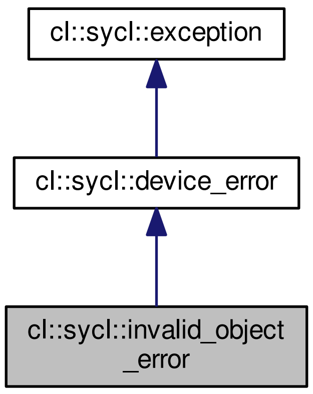
Additional Inherited Members

8.8.2.15 class cl::sycl::invalid_object_error

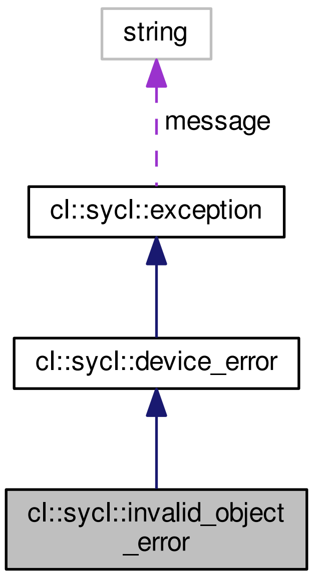
Error regarding any memory objects being used inside the kernel.

Definition at line 154 of file exception.hpp.

Inheritance diagram for cl::sycl::invalid_object_error:



Collaboration diagram for cl::sycl::invalid_object_error:



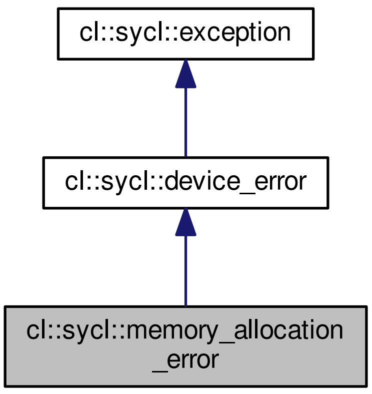
Additional Inherited Members

8.8.2.16 class cl::sycl::memory_allocation_error

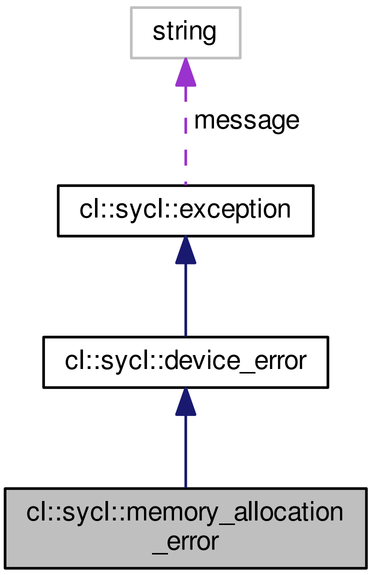
Error on memory allocation on the SYCL device for a SYCL kernel.

Definition at line 160 of file exception.hpp.

Inheritance diagram for cl::sycl::memory_allocation_error:



Collaboration diagram for cl::sycl::memory_allocation_error:



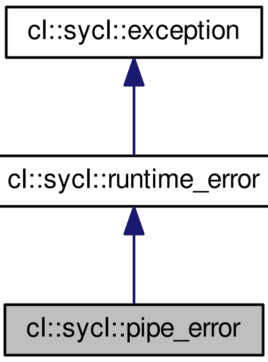
Additional Inherited Members

8.8.2.17 class cl::sycl::pipe_error

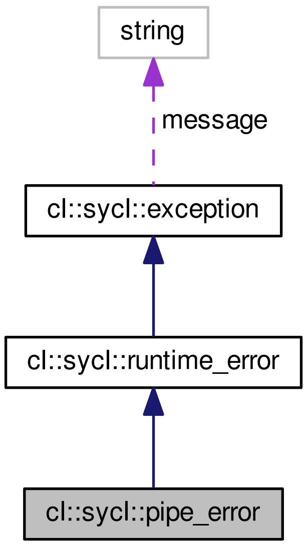
A failing pipe error will trigger this exception on error.

Definition at line 166 of file exception.hpp.

Inheritance diagram for cl::sycl::pipe_error:



Collaboration diagram for cl::sycl::pipe_error:



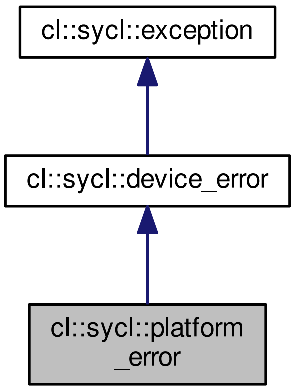
Additional Inherited Members

8.8.2.18 class cl::sycl::platform_error

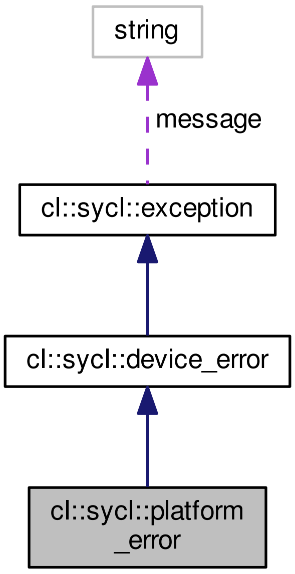
The SYCL platform will trigger this exception on error.

Definition at line 172 of file exception.hpp.

Inheritance diagram for cl::sycl::platform_error:



Collaboration diagram for cl::sycl::platform_error:



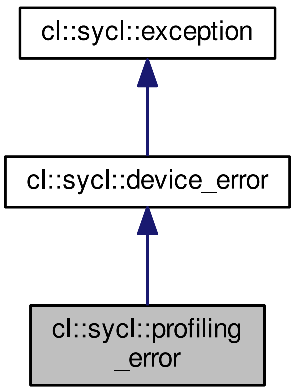
Additional Inherited Members

8.8.2.19 class cl::sycl::profiling_error

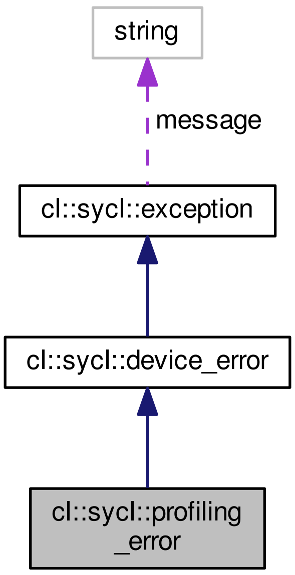
The SYCL runtime will trigger this error if there is an error when profiling info is enabled.

Definition at line 180 of file exception.hpp.

Inheritance diagram for cl::sycl::profiling_error:



Collaboration diagram for cl::sycl::profiling_error:



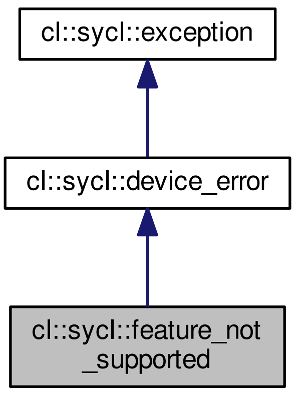
Additional Inherited Members

8.8.2.20 class cl::sycl::feature_not_supported

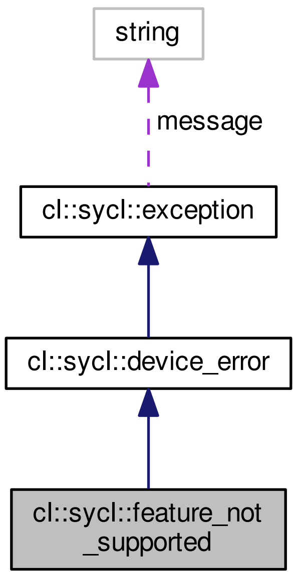
Exception thrown when an optional feature or extension is used in a kernel but its not available on the device the SYCL kernel is being enqueued on.

Definition at line 189 of file exception.hpp.

Inheritance diagram for cl::sycl::feature_not_supported:



Collaboration diagram for cl::sycl::feature_not_supported:



Additional Inherited Members

8.8.2.21 class cl::sycl::non_cl_error

Exception for an OpenCL operation requested in a non OpenCL area.

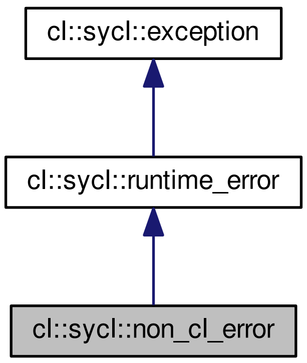
Todo Add to the specification

Todo Clean implementation

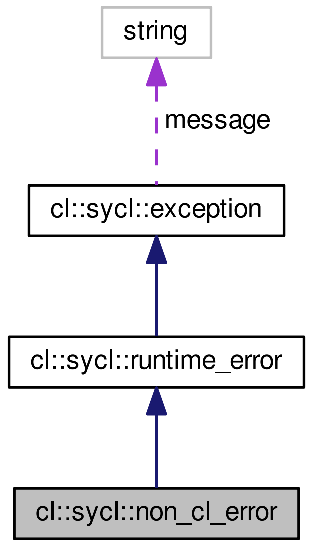
Todo Exceptions are named error in C++

Definition at line 202 of file exception.hpp.

Inheritance diagram for cl::sycl::non_cl_error:



Collaboration diagram for cl::sycl::non_cl_error:



Additional Inherited Members

8.8.3 Typedef Documentation

8.8.3.1 using **cl::sycl::async_handler** = typedef function_class<void, exception_list>

`#include <include/CL/sycl/exception.hpp>`

Definition at line 37 of file exception.hpp.

8.8.3.2 using **cl::sycl::exception_ptr** = typedef std::exception_ptr

`#include <include/CL/sycl/exception.hpp>`

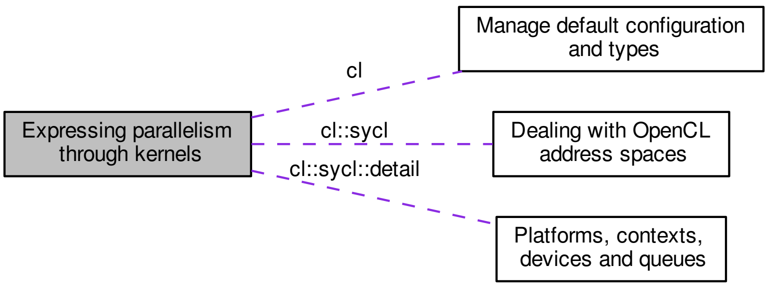
A shared pointer to an exception as in C++ specification.

Todo Do we need this instead of reusing directly the one from C++11?

Definition at line 26 of file exception.hpp.

8.9 Expressing parallelism through kernels

Collaboration diagram for Expressing parallelism through kernels:


Manage default configuration and types
cl
Expressing parallelism through kernels
cl::sycl
Dealing with OpenCL address spaces
cl::sycl::detail
Platforms, contexts, devices and queues

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Classes

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

  A group index used in a parallel_for_workitem to specify a work_group. More...
- class cl::sycl::id< dims >

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

  A SYCL item stores information on a work-item with some more context such as the definition range and offset. More...
- struct cl::sycl::nd_item< dims >

  A SYCL nd_item stores information on a work-item within a work-group, with some more context such as the definition ranges. 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::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id >

  A recursive multi-dimensional iterator that ends up calling f. More...
- struct cl::sycl::detail::parallel_OpenMP_for_iterate< level, Range, ParallelForFunctor, Id >

  A top-level recursive multi-dimensional iterator variant using OpenMP. More...
- struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >

  Stop the recursion when level reaches 0 by simply calling the kernel functor with the constructed id. More...
- class cl::sycl::range< dims >

  A SYCL range defines a multi-dimensional index range that can be used to define launch parallel computation extent or buffer sizes. More...

Functions

- auto cl::sycl::make_id (id< 1 > i)

  Implement a make_id to construct an id<> of the right dimension with implicit conversion from an initializer list for example.
- auto cl::sycl::make_id (id< 2 > i)
- auto cl::sycl::make_id (id< 3 > i)
- template<typename... BasicType>
  auto cl::sycl::make_id (BasicType...Args)

  Construct an id<> from a function call with arguments, like make_id(1, 2, 3)
- template<std::size_t Dimensions = 1, typename ParallelForFunctor , typename Id >
  void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFunctor f, Id)

  Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFunctor f, item< Dimensions >)

  Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFunctor f)

  Calls the appropriate ternary parallel_for overload based on the index type of the kernel function object f.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_global_offset (range< Dimensions > global_size, id< Dimensions > offset, ParallelForFunctor f)

  Implementation of parallel_for with a range<> and an offset.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement a variation of parallel_for to take into account a nd_range<>
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement the loop on the work-groups.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_workitem (const group< Dimensions > &g, ParallelForFunctor f)

  Implement the loop on the work-items inside a work-group.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_work_item (const group< Dimensions > &g, ParallelForFunctor f)

  SYCL parallel_for version that allows a Program object to be specified.
- auto cl::sycl::make_range (range< 1 > r)

  Implement a make_range to construct a range<> of the right dimension with implicit conversion from an initializer list for example.
- auto cl::sycl::make_range (range< 2 > r)
- auto cl::sycl::make_range (range< 3 > r)
- template<typename... BasicType>
  auto cl::sycl::make_range (BasicType...Args)

  Construct a range<> from a function call with arguments, like make_range(1, 2, 3)

8.9.1 Detailed Description

8.9.2 Class Documentation

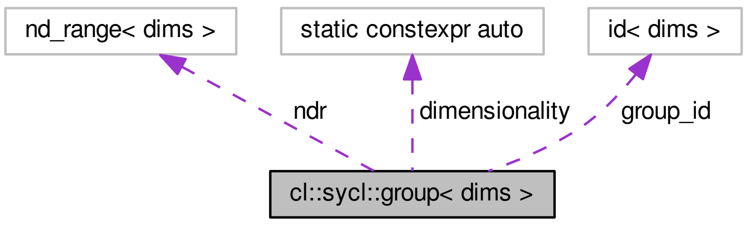
8.9.2.1 struct cl::sycl::group

template<std::size_t dims>
struct cl::sycl::group< dims >

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

Definition at line 24 of file group.hpp.

Collaboration diagram for cl::sycl::group< dims >:



Public Member Functions

- group (const nd_range< dims > &ndr)

  Create a group from an nd_range<> with a 0 id<>
- group (const id< dims > &i, const nd_range< dims > &ndr)

  Create a group from an id and a nd_range<>
- group ()=default

  To be able to copy and assign group, use default constructors too.
- id< dims > get () const

  Return an id representing the index of the group within the nd_range for every dimension.
- size_t get (int dimension) const

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

  Return the index of the group in the given dimension within the nd_range<>
- range< dims > get_group_range () const

  Return a range<> representing the dimensions of the current group.
- size_t get_group_range (int dimension) const

  Return element dimension from the con stituent group range.
- range< dims > get_global_range () const

  Get the local range for this work_group.
- size_t get_global_range (int dimension) const

  Return element dimension from the constituent global range.
- range< dims > get_local_range () const

  Get the local range for this work_group.
- size_t get_local_range (int dimension) const

  Return element dimension from the constituent local range.
- id< dims > get_offset () const

  Get the offset of the NDRange.
- size_t get_offset (int dimension) const

Get the offset of the NDRange.
- nd_range< dims > get_nd_range () const
- size_t get_linear () const

  Get a linearized version of the group ID.
- void parallel_for_work_item (std::function< void(nd_item< dimensionality >)> f) const

  Loop on the work-items inside a work-group.
- void parallel_for_work_item (std::function< void(item< dimensionality >)> f) const

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

Static Public Attributes

- static constexpr auto dimensionality = dims

Private Attributes

- id< dims > group_id

  The coordinate of the group item.
- nd_range< dims > ndr

  Keep a reference on the nd_range to serve potential query on it.

8.9.2.1.1 Constructor & Destructor Documentation

8.9.2.1.1.1 template<std::size_t dims> cl::sycl::group< dims >::group (const nd_range< dims > & *ndr*) [inline]

Create a group from an nd_range<> with a 0 id<>

Todo This should be private since it is only used by the triSYCL implementation

Definition at line 61 of file group.hpp.

```
00061 : ndr { ndr } {}
```

8.9.2.1.1.2 template<std::size_t dims> cl::sycl::group< dims >::group (const id< dims > & *i*, const nd_range< dims > & *ndr*) [inline]

Create a group from an id and a nd_range<>

Todo This should be private somehow, but it is used by the validation infrastructure

Definition at line 69 of file group.hpp.

```
00069                                                         :
00070     group_id { i }, ndr { ndr } {}
```

8.9.2.1.1.3 template<std::size_t dims> cl::sycl::group< dims >::group ( ) [default]

To be able to copy and assign group, use default constructors too.

Todo Make most of them protected, reserved to implementation

8.9.2.1.2 Member Function Documentation

8.9.2.1.2.1 template<std::size_t dims> id<dims> cl::sycl::group< dims >::get ( ) const [inline]

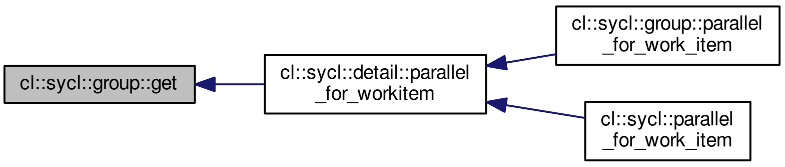
Return an id representing the index of the group within the nd_range for every dimension.

Definition at line 83 of file group.hpp.

Referenced by cl::sycl::detail::parallel_for_workitem().

```
00083 { return group_id; }
```

Here is the caller graph for this function:


cl::sycl::group::get
cl::sycl::detail::parallel_for_workitem
cl::sycl::group::parallel_for_work_item
cl::sycl::parallel_for_work_item

8.9.2.1.2.2 template<std::size_t dims> size_t cl::sycl::group< dims >::get (int *dimension*) const [inline]

Return the index of the group in the given dimension.

Definition at line 87 of file group.hpp.

```
00087 { return get()[dimension]; }
```

8.9.2.1.2.3 template<std::size_t dims> range<dims> cl::sycl::group< dims >::get_global_range ( ) const [inline]

Get the local range for this work_group.

Definition at line 122 of file group.hpp.

```
00122 { return get_nd_range().get_global(); }
```

8.9.2.1.2.4 template< std::size_t dims > size_t cl::sycl::group< dims >::get_global_range (int *dimension*) const [inline]

Return element dimension from the constituent global range.

Definition at line 126 of file group.hpp.

```
                                               {
    return get_global_range()[dimension];
  }
```

8.9.2.1.2.5 template< std::size_t dims > range< dims > cl::sycl::group< dims >::get_group_range ( ) const [inline]

Return a range<> representing the dimensions of the current group.

This local range may have been provided by the programmer, or chosen by the runtime.

Todo Fix this comment and the specification

Definition at line 110 of file group.hpp.

```
                                      {
    return get_nd_range().get_group();
  }
```

8.9.2.1.2.6 template< std::size_t dims > size_t cl::sycl::group< dims >::get_group_range (int *dimension*) const [inline]

Return element dimension from the con stituent group range.

Definition at line 116 of file group.hpp.

```
                                              {
    return get_group_range()[dimension];
  }
```

8.9.2.1.2.7 template< std::size_t dims > size_t cl::sycl::group< dims >::get_linear ( ) const [inline]

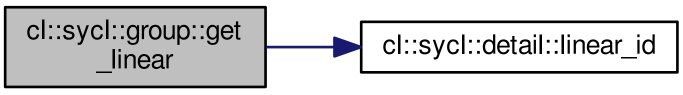
Get a linearized version of the group ID.

Definition at line 168 of file group.hpp.

References cl::sycl::detail::linear_id().

```
                          {
    return detail::linear_id(get_group_range(), get());
  }
```

Here is the call graph for this function:



8.9.2.1.2.8 template<std::size_t dims> range<dims> cl::sycl::group< dims >::get_local_range ( ) const [inline]

Get the local range for this work_group.

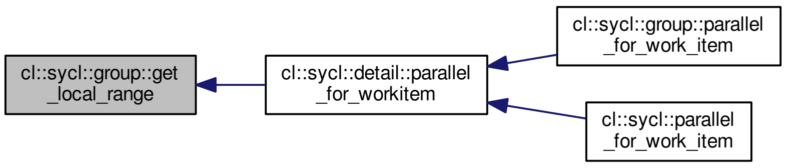
Todo Add to the specification

Definition at line 135 of file group.hpp.

Referenced by cl::sycl::detail::parallel_for_workitem().

```
{ return get_nd_range().get_local(); }
```

Here is the caller graph for this function:


cl::sycl::group::get_local_range
cl::sycl::detail::parallel_for_workitem
cl::sycl::group::parallel_for_work_item
cl::sycl::parallel_for_work_item

8.9.2.1.2.9 template<std::size_t dims> size_t cl::sycl::group< dims >::get_local_range (int *dimension*) const [inline]

Return element dimension from the constituent local range.

Todo Add to the specification

Definition at line 142 of file group.hpp.

```
                                                    {
  return get_local_range()[dimension];
}
```

8.9.2.1.2.10 template<std::size_t dims> nd_range<dims> cl::sycl::group< dims >::get_nd_range ( ) const [inline]

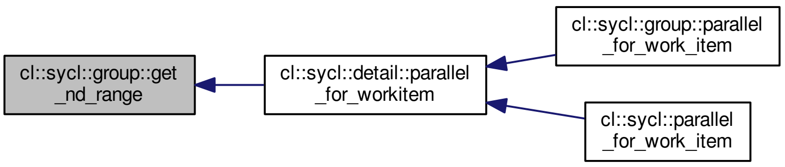
Todo Also provide this access to the current nd_range

Definition at line 162 of file group.hpp.

Referenced by cl::sycl::detail::parallel_for_workitem().

```
00162 { return ndr; }
```

Here is the caller graph for this function:


cl::sycl::group::get_nd_range
cl::sycl::detail::parallel_for_workitem
cl::sycl::group::parallel_for_work_item
cl::sycl::parallel_for_work_item

8.9.2.1.2.11 template<std::size_t dims> id<dims> cl::sycl::group< dims >::get_offset ( ) const [inline]

Get the offset of the NDRange.

Todo Add to the specification

Definition at line 151 of file group.hpp.

```
00151 { return get_nd_range().get_offset(); }
```

8.9.2.1.2.12 template<std::size_t dims> size_t cl::sycl::group< dims >::get_offset (int *dimension*) const [inline]

Get the offset of the NDRange.

Todo Add to the specification

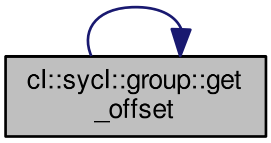
Definition at line 158 of file group.hpp.

References cl::sycl::group< dims >::get_offset().

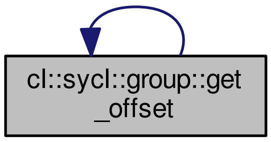
Referenced by cl::sycl::group< dims >::get_offset().

```
00158 { return get_offset()[dimension]; }
```

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.1.2.13 template< std::size_t dims > auto& cl::sycl::group< dims >::operator[ ] (int *dimension*) [inline]

Return the index of the group in the given dimension within the nd_range<>

Todo In this implementation it is not const because the group<> is written in the parallel_for iterators. To fix according to the specification

Definition at line 97 of file group.hpp.

```
                                          {
      return group_id[dimension];
    }
```

8.9.2.1.2.14 template< std::size_t dims > void cl::sycl::group< dims >::parallel_for_work_item (std::function< void(nd_item< dimensionality >)> *f*) const [inline]

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

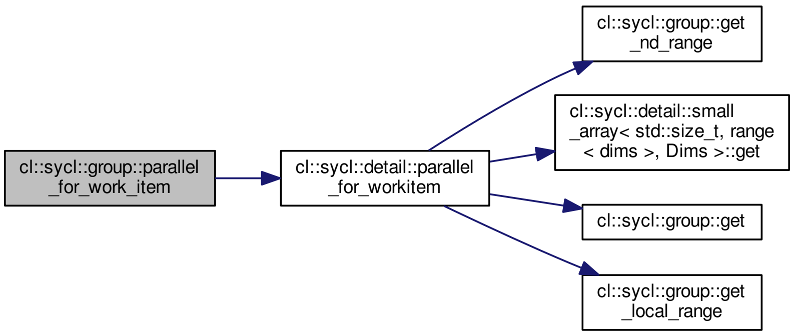
Todo Add this method in the specification

Definition at line 177 of file group.hpp.

References cl::sycl::detail::parallel_for_workitem().

```
        {
    detail::parallel_for_workitem(*this, f);
  }
```

Here is the call graph for this function:


cl::sycl::group::parallel_for_work_item
cl::sycl::detail::parallel_for_workitem
cl::sycl::group::get_nd_range
cl::sycl::detail::small_array< std::size_t, range< dims >, Dims >::get
cl::sycl::group::get
cl::sycl::group::get_local_range

8.9.2.1.2.15 template<std::size_t dims> void cl::sycl::group< dims >::parallel_for_work_item (std::function< void(item< dimensionality >)> *f*) const [inline]

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

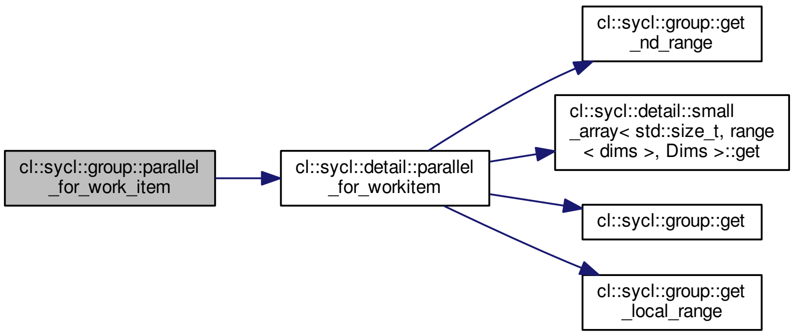
Todo Add this method in the specification

Definition at line 187 of file group.hpp.

References cl::sycl::detail::parallel_for_workitem().

```
        {
    auto item_adapter = [=] (nd_item<dimensionality> ndi) {
      item<dimensionality> i = ndi.get_item();
      f(i);
    };
    detail::parallel_for_workitem(*this, item_adapter);
  }
```

Here is the call graph for this function:


cl::sycl::group::parallel_for_work_item
cl::sycl::detail::parallel_for_workitem
cl::sycl::group::get_nd_range
cl::sycl::detail::small_array< std::size_t, range< dims >, Dims >::get
cl::sycl::group::get
cl::sycl::group::get_local_range

8.9.2.1.3 Member Data Documentation

8.9.2.1.3.1 template< std::size_t dims > constexpr auto cl::sycl::group< dims >::dimensionality = dims [static]

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

Definition at line 44 of file group.hpp.

8.9.2.1.3.2 template< std::size_t dims > id< dims > cl::sycl::group< dims >::group_id [private]

The coordinate of the group item.

Definition at line 49 of file group.hpp.

8.9.2.1.3.3 template< std::size_t dims > nd_range< dims > cl::sycl::group< dims >::ndr [private]

Keep a reference on the nd_range to serve potential query on it.

Definition at line 52 of file group.hpp.

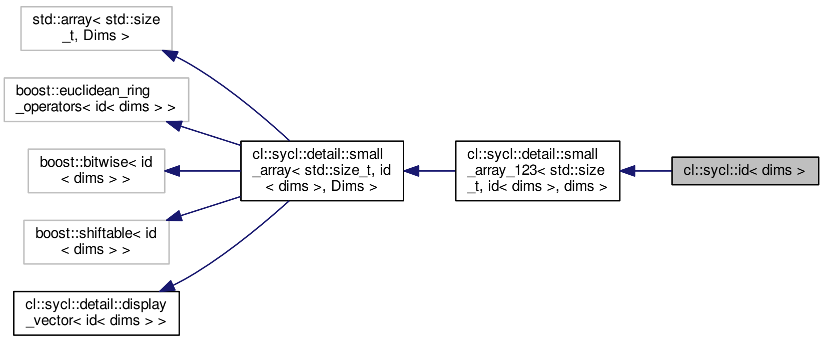
8.9.2.2 class cl::sycl::id

template< std::size_t dims = 1 >
class cl::sycl::id< dims >

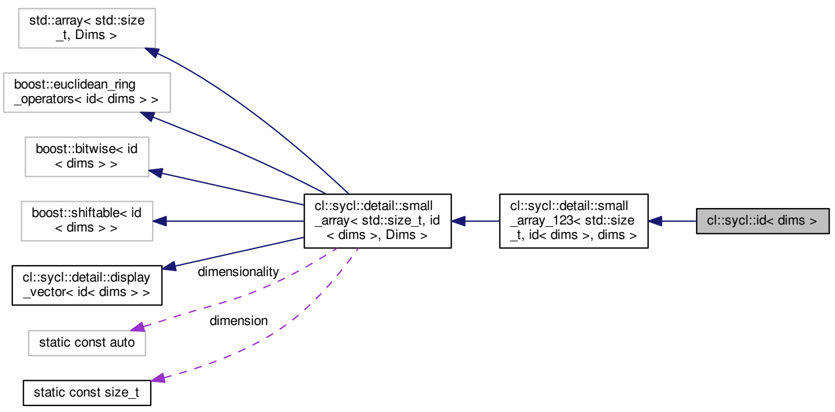
Define a multi-dimensional index, used for example to locate a work item.

Definition at line 31 of file id.hpp.

Inheritance diagram for cl::sycl::id< dims >:


std::array< std::size_t, Dims >
boost::euclidean_ring_operators< id< dims > >
boost::bitwise< id< dims > >
boost::shiftable< id< dims > >
cl::sycl::detail::display_vector< id< dims > >
cl::sycl::detail::small_array< std::size_t, id< dims >, Dims >
cl::sycl::detail::small_array_123< std::size_t, id< dims >, dims >
cl::sycl::id< dims >

Collaboration diagram for cl::sycl::id< dims >:


std::array< std::size _t, Dims >
boost::euclidean_ring _operators< id< dims > >
boost::bitwise< id < dims > >
boost::shiftable< id < dims > >
cl::sycl::detail::display _vector< id< dims > >
static const auto
static const size_t
dimensionality
dimension
cl::sycl::detail::small _array< std::size_t, id < dims >, Dims >
cl::sycl::detail::small _array_123< std::size _t, id< dims >, dims >
cl::sycl::id< dims >

Public Member Functions

- id (const range< dims > &range_size)
  Construct an id from the dimensions of a range.

Additional Inherited Members

8.9.2.2.1 Constructor & Destructor Documentation

8.9.2.2.1.1 template<std::size_t dims = 1> cl::sycl::id< dims >::id (const range< dims > & *range_size*) [inline]

Construct an id from the dimensions of a range.

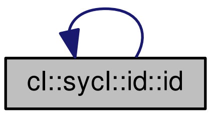
Use the fact we have a constructor of a small_array from a another kind of small_array

Definition at line 42 of file id.hpp.

Referenced by cl::sycl::id< dimensionality >::id().

```
00046     : detail::small_array_123<std::size_t, id<dims>, dims> { range_size } {}
```

Here is the caller graph for this function:



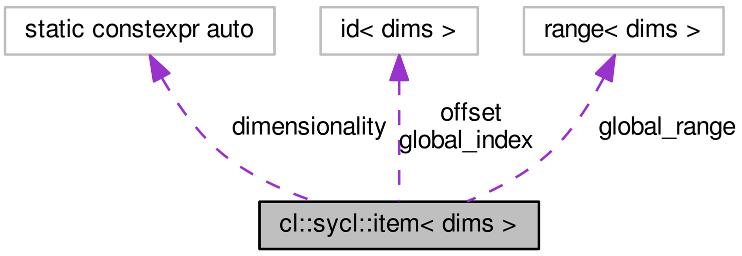
8.9.2.3 class cl::sycl::item

template<std::size_t dims = 1>
class cl::sycl::item< dims >

A SYCL item stores information on a work-item with some more context such as the definition range and offset.

Definition at line 21 of file id.hpp.

Collaboration diagram for cl::sycl::item< dims >:



Public Member Functions

- item (range< dims > global_size, id< dims > global_index, id< dims > offset={})

  Create an item from a local size and an optional offset.
- item ()=default

  To be able to copy and assign item, use default constructors too.
- id< dims > get () const

  Return the constituent local or global id<> representing the work-item's position in the iteration space.
- size_t get (int dimension) const

  Return the requested dimension of the constituent id<> representing the work-item's position in the iteration space.
- auto & operator[ ] (int dimension)

  Return the constituent id<> l-value representing the work-item's position in the iteration space in the given dimension.
- range< dims > get_range () const

  Returns a range<> representing the dimensions of the range of possible values of the item.
- id< dims > get_offset () const

  Returns an id<> representing the n-dimensional offset provided to the parallel_for and that is added by the runtime to the global-ID of each work-item, if this item represents a global range.
- size_t get_linear_id () const

  Return the linearized ID in the item's range.
- void set (id< dims > Index)

  For the implementation, need to set the global index.
- void display () const

  Display the value for debugging and validation purpose.

Static Public Attributes

- static constexpr auto dimensionality = dims

Private Attributes

- range< dims > global_range
- id< dims > global_index
- id< dims > offset

8.9.2.3.1 Constructor & Destructor Documentation

8.9.2.3.1.1 template<std::size_t dims = 1> cl::sycl::item< dims >::item (range< dims > *global_size,* id< dims > *global_index,* id< dims > *offset =* { }) [inline]

Create an item from a local size and an optional offset.

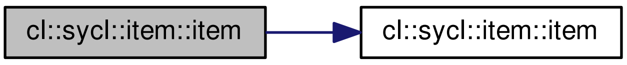
This constructor is used by the triSYCL implementation and the non-regression testing.

Definition at line 50 of file item.hpp.

References cl::sycl::item< dims >::item().

```
                           {}) :
    global_range { global_size },
    global_index { global_index },
    offset { offset }
  {}
```

Here is the call graph for this function:



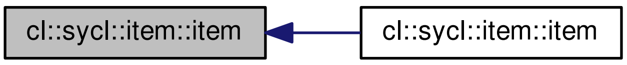
8.9.2.3.1.2 template<std::size_t dims = 1> cl::sycl::item< dims >::item ( ) [default]

To be able to copy and assign item, use default constructors too.

Todo Make most of them protected, reserved to implementation

Referenced by cl::sycl::item< dims >::item().

Here is the caller graph for this function:



8.9.2.3.2 Member Function Documentation

8.9.2.3.2.1 template<std::size_t dims = 1> void **cl::sycl::item**< dims >::display ( ) const [inline]

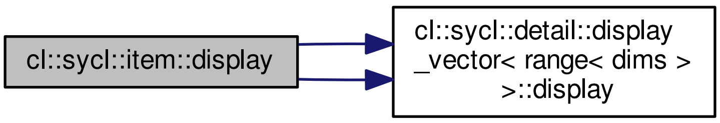
Display the value for debugging and validation purpose.

Definition at line 117 of file item.hpp.

References cl::sycl::detail::display_vector< range< dims > >::display(), and cl::sycl::detail::display_vector< id< dims > >::display().

```
                        {
    global_range.display();
    global_index.display();
    offset.display();
  }
```

Here is the call graph for this function:



8.9.2.3.2.2 template<std::size_t dims = 1> id<dims> **cl::sycl::item**< dims >::get ( ) const [inline]

Return the constituent local or global id<> representing the work-item's position in the iteration space.

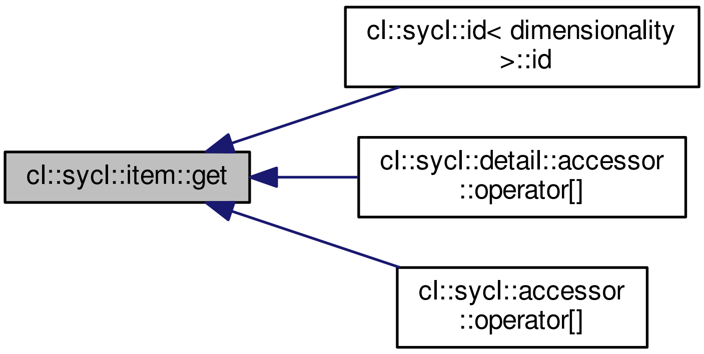
Definition at line 69 of file item.hpp.

References cl::sycl::item< dims >::global_index.

Referenced by cl::sycl::id< dimensionality >::id(), cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ](), and cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[ ]().

```
{ return global_index; }
```

Here is the caller graph for this function:


cl::sycl::item::get
cl::sycl::id< dimensionality >::id
cl::sycl::detail::accessor ::operator[]
cl::sycl::accessor ::operator[]

8.9.2.3.2.3 template<std::size_t dims = 1> size_t cl::sycl::item< dims >::get (int *dimension*) const [inline]

Return the requested dimension of the constituent id<> representing the work-item's position in the iteration space.

Definition at line 75 of file item.hpp.

```
{ return get()[dimension]; }
```

8.9.2.3.2.4 template<std::size_t dims = 1> size_t cl::sycl::item< dims >::get_linear_id ( ) const [inline]

Return the linearized ID in the item's range.

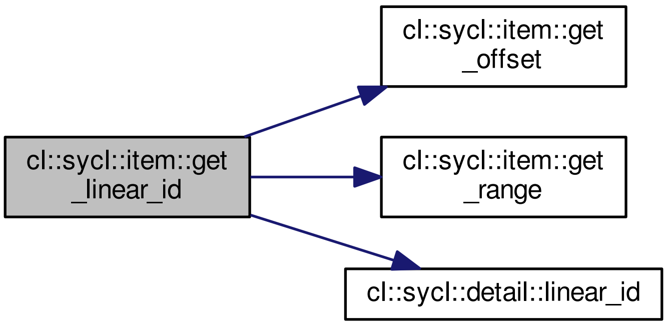
Computed as the flatted ID after the offset is subtracted.

Definition at line 104 of file item.hpp.

References cl::sycl::item< dims >::get_offset(), cl::sycl::item< dims >::get_range(), and cl::sycl::detail::linear_id().

```
                                {
    return detail::linear_id(get_range(), get(),
      get_offset());
  }
```

Here is the call graph for this function:



8.9.2.3.2.5 template<std::size_t dims = 1> id<dims> cl::sycl::item< dims >::get_offset ( ) const [inline]

Returns an id<> representing the n-dimensional offset provided to the parallel_for and that is added by the runtime to the global-ID of each work-item, if this item represents a global range.

For an item representing a local range of where no offset was passed this will always return an id of all 0 values.

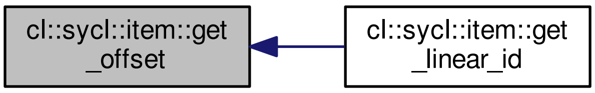
Definition at line 97 of file item.hpp.

References cl::sycl::item< dims >::offset.

Referenced by cl::sycl::item< dims >::get_linear_id().

```
00097 { return offset; }
```

Here is the caller graph for this function:



8.9.2.3.2.6 template<std::size_t dims = 1> range<dims> cl::sycl::item< dims >::get_range ( ) const [inline]

Returns a range<> representing the dimensions of the range of possible values of the item.

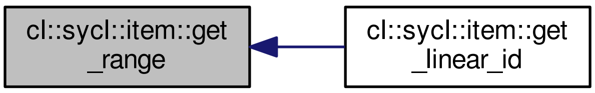
Definition at line 87 of file item.hpp.

References cl::sycl::item< dims >::global_range.

Referenced by cl::sycl::item< dims >::get_linear_id().

```
00087 { return global_range; }
```

Here is the caller graph for this function:



8.9.2.3.2.7 template<std::size_t dims = 1> auto& cl::sycl::item< dims >::operator[ ] (int *dimension*) [inline]

Return the constituent id<> l-value representing the work-item's position in the iteration space in the given dimension.

Definition at line 81 of file item.hpp.

```
00081 { return global_index[dimension]; }
```

8.9.2.3.2.8 template<std::size_t dims = 1> void cl::sycl::item< dims >::set (id< dims > *Index*) [inline]

For the implementation, need to set the global index.

Todo Move to private and add friends

Definition at line 113 of file item.hpp.

```
00113 { global_index = Index; }
```

8.9.2.3.3 Member Data Documentation

8.9.2.3.3.1 template<std::size_t dims = 1> constexpr auto cl::sycl::item< dims >::dimensionality = dims [static]

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

Definition at line 35 of file item.hpp.

8.9.2.3.3.2 template<std::size_t dims = 1> id<dims> cl::sycl::item< dims >::global_index [private]

Definition at line 40 of file item.hpp.

Referenced by cl::sycl::item< dims >::get().

8.9.2.3.3.3 template<std::size_t dims = 1> range<dims> cl::sycl::item< dims >::global_range [private]

Definition at line 39 of file item.hpp.

Referenced by cl::sycl::item< dims >::get_range().

8.9.2.3.3.4 template<std::size_t dims = 1> id<dims> cl::sycl::item< dims >::offset [private]

Definition at line 41 of file item.hpp.

Referenced by cl::sycl::item< dims >::get_offset().

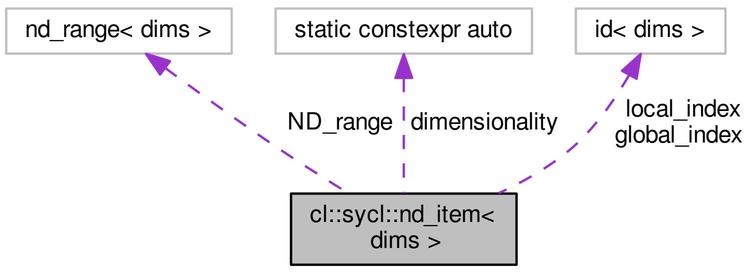
8.9.2.4 struct cl::sycl::nd_item

template< std::size_t dims = 1 >
struct cl::sycl::nd_item< dims >

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

Definition at line 33 of file nd_item.hpp.

Collaboration diagram for cl::sycl::nd_item< dims >:



Public Member Functions

- nd_item (nd_range< dims > ndr)

  Create an empty nd_item<> from an nd_range<>
- nd_item (id< dims > global_index, nd_range< dims > ndr)

  Create a full nd_item.
- nd_item ()=default

  To be able to copy and assign nd_item, use default constructors too.
- id< dims > get_global () const

  Return the constituent global id representing the work-item's position in the global iteration space.
- size_t get_global (int dimension) const

  Return the constituent element of the global id representing the work-item's position in the global iteration space in the given dimension.
- size_t get_global_linear_id () const

  Return the flattened id of the current work-item after subtracting the offset.
- id< dims > get_local () const

  Return the constituent local id representing the work-item's position within the current work-group.
- size_t get_local (int dimension) const

  Return the constituent element of the local id representing the work-item's position within the current work-group in the given dimension.
- size_t get_local_linear_id () const

  Return the flattened id of the current work-item within the current work-group.
- id< dims > get_group () const

  Return the constituent group group representing the work-group's position within the overall nd_range.
- size_t get_group (int dimension) const

Return the constituent element of the group id representing the work-group;s position within the overall nd_range in the given dimension.

- size_t get_group_linear_id () const

  Return the flattened id of the current work-group.
- id< dims > get_num_groups () const

  Return the number of groups in the nd_range.
- size_t get_num_groups (int dimension) const

  Return the number of groups for dimension in the nd_range.
- range< dims > get_global_range () const

  Return a range<> representing the dimensions of the nd_range<>
- range< dims > get_local_range () const

  Return a range<> representing the dimensions of the current work-group.
- id< dims > get_offset () const

  Return an id<> representing the n-dimensional offset provided to the constructor of the nd_range<> and that is added by the runtime to the global-ID of each work-item.
- nd_range< dims > get_nd_range () const

  Return the nd_range<> of the current execution.
- item< dims > get_item () const

  Allows projection down to an item.
- void barrier (access::fence_space flag=access::fence_space::global_and_local) const

  Execute a barrier with memory ordering on the local address space, global address space or both based on the value of flag.
- void set_local (id< dims > Index)
- void set_global (id< dims > Index)

Static Public Attributes

- static constexpr auto dimensionality = dims

Private Attributes

- id< dims > global_index
- id< dims > local_index
- nd_range< dims > ND_range

8.9.2.4.1 Constructor & Destructor Documentation

8.9.2.4.1.1 template<std::size_t dims = 1> cl::sycl::nd_item< dims >::nd_item (nd_range< dims > *ndr*) [inline]

Create an empty nd_item<> from an nd_range<>

Todo This is for the triSYCL implementation which is expected to call set_global() and set_local() later. This should be hidden to the user.

Definition at line 54 of file nd_item.hpp.

```
00054 : ND_range { ndr } {}
```

8.9.2.4.1.2 template< std::size_t dims = 1> cl::sycl::nd_item< dims >::nd_item (id< dims > *global_index*, nd_range< dims > *ndr*) [inline]

Create a full nd_item.

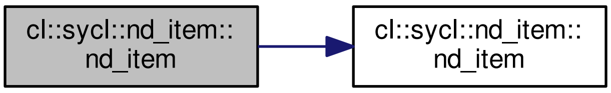
Todo This is for validation purpose. Hide this to the programmer somehow

Definition at line 62 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::nd_item().

```
                               :
    global_index { global_index },
    // Compute the local index using the offset and the group size
    local_index { (global_index - ndr.get_offset())%id<dims> { ndr.get_local() } },
    ND_range { ndr }
  {}
```

Here is the call graph for this function:



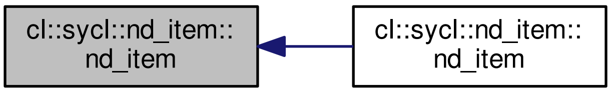
8.9.2.4.1.3 template< std::size_t dims = 1> cl::sycl::nd_item< dims >::nd_item ( ) [default]

To be able to copy and assign nd_item, use default constructors too.

Todo Make most of them protected, reserved to implementation

Referenced by cl::sycl::nd_item< dims >::nd_item().

Here is the caller graph for this function:



8.9.2.4.2 Member Function Documentation

8.9.2.4.2.1 template<std::size_t dims = 1> void cl::sycl::nd_item< dims >::barrier (access::fence_space *flag* = access::fence_space::global_and_local) const [inline]

Execute a barrier with memory ordering on the local address space, global address space or both based on the value of flag.

The current work-item will wait at the barrier until all work-items in the current work-group have reached the barrier.

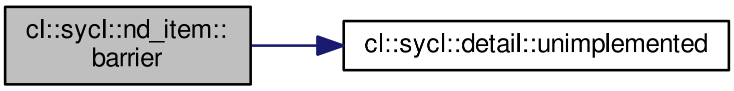
In addition, the barrier performs a fence operation ensuring that all memory accesses in the specified address space issued before the barrier complete before those issued after the barrier

Definition at line 198 of file nd_item.hpp.

References cl::sycl::detail::unimplemented().

```
                                                          {
#if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
    /* Use OpenMP barrier in the implementation with 1 OpenMP thread per
       work-item of the work-group */
#pragma omp barrier
#else
    // \todo To be implemented efficiently otherwise
    detail::unimplemented();
#endif
  }
```

Here is the call graph for this function:



8.9.2.4.2.2 template<std::size_t dims = 1> id<dims> cl::sycl::nd_item< dims >::get_global ( ) const [inline]

Return the constituent global id representing the work-item's position in the global iteration space.

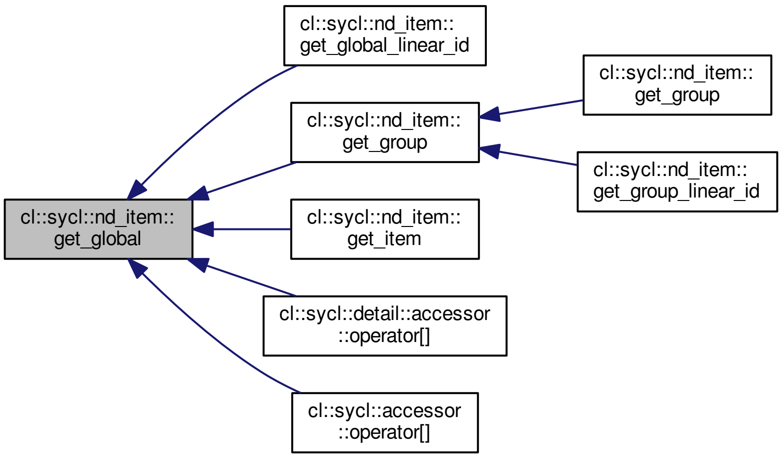
Definition at line 81 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::global_index.

Referenced by cl::sycl::nd_item< dims >::get_global_linear_id(), cl::sycl::nd_item< dims >::get_group(), cl::sycl↩::nd_item< dims >::get_item(), cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ](), and cl↩::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[ ]().

```
{ return global_index; }
```

Here is the caller graph for this function:



8.9.2.4.2.3 template<std::size_t dims = 1> size_t cl::sycl::nd_item< dims >::get_global (int *dimension*) const [inline]

Return the constituent element of the global id representing the work-item's position in the global iteration space in the given dimension.

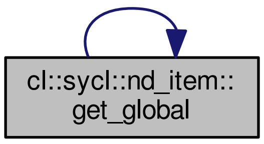
Definition at line 88 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_global().

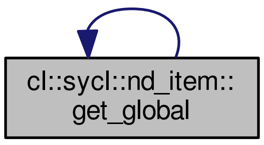
Referenced by cl::sycl::nd_item< dims >::get_global().

```
00088 { return get_global()[dimension]; }
```

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.4.2.4 template< std::size_t dims = 1 > size_t cl::sycl::nd_item< dims >::get_global_linear_id ( ) const [inline]

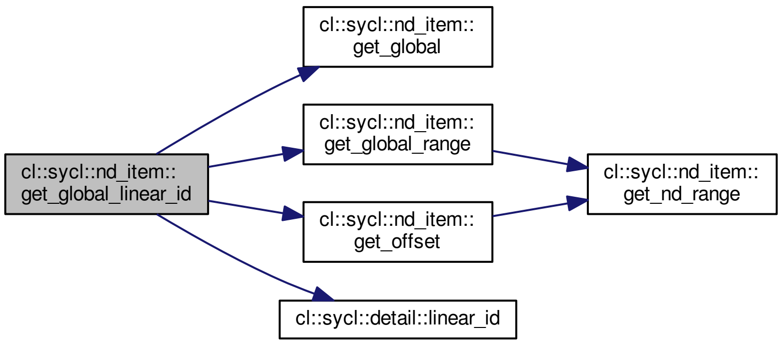
Return the flattened id of the current work-item after subtracting the offset.

Definition at line 94 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_global(), cl::sycl::nd_item< dims >::get_global_range(), cl::sycl::nd_item< dims >::get_offset(), and cl::sycl::detail::linear_id().

```
                                          {
    return detail::linear_id(get_global_range(),
  get_global(), get_offset());
  }
```

Here is the call graph for this function:



8.9.2.4.2.5 template<std::size_t dims = 1> range<dims> cl::sycl::nd_item< dims >::get_global_range ( ) const [inline]

Return a range<> representing the dimensions of the nd_range<>

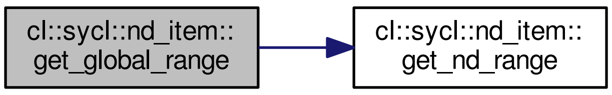
Definition at line 157 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_nd_range().

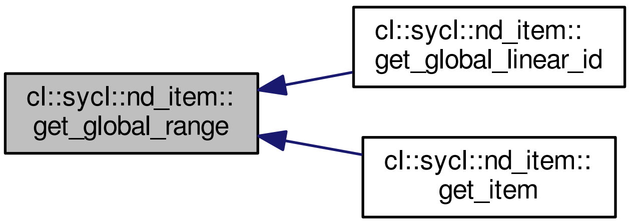
Referenced by cl::sycl::nd_item< dims >::get_global_linear_id(), and cl::sycl::nd_item< dims >::get_item().

```
00157                                          {
00158     return get_nd_range().get_global();
00159   }
```

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.4.2.6 template<std::size_t dims = 1> id<dims> cl::sycl::nd_item< dims >::get_group ( ) const [inline]

Return the constituent group group representing the work-group's position within the overall nd_range.

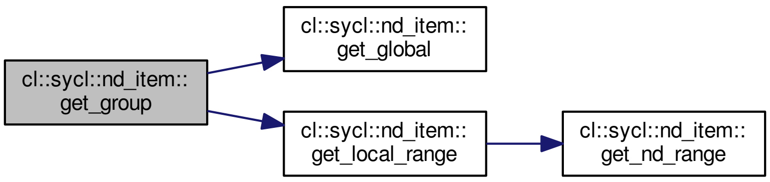
Definition at line 123 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_global(), and cl::sycl::nd_item< dims >::get_local_range().

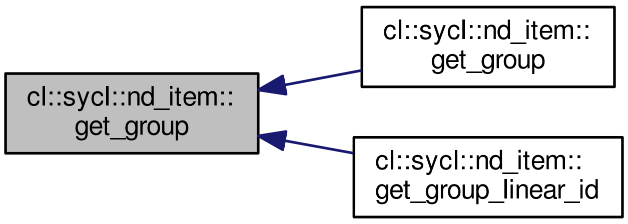
Referenced by cl::sycl::nd_item< dims >::get_group(), and cl::sycl::nd_item< dims >::get_group_linear_id().

```
                            {
    /* Convert get_local_range() to an id<> to remove ambiguity into using
       implicit conversion either from range<> to id<> or the opposite */
    return get_global()/id<dims> { get_local_range() };
  }
```

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.4.2.7 template<std::size_t dims = 1> size_t **cl::sycl::nd_item**< dims >::get_group (int *dimension*) const [inline]

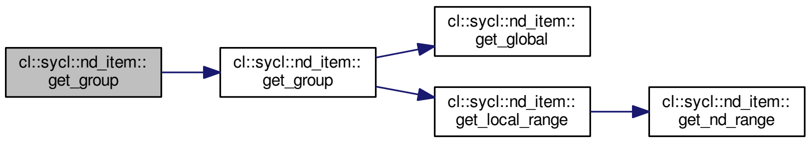
Return the constituent element of the group id representing the work-group;s position within the overall nd_range in the given dimension.

Definition at line 134 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_group().

```
                                        {
    return get_group()[dimension];
  }
```

Here is the call graph for this function:



8.9.2.4.2.8 template< std::size_t dims = 1> size_t **cl::sycl::nd_item**< dims >::get_group_linear_id ( ) const [inline]

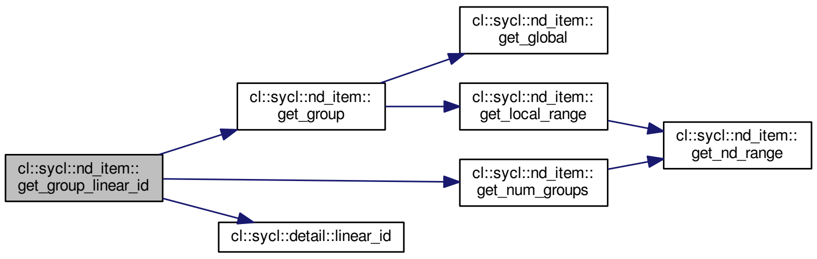
Return the flattened id of the current work-group.

Definition at line 140 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_group(), cl::sycl::nd_item< dims >::get_num_groups(), and cl::sycl::detail::linear_id().

```
                                        {
      return detail::linear_id(get_num_groups(),
      get_group());
    }
```

Here is the call graph for this function:



8.9.2.4.2.9 template< std::size_t dims = 1> item<dims> **cl::sycl::nd_item**< dims >::get_item ( ) const [inline]

Allows projection down to an item.

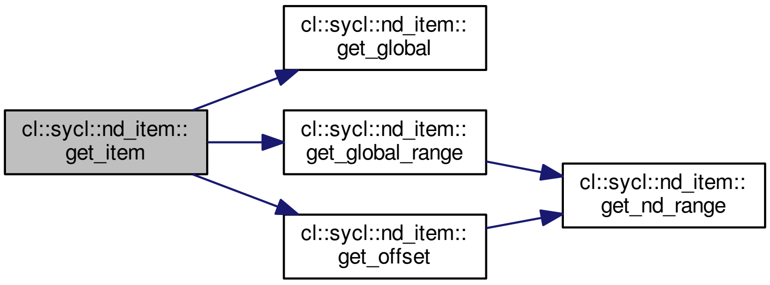
Todo Add to the specification

Definition at line 183 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_global(), cl::sycl::nd_item< dims >::get_global_range(), and cl::sycl::nd_item< dims >::get_offset().

```
                                  {
    return { get_global_range(), get_global(),
get_offset() };
  }
```

Here is the call graph for this function:



8.9.2.4.2.10 template<std::size_t dims = 1> id<dims> cl::sycl::nd_item< dims >::get_local ( ) const [inline]

Return the constituent local id representing the work-item's position within the current work-group.

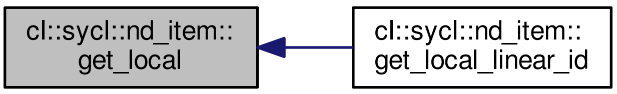
Definition at line 102 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::local_index.

Referenced by cl::sycl::nd_item< dims >::get_local_linear_id().

```
{ return local_index; }
```

Here is the caller graph for this function:



8.9.2.4.2.11 template<std::size_t dims = 1> size_t **cl::sycl::nd_item**< dims >::get_local (int *dimension*) const [inline]

Return the constituent element of the local id representing the work-item's position within the current work-group in the given dimension.

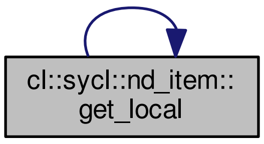
Definition at line 109 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_local().

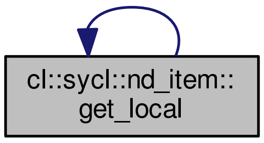
Referenced by cl::sycl::nd_item< dims >::get_local().

```
00109 { return get_local()[dimension]; }
```

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.4.2.12 template<std::size_t dims = 1> size_t **cl::sycl::nd_item**< dims >::get_local_linear_id ( ) const [inline]

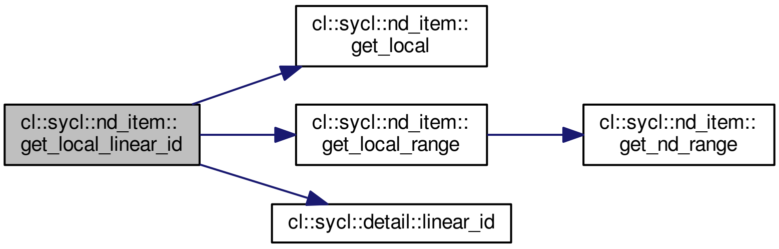
Return the flattened id of the current work-item within the current work-group.

Definition at line 115 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_local(), cl::sycl::nd_item< dims >::get_local_range(), and cl::sycl↩
::detail::linear_id().

```
                                        {
    return detail::linear_id(get_local_range(),
      get_local());
  }
```

Here is the call graph for this function:



8.9.2.4.2.13 template<std::size_t dims = 1> range<dims> cl::sycl::nd_item< dims >::get_local_range ( ) const [inline]

Return a range<> representing the dimensions of the current work-group.

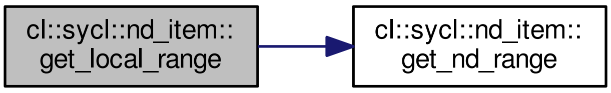
Definition at line 163 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_nd_range().

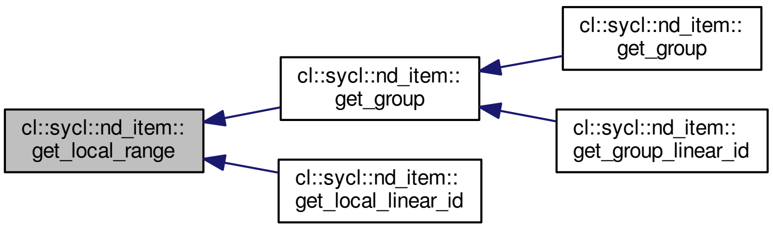
Referenced by cl::sycl::nd_item< dims >::get_group(), and cl::sycl::nd_item< dims >::get_local_linear_id().

```
                                         {
    return get_nd_range().get_local();
  }
```

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.4.2.14 template<std::size_t dims = 1> nd_range<dims> cl::sycl::nd_item< dims >::get_nd_range ( ) const [inline]

Return the nd_range<> of the current execution.

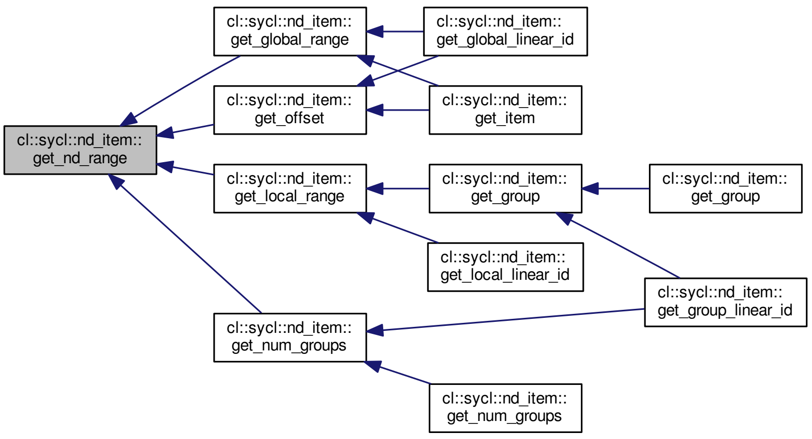
Definition at line 176 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::ND_range.

Referenced by cl::sycl::nd_item< dims >::get_global_range(), cl::sycl::nd_item< dims >::get_local_range(), cl↵
::sycl::nd_item< dims >::get_num_groups(), and cl::sycl::nd_item< dims >::get_offset().

```
00176 { return ND_range; }
```

Here is the caller graph for this function:



8.9.2.4.2.15 template<std::size_t dims = 1> id<dims> cl::sycl::nd_item< dims >::get_num_groups ( ) const [inline]

Return the number of groups in the nd_range.

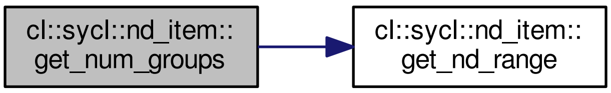
Definition at line 146 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_nd_range().

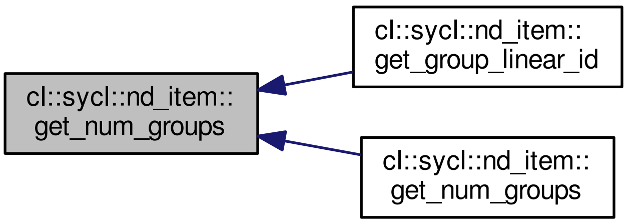
Referenced by cl::sycl::nd_item< dims >::get_group_linear_id(), and cl::sycl::nd_item< dims >::get_num_groups().

```
00146                              {
00147     return get_nd_range().get_group();
00148   }
```

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.4.2.16 template<std::size_t dims = 1> size_t cl::sycl::nd_item< dims >::get_num_groups (int *dimension*) const [inline]

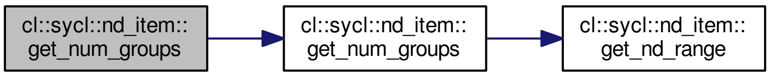
Return the number of groups for dimension in the nd_range.

Definition at line 151 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_num_groups().

```
                                                           {
        return get_num_groups()[dimension];
    }
```

Here is the call graph for this function:



8.9.2.4.2.17 template<std::size_t dims = 1> id<dims> cl::sycl::nd_item< dims >::get_offset ( ) const [inline]

Return an id<> representing the n-dimensional offset provided to the constructor of the nd_range<> and that is added by the runtime to the global-ID of each work-item.

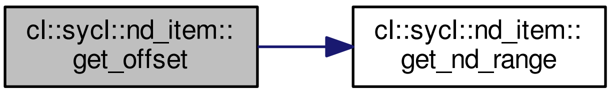
Definition at line 172 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_nd_range().

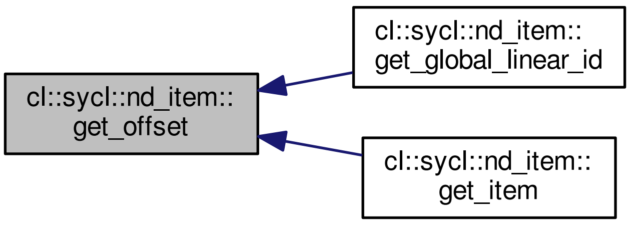
Referenced by cl::sycl::nd_item< dims >::get_global_linear_id(), and cl::sycl::nd_item< dims >::get_item().

```
{ return get_nd_range().get_offset(); }
```

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.4.2.18 template< std::size_t dims = 1> void cl::sycl::nd_item< dims >::set_global (id< dims > *Index*) [inline]

Definition at line 216 of file nd_item.hpp.

```
00216 { global_index = Index; }
```

8.9.2.4.2.19 template< std::size_t dims = 1> void cl::sycl::nd_item< dims >::set_local (id< dims > *Index*) [inline]

Definition at line 212 of file nd_item.hpp.

```
00212 { local_index = Index; }
```

8.9.2.4.3 Member Data Documentation

8.9.2.4.3.1 template< std::size_t dims = 1> constexpr auto cl::sycl::nd_item< dims >::dimensionality = dims [static]

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

Definition at line 36 of file nd_item.hpp.

8.9.2.4.3.2 template< std::size_t dims = 1> id<dims> cl::sycl::nd_item< dims >::global_index [private]

Definition at line 40 of file nd_item.hpp.

Referenced by cl::sycl::nd_item< dims >::get_global().

8.9.2.4.3.3 template< std::size_t dims = 1> id<dims> cl::sycl::nd_item< dims >::local_index [private]

Definition at line 43 of file nd_item.hpp.

Referenced by cl::sycl::nd_item< dims >::get_local().

8.9.2.4.3.4 template< std::size_t dims = 1> nd_range<dims> cl::sycl::nd_item< dims >::ND_range [private]

Definition at line 44 of file nd_item.hpp.

Referenced by cl::sycl::nd_item< dims >::get_nd_range().

8.9.2.5 struct cl::sycl::nd_range

template< std::size_t dims = 1 >
struct cl::sycl::nd_range< dims >

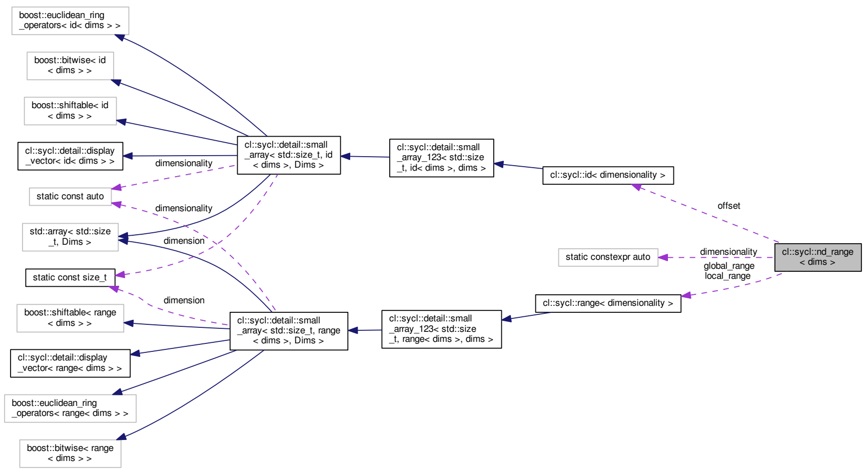
A ND-range, made by a global and local range, to specify work-group and work-item organization.

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

Todo add copy constructors in the specification

Definition at line 33 of file nd_range.hpp.

Collaboration diagram for cl::sycl::nd_range< dims >:


boost::euclidean_ring _operators< id< dims > >
boost::bitwise< id < dims > >
boost::shiftable< id < dims > >
cl::sycl::detail::display _vector< id< dims > >
static const auto
std::array< std::size _t, Dims >
static const size_t
boost::shiftable< range < dims > >
cl::sycl::detail::display _vector< range< dims > >
boost::euclidean_ring _operators< range< dims > >
boost::bitwise< range < dims > >
dimensionality
dimensionality
dimension
dimension
cl::sycl::detail::small _array< std::size_t, id < dims >, Dims >
cl::sycl::detail::small _array< std::size_t, range < dims >, Dims >
cl::sycl::detail::small _array_123< std::size _t, id< dims >, dims >
cl::sycl::detail::small _array_123< std::size _t, range< dims >, dims >
cl::sycl::id< dimensionality >
cl::sycl::range< dimensionality >
offset
static constexpr auto
dimensionality_
global_range
local_range
cl::sycl::nd_range < dims >

Public Member Functions

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

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

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

    Get the local part of the iteration space range.
- auto get_group () const

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

    Display the value for debugging and validation purpose.

Static Public Attributes

- static constexpr auto dimensionality = dims

Private Attributes

- range< dimensionality > global_range
- range< dimensionality > local_range
- id< dimensionality > offset

8.9.2.5.1 Constructor & Destructor Documentation

8.9.2.5.1.1 template<std::size_t dims = 1> cl::sycl::nd_range< dims >::nd_range (range< dims > *global_size*, range< dims > *local_size*, id< dims > *offset* = { }) [inline]

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

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

Definition at line 50 of file nd_range.hpp.

```
                                {}) :
    global_range { global_size }, local_range { local_size },
offset { offset }
  { }
```

8.9.2.5.2 Member Function Documentation

8.9.2.5.2.1 template<std::size_t dims = 1> void cl::sycl::nd_range< dims >::display ( ) const [inline]

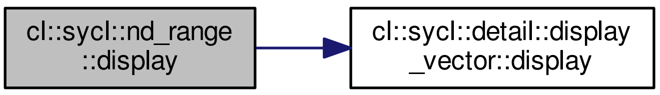
Display the value for debugging and validation purpose.

Definition at line 80 of file nd_range.hpp.

References cl::sycl::detail::display_vector< T >::display().

```
                          {
    global_range.display();
    local_range.display();
    offset.display();
  }
```

Here is the call graph for this function:


cl::sycl::nd_range ::display
cl::sycl::detail::display _vector::display

8.9.2.5.2.2 template<std::size_t dims = 1> range<dims> cl::sycl::nd_range< dims >::get_global ( ) const [inline]

Get the global iteration space range.

Definition at line 58 of file nd_range.hpp.

References cl::sycl::nd_range< dims >::global_range.

```
{ return global_range; }
```

8.9.2.5.2.3 template<std::size_t dims = 1> auto cl::sycl::nd_range< dims >::get_group ( ) const [inline]

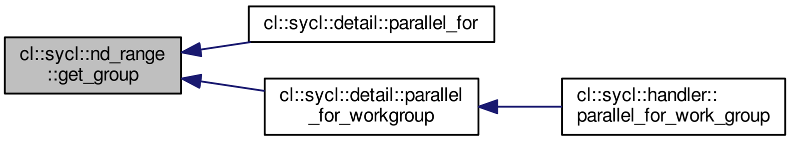
Get the range of work-groups needed to run this ND-range.

Definition at line 66 of file nd_range.hpp.

Referenced by cl::sycl::detail::parallel_for(), and cl::sycl::detail::parallel_for_workgroup().

```
                         {
    /* This is basically global_range/local_range, round up to the
       next integer, in case the global eange is not a multiple of the
       local range. Note this is a motivating example to build a range
       from a scalar with a broadcasting constructor. */
    return (global_range + local_range - range<dims>{ 1 })/
local_range;
  }
```

Here is the caller graph for this function:


cl::sycl::nd_range ::get_group
cl::sycl::detail::parallel_for
cl::sycl::detail::parallel _for_workgroup
cl::sycl::handler:: parallel_for_work_group

8.9.2.5.2.4 template<std::size_t dims = 1> range<dims> cl::sycl::nd_range< dims >::get_local ( ) const [inline]

Get the local part of the iteration space range.

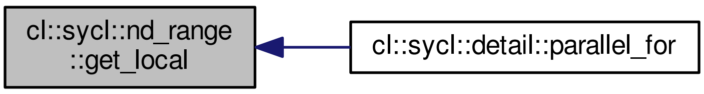
Definition at line 62 of file nd_range.hpp.

References cl::sycl::nd_range< dims >::local_range.

Referenced by cl::sycl::detail::parallel_for().

```
{ return local_range; }
```

Here is the caller graph for this function:


cl::sycl::nd_range ::get_local
cl::sycl::detail::parallel_for

8.9.2.5.2.5 template<std::size_t dims = 1> id<dims> cl::sycl::nd_range< dims >::get_offset ( ) const [inline]

Todo get_offset() is lacking in the specification

Definition at line 76 of file nd_range.hpp.

References cl::sycl::nd_range< dims >::offset.

```
00076 { return offset; }
```

8.9.2.5.3 Member Data Documentation

8.9.2.5.3.1 template<std::size_t dims = 1> constexpr auto cl::sycl::nd_range< dims >::dimensionality = dims [static]

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

Definition at line 36 of file nd_range.hpp.

8.9.2.5.3.2 template<std::size_t dims = 1> range<dimensionality> cl::sycl::nd_range< dims >::global_range [private]

Definition at line 40 of file nd_range.hpp.

Referenced by cl::sycl::nd_range< dims >::get_global().

8.9.2.5.3.3 template<std::size_t dims = 1> range<dimensionality> cl::sycl::nd_range< dims >::local_range [private]

Definition at line 41 of file nd_range.hpp.

Referenced by cl::sycl::nd_range< dims >::get_local().

8.9.2.5.3.4 template<std::size_t dims = 1> id<dimensionality> cl::sycl::nd_range< dims >::offset [private]

Definition at line 42 of file nd_range.hpp.

Referenced by cl::sycl::nd_range< dims >::get_offset().

8.9.2.6 struct cl::sycl::detail::parallel_for_iterate

**template<std::size_t level, typename Range, typename ParallelForFunctor, typename Id>
struct cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id >**

A recursive multi-dimensional iterator that ends up calling f.

The iteration order may be changed later.

Since partial specialization of function template is not possible in C++14, use a class template instead with everything in the constructor.

Definition at line 47 of file parallelism.hpp.

Public Member Functions

- parallel_for_iterate (Range r, ParallelForFunctor &f, Id &index)

8.9.2.6.1 Constructor & Destructor Documentation

8.9.2.6.1.1 template<std::size_t level, typename Range , typename ParallelForFunctor , typename Id > cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id >::parallel_for_iterate (Range *r*, ParallelForFunctor & *f*, Id & *index*) [inline]

Definition at line 48 of file parallelism.hpp.

```
                                                                      {
    for (boost::multi_array_types::index _sycl_index = 0,
           _sycl_end = r[Range::dimensionality - level];
         _sycl_index < _sycl_end;
         _sycl_index++) {
      // Set the current value of the index for this dimension
      index[Range::dimensionality - level] = _sycl_index;
      // Iterate further on lower dimensions
      parallel_for_iterate<level - 1,
                           Range,
                           ParallelForFunctor,
                           Id> { r, f, index };
    }
  }
```

8.9.2.7 struct cl::sycl::detail::parallel_OpenMP_for_iterate

template<std::size_t level, typename Range, typename ParallelForFunctor, typename Id>
struct cl::sycl::detail::parallel_OpenMP_for_iterate< level, Range, ParallelForFunctor, Id >

A top-level recursive multi-dimensional iterator variant using OpenMP.

Only the top-level loop uses OpenMP and goes on with the normal recursive multi-dimensional.

Definition at line 74 of file parallelism.hpp.

Public Member Functions

- parallel_OpenMP_for_iterate (Range r, ParallelForFunctor &f)

8.9.2.7.1 Constructor & Destructor Documentation

8.9.2.7.1.1 template<std::size_t level, typename Range , typename ParallelForFunctor , typename Id > cl::sycl::detail::parallel_OpenMP_for_iterate< level, Range, ParallelForFunctor, Id >::parallel_OpenMP_for_iterate (Range *r*, ParallelForFunctor & *f*) [inline]

Definition at line 75 of file parallelism.hpp.

```
                                                             {
    // Create the OpenMP threads before the for-loop to avoid creating an
    // index in each iteration
#pragma omp parallel
    {
      // Allocate an OpenMP thread-local index
      Id index;
      // Make a simple loop end condition for OpenMP
      boost::multi_array_types::index _sycl_end =
        r[Range::dimensionality - level];
      /* Distribute the iterations on the OpenMP threads. Some OpenMP
         "collapse" could be useful for small iteration space, but it
         would need some template specialization to have real contiguous
         loop nests */
#pragma omp for
      for (boost::multi_array_types::index _sycl_index = 0;
           _sycl_index < _sycl_end;
           _sycl_index++) {
        // Set the current value of the index for this dimension
        index[Range::dimensionality - level] = _sycl_index;
        // Iterate further on lower dimensions
        parallel_for_iterate<level - 1,
                             Range,
                             ParallelForFunctor,
                             Id> { r, f, index };
      }
    }
  }
```

8.9.2.8 struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >

template<typename Range, typename ParallelForFunctor, typename Id>
struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >

Stop the recursion when level reaches 0 by simply calling the kernel functor with the constructed id.

Definition at line 109 of file parallelism.hpp.

Public Member Functions

- parallel_for_iterate (Range r, ParallelForFunctor &f, Id &index)

8.9.2.8.1 Constructor & Destructor Documentation

8.9.2.8.1.1 template<typename Range , typename ParallelForFunctor , typename Id > cl::sycl::detail::parallel_for_↩ iterate< 0, Range, ParallelForFunctor, Id >::parallel_for_iterate (Range *r,* ParallelForFunctor & *f,* Id & *index*) [inline]

Definition at line 110 of file parallelism.hpp.

```
                                                                   {
    f(index);
  }
```

8.9.2.9 class cl::sycl::range

template< std::size_t dims = 1 >
class cl::sycl::range< dims >

A SYCL range defines a multi-dimensional index range that can be used to define launch parallel computation extent or buffer sizes.

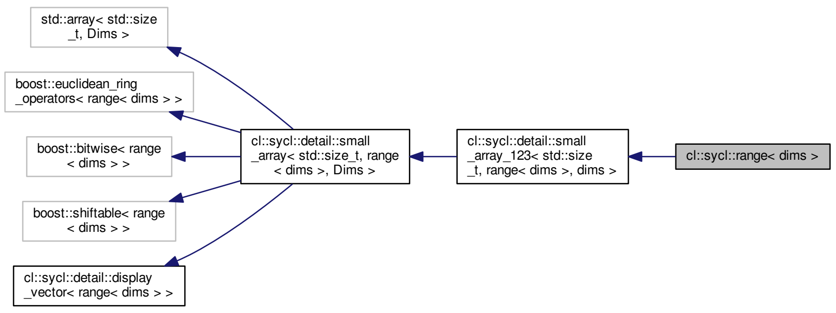
Todo use std::size_t dims instead of int dims in the specification?

Todo add to the specification this default parameter value?

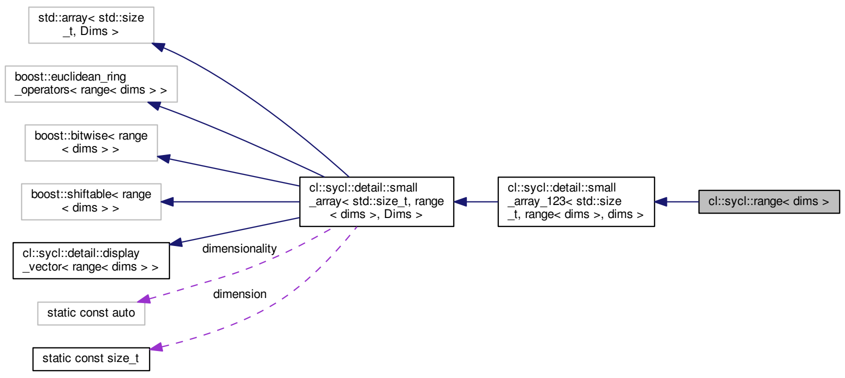
Todo add to the specification some way to specify an offset?

Definition at line 33 of file range.hpp.

Inheritance diagram for cl::sycl::range< dims >:


std::array< std::size_t, Dims >
boost::euclidean_ring_operators< range< dims > >
boost::bitwise< range< dims > >
boost::shiftable< range< dims > >
cl::sycl::detail::display_vector< range< dims > >
cl::sycl::detail::small_array< std::size_t, range< dims >, Dims >
cl::sycl::detail::small_array_123< std::size_t, range< dims >, dims >
cl::sycl::range< dims >

Collaboration diagram for cl::sycl::range< dims >:


std::array< std::size_t, Dims >
boost::euclidean_ring_operators< range< dims > >
boost::bitwise< range< dims > >
boost::shiftable< range< dims > >
cl::sycl::detail::display_vector< range< dims > >
dimensionality
static const auto
dimension
static const size_t
cl::sycl::detail::small_array< std::size_t, range< dims >, Dims >
cl::sycl::detail::small_array_123< std::size_t, range< dims >, dims >
cl::sycl::range< dims >

Public Member Functions

- size_t get_count ()

  Return the number of elements in the range.

Additional Inherited Members

8.9.2.9.1 Member Function Documentation

8.9.2.9.1.1 template<std::size_t dims = 1> size_t cl::sycl::range< dims >::get_count ( ) [inline]

Return the number of elements in the range.

Todo Give back size() its real meaning in the specification

Todo add this method to the specification

Definition at line 49 of file range.hpp.

```
                        {
    // Return the product of the sizes in each dimension
    return std::accumulate(this->cbegin(),
                           this->cend(),
                           1,
                           std::multiplies<size_t> {});
  }
```

8.9.3 Function Documentation

8.9.3.1 auto cl::sycl::make_id (id< 1 > *i*) [inline]

`#include <include/CL/sycl/id.hpp>`

Implement a make_id to construct an id<> of the right dimension with implicit conversion from an initializer list for example.

Cannot use a template on the number of dimensions because the implicit conversion would not be tried.

Definition at line 66 of file id.hpp.

```
{ return i; }
```

8.9.3.2 auto cl::sycl::make_id (id< 2 > *i*) [inline]

`#include <include/CL/sycl/id.hpp>`

Definition at line 67 of file id.hpp.

```
{ return i; }
```

8.9.3.3 auto cl::sycl::make_id (**id**< 3 > *i*) [inline]

`#include <include/CL/sycl/id.hpp>`

Definition at line 68 of file id.hpp.

```
00068 { return i; }
```

8.9.3.4 template<typename... BasicType> auto cl::sycl::make_id (BasicType... *Args*)

`#include <include/CL/sycl/id.hpp>`

Construct an id<> from a function call with arguments, like make_id(1, 2, 3)

Definition at line 74 of file id.hpp.

```
                              {
  // Call constructor directly to allow narrowing
  return id<sizeof...(Args)>(Args...);
}
```

8.9.3.5 auto cl::sycl::make_range (**range**< 1 > *r*) [inline]

`#include <include/CL/sycl/range.hpp>`

Implement a make_range to construct a range<> of the right dimension with implicit conversion from an initializer list for example.

Cannot use a template on the number of dimensions because the implicit conversion would not be tried.

Definition at line 65 of file range.hpp.

```
00065 { return r; }
```

8.9.3.6 auto cl::sycl::make_range (**range**< 2 > *r*) [inline]

`#include <include/CL/sycl/range.hpp>`

Definition at line 66 of file range.hpp.

```
00066 { return r; }
```

8.9.3.7 auto cl::sycl::make_range (**range**< 3 > *r*) [inline]

`#include <include/CL/sycl/range.hpp>`

Definition at line 67 of file range.hpp.

```
00067 { return r; }
```

8.9.3.8 template<typename... BasicType> auto cl::sycl::make_range (BasicType... *Args*)

`#include <include/CL/sycl/range.hpp>`

Construct a range<> from a function call with arguments, like make_range(1, 2, 3)

Definition at line 74 of file range.hpp.

```
                                   {
  // Call constructor directly to allow narrowing
  return range<sizeof...(Args)>(Args...);
}
```

8.9.3.9 template<std::size_t Dimensions = 1, typename ParallelForFunctor , typename Id > void cl::sycl::detail::parallel_for (range< Dimensions > *r*, ParallelForFunctor *f*, Id)

`#include <include/CL/sycl/parallelism/detail/parallelism.hpp>`

Implementation of a data parallel computation with parallelism specified at launch time by a range<>.

Kernel index is id or int.

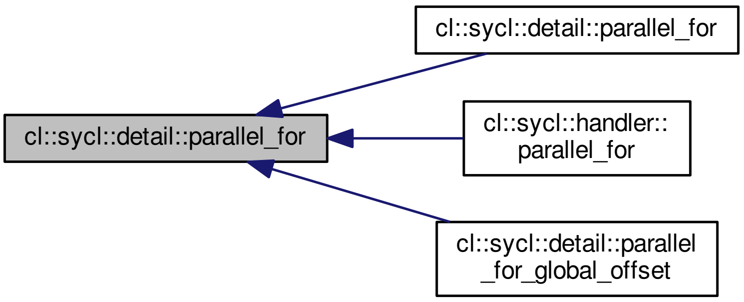
This implementation use OpenMP 3 if compiled with the right flag.

Definition at line 122 of file parallelism.hpp.

Referenced by cl::sycl::detail::parallel_for(), cl::sycl::handler::parallel_for(), and cl::sycl::detail::parallel_for_global↵
_offset().

```
                      {
#ifdef _OPENMP
  // Use OpenMP for the top loop level
  parallel_OpenMP_for_iterate<Dimensions,
                              range<Dimensions>,
                              ParallelForFunctor,
                              id<Dimensions>> { r, f };
#else
  // In a sequential execution there is only one index processed at a time
  id<Dimensions> index;
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       ParallelForFunctor,
                       id<Dimensions>> { r, f, index };
#endif
}
```

Here is the caller graph for this function:


cl::sycl::detail::parallel_for
cl::sycl::detail::parallel_for
cl::sycl::handler::
parallel_for
cl::sycl::detail::parallel
_for_global_offset

8.9.3.10 template<std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::detail::parallel_for (range< Dimensions > *r*, ParallelForFunctor *f*, item< Dimensions >)

```
#include <include/CL/sycl/parallelism/detail/parallelism.hpp>
```

Implementation of a data parallel computation with parallelism specified at launch time by a range<>.

Kernel index is item.

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

Definition at line 148 of file parallelism.hpp.

```
00150                                           {
00151   auto reconstruct_item = [&] (id<Dimensions> l) {
00152     // Reconstruct the global item
00153     item<Dimensions> index { r, l };
00154     // Call the user kernel with the item<> instead of the id<>
00155     f(index);
00156   };
00157 #ifdef _OPENMP
00158   // Use OpenMP for the top loop level
00159   parallel_OpenMP_for_iterate<Dimensions,
00160                               range<Dimensions>,
00161                               decltype(reconstruct_item),
00162                               id<Dimensions>> { r, reconstruct_item };
00163 #else
00164   // In a sequential execution there is only one index processed at a time
00165   id<Dimensions> index;
00166   parallel_for_iterate<Dimensions,
00167                        range<Dimensions>,
00168                        decltype(reconstruct_item),
00169                        id<Dimensions>> { r, reconstruct_item, index };
00170 #endif
00171 }
```

8.9.3.11 template<std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::detail::parallel_for (range< Dimensions > *r*, ParallelForFunctor *f*)

```
#include <include/CL/sycl/parallelism/detail/parallelism.hpp>
```

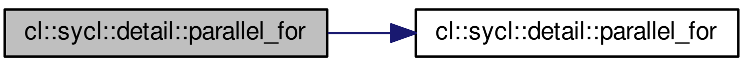
Calls the appropriate ternary parallel_for overload based on the index type of the kernel function object f.

Definition at line 179 of file parallelism.hpp.

References cl::sycl::detail::parallel_for().

```
00179                                                                     {
00180   using mf_t  = decltype(std::mem_fn(&ParallelForFunctor::operator()));
00181   using arg_t = typename mf_t::second_argument_type;
00182   parallel_for(r,f,arg_t{});
00183 }
```

Here is the call graph for this function:


cl::sycl::detail::parallel_for
cl::sycl::detail::parallel_for

8.9.3.12 template< std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::detail::parallel_for (nd_range< Dimensions > *r*, ParallelForFunctor *f*)

```
#include <include/CL/sycl/parallelism/detail/parallelism.hpp>
```

Implement a variation of parallel_for to take into account a nd_range<>

Todo Add an OpenMP implementation

Todo Deal with incomplete work-groups

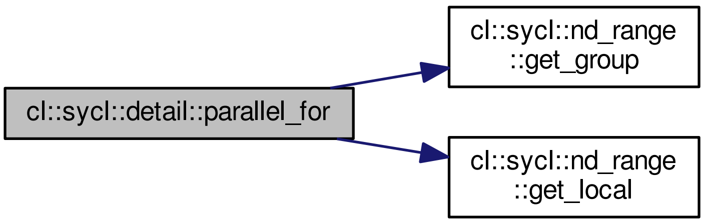
Todo Implement with parallel_for_workgroup()/parallel_for_workitem()

Definition at line 214 of file parallelism.hpp.

References cl::sycl::nd_range< dims >::get_group(), and cl::sycl::nd_range< dims >::get_local().

```
                                          {
  // In a sequential execution there is only one index processed at a time
  nd_item<Dimensions> index { r };
  // To iterate on the work-group
  id<Dimensions> group;
  range<Dimensions> group_range = r.get_group();
  // To iterate on the local work-item
  id<Dimensions> local;

  range<Dimensions> local_range = r.get_local();

  // Reconstruct the nd_item from its group and local id
  auto reconstruct_item = [&] (id<Dimensions> l) {
    //local.display();
    // Reconstruct the global nd_item
    index.set_local(local);
    // Upgrade local_range to an id<> so that we can * with the group (an id<>)
    index.set_global(local + id<Dimensions>(local_range)*group);
    // Call the user kernel at last
    f(index);
  };

  /* To recycle the parallel_for on range<>, wrap the ParallelForFunctor f
     into another functor that iterates inside the work-group and then
     calls f */
  auto iterate_in_work_group = [&] (id<Dimensions> g) {
    //group.display();
    // Then iterate on the local work-groups
    parallel_for_iterate<Dimensions,
                         range<Dimensions>,
                         decltype(reconstruct_item),
                         id<Dimensions>> { local_range,
                                           reconstruct_item,
                                           local };
  };

  // First iterate on all the work-groups
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       decltype(iterate_in_work_group),
                       id<Dimensions>> { group_range,
                                         iterate_in_work_group,
      group };
}
```

Here is the call graph for this function:


cl::sycl::detail::parallel_for
cl::sycl::nd_range ::get_group
cl::sycl::nd_range ::get_local

8.9.3.13 template< std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::detail::parallel_for_global_offset (range< Dimensions > *global_size,* id< Dimensions > *offset,* ParallelForFunctor *f*)

```
#include <include/CL/sycl/parallelism/detail/parallelism.hpp>
```

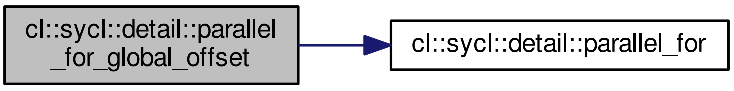
Implementation of parallel_for with a range<> and an offset.

Definition at line 188 of file parallelism.hpp.

References cl::sycl::detail::parallel_for().

```
                                                    {
  // Reconstruct the item from its id<> and its offset
  auto reconstruct_item = [&] (id<Dimensions> l) {
    // Reconstruct the global item
    item<Dimensions> index { global_size, l + offset, offset };
    // Call the user kernel with the item<> instead of the id<>
    f(index);
  };

  // First iterate on all the work-groups
  parallel_for(global_size, reconstruct_item);
}
```

Here is the call graph for this function:


cl::sycl::detail::parallel _for_global_offset
cl::sycl::detail::parallel_for

8.9.3.14 template< std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::parallel_for_work_item (const group< Dimensions > & *g*, ParallelForFunctor *f*)

`#include <include/CL/sycl/parallelism.hpp>`

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

Todo To be implemented

Loop on the work-items inside a work-group

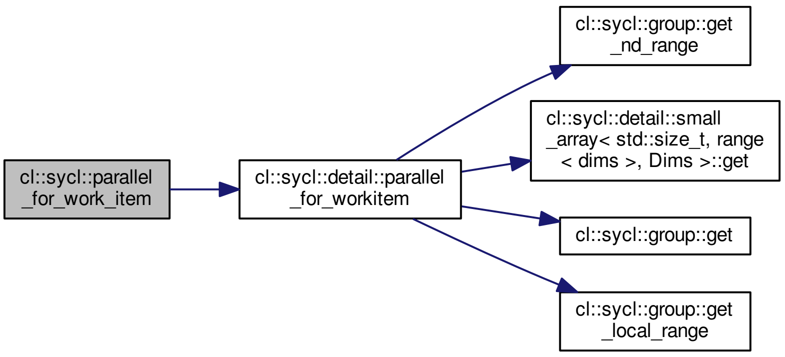
Todo Deprecate this function in the specification to use instead the group method

Definition at line 38 of file parallelism.hpp.

References cl::sycl::detail::parallel_for_workitem().

```
                                                    {
    detail::parallel_for_workitem(g, f);
  }
```

Here is the call graph for this function:


cl::sycl::parallel_for_work_item
cl::sycl::detail::parallel_for_workitem
cl::sycl::group::get_nd_range
cl::sycl::detail::small_array< std::size_t, range< dims >, Dims >::get
cl::sycl::group::get
cl::sycl::group::get_local_range

8.9.3.15 template< std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::detail::parallel_for_workgroup (nd_range< Dimensions > *r*, ParallelForFunctor *f*)

`#include <include/CL/sycl/parallelism/detail/parallelism.hpp>`

Implement the loop on the work-groups.

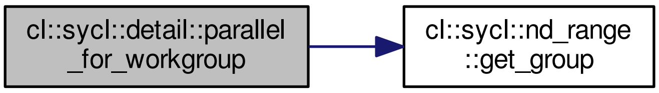
Definition at line 263 of file parallelism.hpp.

References cl::sycl::nd_range< dims >::get_group().

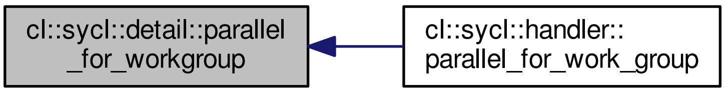
Referenced by cl::sycl::handler::parallel_for_work_group().

```
                                                {
  // In a sequential execution there is only one index processed at a time
  group<Dimensions> g { r };

  // First iterate on all the work-groups
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       ParallelForFunctor,
                       group<Dimensions>> {
    r.get_group(),
    f,
    g };
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.3.16 template< std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::detail::parallel_for_workitem (const group< Dimensions > & *g*, ParallelForFunctor *f*)

`#include <include/CL/sycl/group.hpp>`

Implement the loop on the work-items inside a work-group.

Todo Better type the functor

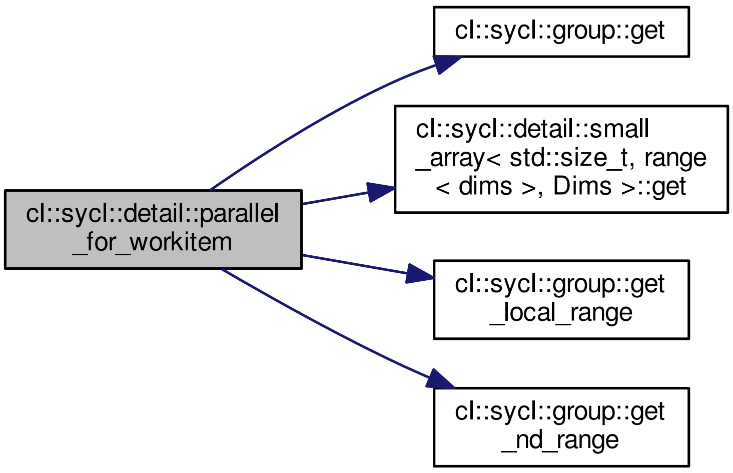
Definition at line 284 of file parallelism.hpp.

References cl::sycl::group< dims >::get(), cl::sycl::detail::small_array< std::size_t, range< dims >, Dims >::get(), cl::sycl::group< dims >::get_local_range(), and cl::sycl::group< dims >::get_nd_range().

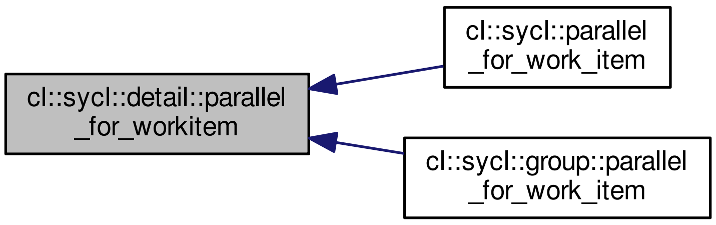
Referenced by cl::sycl::parallel_for_work_item(), and cl::sycl::group< dims >::parallel_for_work_item().

```
                                                    {
#if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
  /* To implement barriers With OpenMP, one thread is created for each
     work-item in the group and thus an OpenMP barrier has the same effect
     of an OpenCL barrier executed by the work-items in a workgroup

     The issue is that the parallel_for_workitem() execution is slow even
     when nd_item::barrier() is not used
  */


  // Is the above comment true anymore ?
  // Maybe the following will be enough
  // #ifdef _OPENMP

  // With OMP, one task is created for each work-item in the group

  range<Dimensions> l_r = g.get_nd_range().get_local();
  int tot = l_r.get(0);
  for (int i = 1; i < (int) Dimensions; ++i){
    tot *= l_r.get(i);
  }
#pragma omp parallel
  {
#pragma omp single nowait
    {
      for (int th_id = 0; th_id < tot; ++th_id) {
#pragma omp task firstprivate(th_id)
        {
          nd_item<Dimensions> index { g.get_nd_range() };
          id<Dimensions> local; // to initialize correctly

          if (Dimensions ==1) {
            local[0] = th_id;
          } else if (Dimensions == 2) {
            local[0] = th_id / l_r.get(1);
            local[1] = th_id - local[0]*l_r.get(1);
          } else if (Dimensions == 3) {
            int tmp = l_r.get(1)*l_r.get(2);
            local[0] = th_id / tmp;
            local[1] = (th_id - local[0]*tmp) / l_r.get(1);
            local[2] = th_id - local[0]*tmp - local[1]*l_r.get(1);
          }
          index.set_local(local);
          index.set_global(local + id<Dimensions>(l_r)*g.get());
          f(index);
        }
      }
    }
  }
#else
  // In a sequential execution there is only one index processed at a time
  nd_item<Dimensions> index { g.get_nd_range() };
  // To iterate on the local work-item
  id<Dimensions> local;

  // Reconstruct the nd_item from its group and local id
  auto reconstruct_item = [&] (id<Dimensions> l) {
    //local.display();
    //l.display();
    // Reconstruct the global nd_item
    index.set_local(local);
    // \todo Some strength reduction here
    index.set_global(local + id<Dimensions>(g.get_local_range())*g.get());
    // Call the user kernel at last
    f(index);
  };

  // Then iterate on all the work-items of the work-group
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       decltype(reconstruct_item),
                       id<Dimensions>> {
    g.get_local_range(),
    reconstruct_item,
    local };
#endif
}
```

Here is the call graph for this function:


cl::sycl::group::get
cl::sycl::detail::small_array< std::size_t, range< dims >, Dims >::get
cl::sycl::detail::parallel_for_workitem
cl::sycl::group::get_local_range
cl::sycl::group::get_nd_range

Here is the caller graph for this function:


cl::sycl::parallel_for_work_item
cl::sycl::detail::parallel_for_workitem
cl::sycl::group::parallel_for_work_item

8.10 Vector types in SYCL

Classes

- class cl::sycl::vec< DataType, NumElements >

  Small OpenCL vector class. More...

Macros

- #define TRISYCL_DEFINE_VEC_TYPE_SIZE(type, size, actual_type) using type##size = vec<actual_type, size>;

  A macro to define type alias, such as for type=uchar, size=4 and real_type=unsigned char, uchar4 is equivalent to vec<float, 4>
- #define TRISYCL_DEFINE_VEC_TYPE(type, actual_type)

  Declare the vector types of a type for all the sizes.

8.10.1 Detailed Description

8.10.2 Class Documentation

8.10.2.1 class cl::sycl::vec

template<typename DataType, size_t NumElements>
class cl::sycl::vec< DataType, NumElements >

Small OpenCL vector class.

Todo add [] operator

Todo add iterators on elements, with begin() and end()

Todo having vec<> sub-classing array<> instead would solve the previous issues

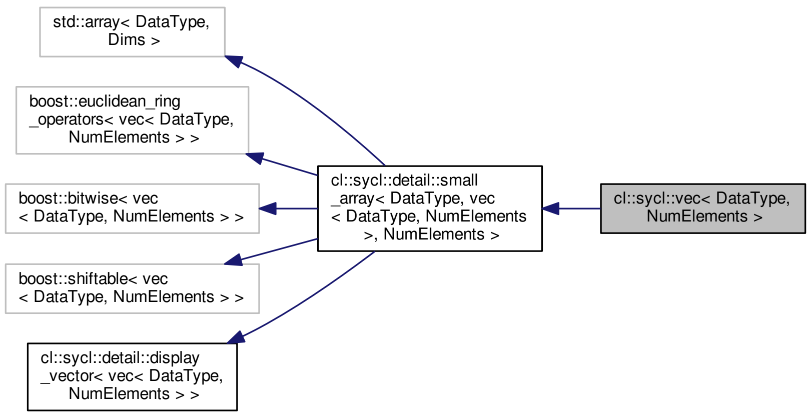
Todo move the implementation elsewhere

Todo simplify the helpers by removing some template types since there are now inside the vec<> class.

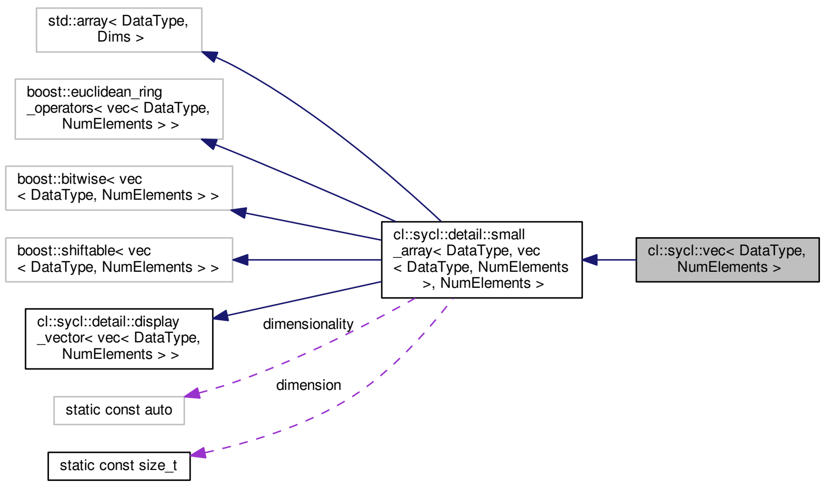
Todo rename in the specification element_type to value_type

Definition at line 42 of file vec.hpp.

Inheritance diagram for cl::sycl::vec< DataType, NumElements >:


std::array< DataType, Dims >
boost::euclidean_ring _operators< vec< DataType, NumElements > >
boost::bitwise< vec < DataType, NumElements > >
boost::shiftable< vec < DataType, NumElements > >
cl::sycl::detail::display _vector< vec< DataType, NumElements > >
cl::sycl::detail::small _array< DataType, vec < DataType, NumElements >, NumElements >
cl::sycl::vec< DataType, NumElements >

Collaboration diagram for cl::sycl::vec< DataType, NumElements >:


std::array< DataType, Dims >
boost::euclidean_ring _operators< vec< DataType, NumElements > >
boost::bitwise< vec < DataType, NumElements > >
boost::shiftable< vec < DataType, NumElements > >
cl::sycl::detail::display _vector< vec< DataType, NumElements > >
static const auto
static const size_t
dimensionality
dimension
cl::sycl::detail::small _array< DataType, vec < DataType, NumElements >, NumElements >
cl::sycl::vec< DataType, NumElements >

Public Member Functions

- template<typename... Types>
  vec (const Types...args)

  Construct a vec from anything from a scalar (to initialize all the elements with this value) up to an aggregate of scalar and vector types (in this case the total number of elements must match the size of the vector)
- vec ()=default

  Use classical constructors too.

Private Types

- using basic_type = typename detail::small_array< DataType, vec< DataType, NumElements >, Num↩ Elements >

Static Private Member Functions

- template<typename V , typename Element , size_t s>
  static auto flatten (const vec< Element, s > i)

  Flattening helper that does not change scalar values but flatten a vec<T, n> v into a tuple<T, T,..., T>{ v[0], v[1],..., v[n-1] }.
- template<typename V , typename Type >
  static auto flatten (const Type i)

  If we do not have a vector, just forward it as a tuple up to the final initialization.
- template<typename V , typename... Types>
  static auto flatten_to_tuple (const Types...i)

  Take some initializer values and apply flattening on each value.

Additional Inherited Members

8.10.2.1.1 Member Typedef Documentation

8.10.2.1.1.1 template<typename DataType, size_t NumElements> using **cl::sycl::vec**< DataType, NumElements >::**basic_type** = typename **detail::small_array**<DataType, **vec**<DataType, NumElements>, NumElements> [private]

Definition at line 47 of file vec.hpp.

8.10.2.1.2 Constructor & Destructor Documentation

8.10.2.1.2.1 template<typename DataType, size_t NumElements> template<typename... Types> **cl::sycl::vec**< DataType, NumElements >::**vec** (const Types... *args*) [inline]

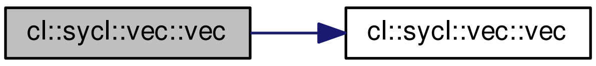
Construct a vec from anything from a scalar (to initialize all the elements with this value) up to an aggregate of scalar and vector types (in this case the total number of elements must match the size of the vector)

Definition at line 57 of file vec.hpp.

References cl::sycl::vec< DataType, NumElements >::vec().

```
00058     : basic_type { detail::expand<vec>(flatten_to_tuple<vec>(args...)) } { }
```

Here is the call graph for this function:

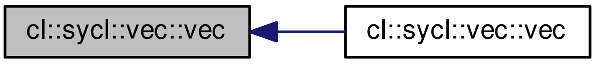


8.10.2.1.2.2 template<typename DataType, size_t NumElements> cl::sycl::vec< DataType, NumElements >::vec ( ) [default]

Use classical constructors too.

Referenced by cl::sycl::vec< DataType, NumElements >::vec().

Here is the caller graph for this function:



8.10.2.1.3 Member Function Documentation

8.10.2.1.3.1 template<typename DataType, size_t NumElements> template<typename V , typename Element , size_t s> static auto cl::sycl::vec< DataType, NumElements >::flatten (const vec< Element, s > *i*) [inline], [static], [private]

Flattening helper that does not change scalar values but flatten a vec<T, n> v into a tuple<T, T,..., T>{ v[0], v[1],..., v[n-1] }.

If we have a vector, just forward its array content since an array has also a tuple interface :-) (23.3.2.9 Tuple interface to class template array [array.tuple])

Definition at line 78 of file vec.hpp.

```
                                            {
    static_assert(s <= V::dimension,
                  "The element i will not fit in the vector");
    return static_cast<std::array<Element, s>>(i);
  }
```

8.10.2.1.3.2 template<typename DataType, size_t NumElements> template<typename V , typename Type > static auto cl::sycl::vec< DataType, NumElements >::flatten (const Type *i*) [inline], [static], [private]

If we do not have a vector, just forward it as a tuple up to the final initialization.

Returns

typically tuple<double>{ 2.4 } from 2.4 input for example

Definition at line 91 of file vec.hpp.

```
                                   {
    return std::make_tuple(i);
  }
```

8.10.2.1.3.3 template<typename DataType, size_t NumElements> template<typename V , typename... Types> static auto cl::sycl::vec< DataType, NumElements >::flatten_to_tuple (const Types... *i*) [inline], [static], [private]

Take some initializer values and apply flattening on each value.

Returns

a tuple of scalar initializer values

Definition at line 101 of file vec.hpp.

```
                                              {
    // Concatenate the tuples returned by each flattening
    return std::tuple_cat(flatten<V>(i)...);
  }
```

8.10.3 Macro Definition Documentation

8.10.3.1 #define TRISYCL_DEFINE_VEC_TYPE(*type,* *actual_type*)

```
#include <include/CL/sycl/vec.hpp>
```

Value:

```
TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 1, actual_type)
      \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 2, actual_type)
      \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 3, actual_type)
      \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 4, actual_type)
      \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 8, actual_type)
      \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 16, actual_type)
```

Declare the vector types of a type for all the sizes.

Definition at line 162 of file vec.hpp.

8.10.3.2 #define TRISYCL_DEFINE_VEC_TYPE_SIZE(*type,* *size,* *actual_type*) using type##size = vec<actual_type, size>;

```
#include <include/CL/sycl/vec.hpp>
```

A macro to define type alias, such as for type=uchar, size=4 and real_type=unsigned char, uchar4 is equivalent to vec<float, 4>

Definition at line 158 of file vec.hpp.

Chapter 9

Namespace Documentation

9.1 cl Namespace Reference

The vector type to be used as SYCL vector.

Namespaces

- sycl

9.1.1 Detailed Description

The vector type to be used as SYCL vector.

The weak pointer type to be used as SYCL weak pointer.

The shared pointer type to be used as SYCL shared pointer.

The unique pointer type to be used as SYCL unique pointer.

The mutex type to be used as SYCL mutex.

The functional type to be used as SYCL function.

The string type to be used as SYCL string.

9.2 cl::sycl Namespace Reference

Namespaces

- access

    Describe the type of access by kernels.
- detail
- info
- trisycl

Classes

- class accessor

  The accessor abstracts the way buffer or pipe data are accessed inside a kernel in a multidimensional variable length array way. More...
- class accessor< DataType, 1, AccessMode, access::target::blocking_pipe >

  The pipe accessor abstracts the way pipe data are accessed inside a kernel. More...
- class accessor< DataType, 1, AccessMode, access::target::pipe >

  The pipe accessor abstracts the way pipe data are accessed inside a kernel. More...
- class accessor_error

  Error regarding the cl::sycl::accessor objects defined. More...
- struct async_exception

  An error stored in an exception_list for asynchronous errors. More...
- class buffer

  <T, Dimensions, Mode, Target>up data Data access and storage in SYCL
- class cl_exception

  Returns the OpenCL error code encapsulated in the exception. More...
- class compile_program_error

  Error while compiling the SYCL kernel to a SYCL device. More...
- class context

  SYCL context. More...
- class device

  SYCL device. More...
- class device_error

  The SYCL device will trigger this exception on error. More...
- class device_selector

  The SYCL heuristics to select a device. More...
- class device_type_selector

  A device selector by device_type. More...
- class device_typename_selector

  Select a device by template device_type parameter. More...
- struct 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...
- class event
- class event_error

  Error regarding associated cl::sycl::event objects. More...
- class exception

  Encapsulate a SYCL error information. More...
- struct exception_list

  Exception list to store several exceptions. More...
- class feature_not_supported

  Exception thrown when an optional feature or extension is used in a kernel but its not available on the device the SYCL kernel is being enqueued on. More...
- struct group

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

  Command group handler class. More...
- class id

  Define a multi-dimensional index, used for example to locate a work item. More...
- struct image
- class invalid_object_error

Error regarding any memory objects being used inside the kernel. More...

- class invalid_parameter_error

    Error regarding parameters to the SYCL kernel, it may apply to any captured parameters to the kernel lambda. More...

- class item

    A SYCL item stores information on a work-item with some more context such as the definition range and offset. More...

- class kernel

    SYCL kernel. More...

- class kernel_error

    Error that occurred before or while enqueuing the SYCL kernel. More...

- class link_program_error

    Error while linking the SYCL kernel to a SYCL device. More...

- class memory_allocation_error

    Error on memory allocation on the SYCL device for a SYCL kernel. More...

- struct nd_item

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

- struct nd_range

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

- class nd_range_error

    Error regarding the cl::sycl::nd_range specified for the SYCL kernel. More...

- class non_cl_error

    Exception for an OpenCL operation requested in a non OpenCL area. More...

- class pipe

    A SYCL pipe. More...

- class pipe_error

    A failing pipe error will trigger this exception on error. More...

- struct pipe_reservation

    The pipe reservation station allows to reserve an array-like view inside the pipe for ordered race-free access from various work-items for example. More...

- class platform

    Abstract the OpenCL platform. More...

- class platform_error

    The SYCL platform will trigger this exception on error. More...

- class profiling_error

    The SYCL runtime will trigger this error if there is an error when profiling info is enabled. More...

- class queue

    SYCL queue, similar to the OpenCL queue concept. More...

- class range

    A SYCL range defines a multi-dimensional index range that can be used to define launch parallel computation extent or buffer sizes. More...

- class runtime_error
- class static_pipe

    A SYCL static-scoped pipe equivalent to an OpenCL program-scoped pipe. More...

- class vec

    Small OpenCL vector class. More...

Typedefs

- template<typename T >
  using constant = detail::addr_space< T, constant_address_space >

  Declare a variable to be in the OpenCL constant address space.
- template<typename T >
  using generic = detail::addr_space< T, generic_address_space >

  Declare a variable to be in the OpenCL 2 generic address space.
- template<typename T >
  using global = detail::addr_space< T, global_address_space >

  Declare a variable to be in the OpenCL global address space.
- template<typename T >
  using local = detail::addr_space< T, local_address_space >

  Declare a variable to be in the OpenCL local address space.
- template<typename T >
  using priv = detail::addr_space< T, private_address_space >

  Declare a variable to be in the OpenCL private address space.
- template<typename Pointer , address_space AS>
  using multi_ptr = detail::address_space_ptr< Pointer, AS >

  A pointer that can be statically associated to any address-space.
- template<typename T >
  using buffer_allocator = std::allocator< T >

  The default buffer allocator used by the runtime, when no allocator is defined by the user.
- template<class T , class Alloc = std::allocator<T>>
  using vector_class = std::vector< T, Alloc >
- using string_class = std::string
- template<class R , class... ArgTypes>
  using function_class = std::function< R(ArgTypes...)>
- using mutex_class = std::mutex
- template<class T , class D = std::default_delete<T>>
  using unique_ptr_class = std::unique_ptr< T[ ], D >
- template<class T >
  using shared_ptr_class = std::shared_ptr< T >
- template<class T >
  using weak_ptr_class = std::weak_ptr< T >
- using default_selector = device_typename_selector< info::device_type::defaults >

  Devices selected by heuristics of the system.
- using gpu_selector = device_typename_selector< info::device_type::gpu >

  Select devices according to device type info::device::device_type::gpu from all the available OpenCL devices.
- using cpu_selector = device_typename_selector< info::device_type::cpu >

  Select devices according to device type info::device::device_type::cpu from all the available devices and heuristics.
- using host_selector = device_typename_selector< info::device_type::host >

  Selects the SYCL host CPU device that does not require an OpenCL runtime.
- using exception_ptr = std::exception_ptr

  A shared pointer to an exception as in C++ specification.
- using async_handler = function_class< void, exception_list >

Enumerations

- enum address_space {
  constant_address_space, generic_address_space, global_address_space, local_address_space,
  private_address_space }

  Enumerate the different OpenCL 2 address spaces.

Functions

- template< typename Accessor >
  static auto & get_pipe_detail (Accessor &a)

  Top-level function to break circular dependencies on the the types to get the pipe implementation.
- template<typename T , address_space AS>
  multi_ptr< T, AS > make_multi (multi_ptr< T, AS > pointer)

  Construct a cl::sycl::multi_ptr<> with the right type.
- auto make_id (id< 1 > i)

  Implement a make_id to construct an id<> of the right dimension with implicit conversion from an initializer list for example.
- auto make_id (id< 2 > i)
- auto make_id (id< 3 > i)
- template<typename... BasicType>
  auto make_id (BasicType...Args)

  Construct an id<> from a function call with arguments, like make_id(1, 2, 3)
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_work_item (const group< Dimensions > &g, ParallelForFunctor f)

  SYCL parallel_for version that allows a Program object to be specified.
- auto make_range (range< 1 > r)

  Implement a make_range to construct a range<> of the right dimension with implicit conversion from an initializer list for example.
- auto make_range (range< 2 > r)
- auto make_range (range< 3 > r)
- template<typename... BasicType>
  auto make_range (BasicType...Args)

  Construct a range<> from a function call with arguments, like make_range(1, 2, 3)

9.2.1 Typedef Documentation

9.2.1.1 template<class R , class... ArgTypes> using **cl::sycl::function_class** = typedef std::function<R(ArgTypes...)>

Definition at line 55 of file default_classes.hpp.

9.2.1.2 using **cl::sycl::mutex_class** = typedef std::mutex

Definition at line 69 of file default_classes.hpp.

9.2.1.3 template<class T > using **cl::sycl::shared_ptr_class** = typedef std::shared_ptr<T>

Definition at line 99 of file default_classes.hpp.

9.2.1.4 using **cl::sycl::string_class** = typedef std::string

Definition at line 40 of file default_classes.hpp.

9.2.1.5 template<class T , class D = std::default_delete<T>> using **cl::sycl::unique_ptr_class** = typedef std::unique_ptr<T[ ], D>

Definition at line 84 of file default_classes.hpp.

9.2.1.6 template<class T , class Alloc = std::allocator<T>> using **cl::sycl::vector_class** = typedef std::vector<T, Alloc>

Definition at line 26 of file default_classes.hpp.

9.2.1.7 template<class T > using **cl::sycl::weak_ptr_class** = typedef std::weak_ptr<T>

Definition at line 114 of file default_classes.hpp.

9.3 cl::sycl::access Namespace Reference

Describe the type of access by kernels.

Enumerations

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

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

  The target enumeration describes the type of object to be accessed via the accessor.
- enum fence_space : char { fence_space::local_space, fence_space::global_space, fence_space::global_↩
  and_local }

  Precise the address space a barrier needs to act on.

9.3.1 Detailed Description

Describe the type of access by kernels.

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

9.3.2 Enumeration Type Documentation

9.3.2.1 enum cl::sycl::access::fence_space : char [strong]

Precise the address space a barrier needs to act on.

Enumerator

local_space

global_space

global_and_local

Definition at line 63 of file access.hpp.

```
                              : char {
     local_space,
     global_space,
     global_and_local
   };
```

9.3.2.2 enum cl::sycl::access::mode [strong]

This describes the type of the access mode to be used via accessor.

Enumerator

read Read-only access. Insist on the fact that read_write != read + write

write Write-only access, but previous content *not* discarded.

read_write Read and write access.

discard_write Write-only access and previous content discarded.

discard_read_write Read and write access and previous content discarded.

atomic Atomic access.

Definition at line 33 of file access.hpp.

```
                     {
     read = 42, /**< Read-only access. Insist on the fact that
                     read_write != read + write */
     write, ///< Write-only access, but previous content *not* discarded
     read_write, ///< Read and write access
     discard_write, ///< Write-only access and previous content discarded
     discard_read_write, /**< Read and write access and previous
                              content discarded*/
     atomic ///< Atomic access
   };
```

9.3.2.3 enum cl::sycl::access::target [strong]

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

Enumerator

global_buffer

constant_buffer

local

image

host_buffer

host_image

image_array

pipe

blocking_pipe

Definition at line 48 of file access.hpp.

```
                    {
  global_buffer = 2014, //< Just pick a random number...
  constant_buffer,
  local,
  image,
  host_buffer,
  host_image,
  image_array,
  pipe,
  blocking_pipe
};
```

9.4 cl::sycl::detail Namespace Reference

Classes

- class accessor

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

  Implementation of an array variable with an OpenCL address space. More...
- struct address_space_base

  Implementation of the base infrastructure to wrap something in an OpenCL address space. More...
- struct address_space_fundamental

  Implementation of a fundamental type with an OpenCL address space. More...
- struct address_space_object

  Implementation of an object type with an OpenCL address space. More...
- struct address_space_ptr

  Implementation for an OpenCL address space pointer. More...
- struct address_space_variable

  Implementation of a variable with an OpenCL address space. More...
- class buffer

  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 buffer_base

  Factorize some template independent buffer aspects in a base class.
- class buffer_waiter

  A helper class to wait for the final buffer destruction if the conditions for blocking are met. More...
- class cache

  A simple thread safe cache mechanism to cache std::shared_ptr of values indexed by keys.
- struct debug

  Class used to trace the construction, copy-construction, move-construction and destruction of classes that inherit from it. More...
- class device

  An abstract class representing various models of SYCL devices. More...
- struct display_vector

  Class used to display a vector-like type of classes that inherit from it. More...
- struct expand_to_vector

  Allows optional expansion of a 1-element tuple to a V::dimension tuple to replicate scalar values in vector initialization. More...
- struct expand_to_vector< V, Tuple, true >

  Specialization in the case we ask for expansion. More...
- class host_device

  SYCL host device.
- class host_platform

  SYCL host platform. More...
- class host_queue

  Some implementation details about the SYCL queue.
- class kernel

  Abstract SYCL kernel. More...
- class opencl_device

  SYCL OpenCL device.
- class opencl_kernel

  An abstraction of the OpenCL kernel.
- class opencl_platform

  SYCL OpenCL platform. More...
- class opencl_queue

  Some implementation details about the SYCL queue.
- struct opencl_type

  Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device. More...
- struct opencl_type< T, constant_address_space >

  Add an attribute for __constant address space. More...
- struct opencl_type< T, generic_address_space >

  Add an attribute for __generic address space. More...
- struct opencl_type< T, global_address_space >

  Add an attribute for __global address space. More...
- struct opencl_type< T, local_address_space >

  Add an attribute for __local address space. More...
- struct opencl_type< T, private_address_space >

  Add an attribute for __private address space. More...
- struct parallel_for_iterate

  A recursive multi-dimensional iterator that ends up calling f. More...
- struct parallel_for_iterate< 0, Range, ParallelForFunctor, Id >

  Stop the recursion when level reaches 0 by simply calling the kernel functor with the constructed id. More...
- struct parallel_OpenMP_for_iterate

A top-level recursive multi-dimensional iterator variant using OpenMP. More...

- class pipe

  Implement a pipe object. More...
- class pipe_accessor

  The accessor abstracts the way pipe data are accessed inside a kernel. More...
- class pipe_reservation

  The implementation of the pipe reservation station. More...
- class platform

  An abstract class representing various models of SYCL platforms. More...
- struct queue

  Some implementation details about the SYCL queue.
- struct reserve_id

  A private description of a reservation station. More...
- struct shared_ptr_implementation

  Provide an implementation as shared_ptr with total ordering and hashing to be used with algorithms and in (un)ordered containers.
- struct singleton

  Provide a singleton factory.
- struct small_array

  Define a multi-dimensional index, used for example to locate a work item or a buffer element. More...
- struct small_array_123

  A small array of 1, 2 or 3 elements with the implicit constructors. More...
- struct small_array_123< BasicType, FinalType, 1 >

  Use some specializations so that some function overloads can be determined according to some implicit constructors and to have an implicit conversion from/to BasicType (such as an int typically) if dims = 1. More...
- struct small_array_123< BasicType, FinalType, 2 >
- struct small_array_123< BasicType, FinalType, 3 >
- struct task

  The abstraction to represent SYCL tasks executing inside command_group.

Typedefs

- template<typename T , address_space AS>
  using addr_space = typename std::conditional< std::is_pointer< T >::value, address_space_ptr< T, AS >, typename std::conditional< std::is_class< T >::value, address_space_object< T, AS >, typename std↩ ::conditional< std::is_array< T >::value, address_space_array< T, AS >, address_space_fundamental< T, AS > >::type >::type >::type

  Dispatch the address space implementation according to the requested type.

Functions

- template<typename BufferDetail >
  static std::shared_ptr< detail::task > buffer_add_to_task (BufferDetail buf, handler ∗command_group_↩ handler, bool is_write_mode)

  Proxy function to avoid some circular type recursion.
- static std::shared_ptr< detail::task > add_buffer_to_task (handler ∗command_group_handler, std::shared↩ _ptr< detail::buffer_base > b, bool is_write_mode)
- template<typename T , std::size_t Dimensions = 1>
  auto waiter (detail::buffer< T, Dimensions > ∗b)

  Helper function to create a new buffer_waiter.

- template<typename V , typename Tuple , size_t... Is>
  std::array< typename V::element_type, V::dimension > tuple_to_array_iterate (Tuple t, std::index_↩ sequence< Is... >)

  Helper to construct an array from initializer elements provided as a tuple.
- template<typename V , typename Tuple >
  auto tuple_to_array (Tuple t)

  Construct an array from initializer elements provided as a tuple.
- template<typename V , typename Tuple >
  auto expand (Tuple t)

  Create the array data of V from a tuple of initializer.
- template<typename KernelName , typename Functor >
  auto trace_kernel (const Functor &f)

  Wrap a kernel functor in some tracing messages to have start/stop information when TRISYCL_TRACE_KERNEL macro is defined.
- template<typename Range , typename Id >
  size_t constexpr linear_id (Range range, Id id, Id offset={})

  Compute a linearized array access used in the OpenCL 2 world.
- void unimplemented ()

  Display an "unimplemented" message.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_workitem (const group< Dimensions > &g, ParallelForFunctor f)

  Implement the loop on the work-items inside a work-group.
- static std::shared_ptr< detail::task > add_buffer_to_task (handler *command_group_handler, std::shared↩ _ptr< detail::buffer_base > b, bool is_write_mode)

  Register a buffer as used by a task.
- detail::cache< cl_kernel, detail::opencl_kernel > opencl_kernel::cache __attribute__ ((weak))
- template<std::size_t Dimensions = 1, typename ParallelForFunctor , typename Id >
  void parallel_for (range< Dimensions > r, ParallelForFunctor f, Id)

  Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for (range< Dimensions > r, ParallelForFunctor f, item< Dimensions >)

  Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for (range< Dimensions > r, ParallelForFunctor f)

  Calls the appropriate ternary parallel_for overload based on the index type of the kernel function object f.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_global_offset (range< Dimensions > global_size, id< Dimensions > offset, ParallelFor↩ Functor f)

  Implementation of parallel_for with a range<> and an offset.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement a variation of parallel_for to take into account a nd_range<>
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement the loop on the work-groups.

Variables

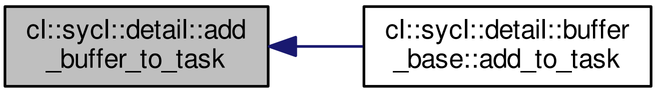
- detail::cache< cl_device_id, detail::opencl_device > opencl_device::cache __attribute__ ((weak))

9.4.1 Function Documentation

9.4.1.1 static std::shared_ptr<detail::task> cl::sycl::detail::add_buffer_to_task (handler * *command_group_handler*, std::shared_ptr< detail::buffer_base > *b*, bool *is_write_mode*) [inline], [static]

Referenced by cl::sycl::detail::buffer_base::add_to_task().

Here is the caller graph for this function:



9.4.1.2 static std::shared_ptr<detail::task> cl::sycl::detail::add_buffer_to_task (handler * *command_group_handler*, std::shared_ptr< detail::buffer_base > *b*, bool *is_write_mode*) [static]

Register a buffer as used by a task.

This is a proxy function to avoid complicated type recursion.

Definition at line 394 of file handler.hpp.

References cl::sycl::handler::task.

```
                                        {
  command_group_handler->task->add_buffer(b, is_write_mode);
  return command_group_handler->task;
}
```

9.5 cl::sycl::info Namespace Reference

Classes

- struct param_traits

  Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value)

Typedefs

- using gl_context_interop = bool
- using device_fp_config = unsigned int
- using device_exec_capabilities = unsigned int
- using device_queue_properties = unsigned int
- using queue_profiling = bool

Enumerations

- enum context : int { context::reference_count, context::num_devices, context::gl_interop }

  Context information descriptors.
- enum device_type : unsigned int {
  device_type::cpu, device_type::gpu, device_type::accelerator, device_type::custom,
  device_type::defaults, device_type::host, device_type::opencl, device_type::all }

  Type of devices.
- enum device : int {
  device::device_type, device::vendor_id, device::max_compute_units, device::max_work_item_dimensions,
  device::max_work_item_sizes, device::max_work_group_size, device::preferred_vector_width_char,
  device::preferred_vector_width_short,
  device::preferred_vector_width_int, device::preferred_vector_width_long_long, device::preferred_vector_↩
  width_float, device::preferred_vector_width_double,
  device::preferred_vector_width_half, device::native_vector_witdth_char, device::native_vector_witdth_short,
  device::native_vector_witdth_int,
  device::native_vector_witdth_long_long, device::native_vector_witdth_float, device::native_vector_witdth_↩
  double, device::native_vector_witdth_half,
  device::max_clock_frequency, device::address_bits, device::max_mem_alloc_size, device::image_support,
  device::max_read_image_args, device::max_write_image_args, device::image2d_max_height, device↩
  ::image2d_max_width,
  device::image3d_max_height, device::image3d_max_widht, device::image3d_mas_depth, device::image_↩
  max_buffer_size,
  device::image_max_array_size, device::max_samplers, device::max_parameter_size, device::mem_base↩
  _addr_align,
  device::single_fp_config, device::double_fp_config, device::global_mem_cache_type, device::global_mem↩
  _cache_line_size,
  device::global_mem_cache_size, device::global_mem_size, device::max_constant_buffer_size, device↩
  ::max_constant_args,
  device::local_mem_type, device::local_mem_size, device::error_correction_support, device::host_unified_↩
  memory,
  device::profiling_timer_resolution, device::endian_little, device::is_available, device::is_compiler_available,
  device::is_linker_available, device::execution_capabilities, device::queue_properties, device::built_in_↩
  kernels,
  device::platform, device::name, device::vendor, device::driver_version,
  device::profile, device::device_version, device::opencl_version, device::extensions,
  device::printf_buffer_size, device::preferred_interop_user_sync, device::parent_device, device::partition_↩
  max_sub_devices,
  device::partition_properties, device::partition_affinity_domain, device::partition_type, device::reference_↩
  count }

  Device information descriptors.
- enum device_partition_property : int {
  device_partition_property::unsupported, device_partition_property::partition_equally, device_partition_↩
  property::partition_by_counts, device_partition_property::partition_by_affinity_domain,
  device_partition_property::partition_affinity_domain_next_partitionable }
- enum device_affinity_domain : int {
  device_affinity_domain::unsupported, device_affinity_domain::numa, device_affinity_domain::L4_cache,
  device_affinity_domain::L3_cache,
  device_affinity_domain::L2_cache, device_affinity_domain::next_partitionable }
- enum device_partition_type : int {
  device_partition_type::no_partition, device_partition_type::numa, device_partition_type::L4_cache, device↩
  _partition_type::L3_cache,
  device_partition_type::L2_cache, device_partition_type::L1_cache }
- enum local_mem_type : int { local_mem_type::none, local_mem_type::local, local_mem_type::global }
- enum fp_config : int {
  fp_config::denorm, fp_config::inf_nan, fp_config::round_to_nearest, fp_config::round_to_zero,
  fp_config::round_to_inf, fp_config::fma, fp_config::correctly_rounded_divide_sqrt, fp_config::soft_float }

- enum global_mem_cache_type : int { global_mem_cache_type::none, global_mem_cache_type::read_only, global_mem_cache_type::write_only }
- enum device_execution_capabilities : unsigned int { device_execution_capabilities::exec_kernel, device_←execution_capabilities::exec_native_kernel }
- enum platform : unsigned int {
  platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_PROFILE), platform::TRISYCL_SKIP_OPEN←CL =(= CL_PLATFORM_VERSION), platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_NAME), platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_VENDOR), platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_EXTENSIONS) }

  Platform information descriptors.
- enum queue : int { queue::context, queue::device, queue::reference_count, queue::properties }

  Queue information descriptors.

9.5.1 Typedef Documentation

9.5.1.1 using **cl::sycl::info::gl_context_interop** = typedef **bool**

Definition at line 31 of file context.hpp.

9.5.1.2 using **cl::sycl::info::queue_profiling** = typedef **bool**

Definition at line 46 of file queue.hpp.

9.5.2 Enumeration Type Documentation

9.5.2.1 enum **cl::sycl::info::context** : int [strong]

Context information descriptors.

Todo Should be unsigned int to be consistent with others?

Enumerator

reference_count

num_devices

gl_interop

Definition at line 37 of file context.hpp.

```
                         : int {
  reference_count,
  num_devices,
  gl_interop
};
```

9.5.2.2 enum cl::sycl::info::queue : int [strong]

Queue information descriptors.

From specification C.4

Todo unsigned int?

Todo To be implemented

Enumerator

context

device

reference_count

properties

Definition at line 56 of file queue.hpp.

```
                    : int {
  context,
  device,
  reference_count,
  properties
};
```

9.6 cl::sycl::trisycl Namespace Reference

Classes

- struct default_error_handler

9.6.1 Detailed Description

Todo Refactor when updating to latest specification

9.7 std Namespace Reference

Classes

- struct hash< cl::sycl::buffer< T, Dimensions, Allocator > >
- struct hash< cl::sycl::device >
- struct hash< cl::sycl::kernel >
- struct hash< cl::sycl::platform >
- struct hash< cl::sycl::queue >

Chapter 10

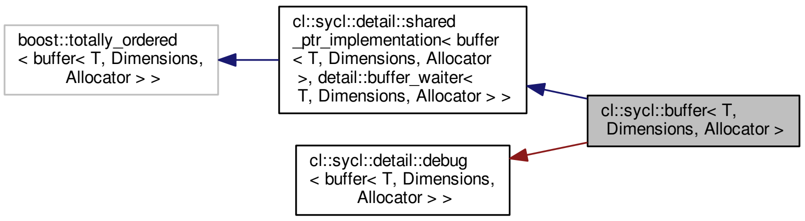
Class Documentation

10.1 cl::sycl::buffer< T, Dimensions, Allocator > Class Template Reference

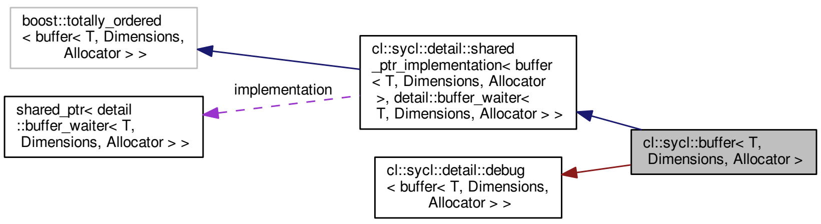
<T, Dimensions, Mode, Target>up data Data access and storage in SYCL

`#include <accessor.hpp>`

Inheritance diagram for cl::sycl::buffer< T, Dimensions, Allocator >:


boost::totally_ordered < buffer< T, Dimensions, Allocator > >
cl::sycl::detail::shared _ptr_implementation< buffer < T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >
cl::sycl::buffer< T, Dimensions, Allocator >
cl::sycl::detail::debug < buffer< T, Dimensions, Allocator > >

Collaboration diagram for cl::sycl::buffer< T, Dimensions, Allocator >:


boost::totally_ordered < buffer< T, Dimensions, Allocator > >
implementation
shared_ptr< detail ::buffer_waiter< T, Dimensions, Allocator > >
cl::sycl::detail::shared _ptr_implementation< buffer < T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >
cl::sycl::buffer< T, Dimensions, Allocator >
cl::sycl::detail::debug < buffer< T, Dimensions, Allocator > >

Public Types

- using value_type = T

  The STL-like types.
- using reference = value_type &
- using const_reference = const value_type &
- using allocator_type = Allocator

Public Member Functions

- buffer ()=default

  Use default constructors so that we can create a new buffer copy from another one, with either a l-value or an r-value (for std::move() for example).
- buffer (const range< Dimensions > &r, Allocator allocator={})

  Create a new buffer of the given size with storage managed by the SYCL runtime.
- buffer (const T ∗host_data, const range< Dimensions > &r, Allocator allocator={})

  Create a new buffer with associated host memory.
- buffer (T ∗host_data, const range< Dimensions > &r, Allocator allocator={})

  Create a new buffer with associated host memory.
- buffer (shared_ptr_class< T > &host_data, const range< Dimensions > &buffer_range, cl::sycl::mutex_↩ class &m, Allocator allocator={})

  Create a new buffer with associated memory, using the data in host_data.
- buffer (shared_ptr_class< T > host_data, const range< Dimensions > &buffer_range, Allocator allocator={})

  Create a new buffer with associated memory, using the data in host_data.
- template<typename D = std::default_delete<T>>
  buffer (unique_ptr_class< T, D > &&host_data, const range< Dimensions > &buffer_range, Allocator allocator={})

  Create a new buffer which is initialized by host_data.
- template<typename InputIterator , typename ValueType = typename std::iterator_traits<InputIterator>::value_type>
  buffer (InputIterator start_iterator, InputIterator end_iterator, Allocator allocator={})

  Create a new allocated 1D buffer initialized from the given elements ranging from first up to one before last.
- buffer (buffer< T, Dimensions, Allocator > &b, const id< Dimensions > &base_index, const range< Dimensions > &sub_range, Allocator allocator={})

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

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

  Get an accessor to the buffer with the required mode.
- template<access::mode Mode, access::target Target = access::target::host_buffer>
  accessor< T, Dimensions, Mode, Target > get_access ()

  Get a host accessor to the buffer with the required mode.
- auto get_range () const

  Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.
- auto get_count () const

  Returns the total number of elements in the buffer.
- size_t get_size () const

  Returns the size of the buffer storage in bytes.
- auto use_count () const

  Returns the number of buffers that are shared/referenced.
- bool is_read_only () const

  Ask for read-only status of the buffer.
- void set_final_data (weak_ptr_class< T > finalData)

  Set destination of buffer data on destruction.

Private Types

- using implementation_t = detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail↩ ::buffer_waiter< T, Dimensions, Allocator >>

Additional Inherited Members

10.1.1 Detailed Description

template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>>
class cl::sycl::buffer< T, Dimensions, Allocator >

<T, Dimensions, Mode, Target>up data Data access and storage in SYCL

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.

Todo We have some read-write buffers and some read-only buffers, according to the constructor called. So we could have some static checking for correctness with the accessors used, but we do not have a way in the specification to have a read-only buffer type for this.

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

Todo Finish allocator implementation

Todo Think about the need of an allocator when constructing a buffer from other buffers

Todo Add constructors from arrays so that in C++17 the range and type can be infered from the constructor

Todo Add constructors from array_ref

Definition at line 27 of file accessor.hpp.

10.1.2 Member Typedef Documentation

10.1.2.1 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> using cl::sycl::buffer< T, Dimensions, Allocator >::allocator_type = Allocator

Definition at line 73 of file buffer.hpp.

10.1.2.2 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> using cl::sycl::buffer< T, Dimensions, Allocator >::const_reference = const value_type&

Definition at line 72 of file buffer.hpp.

10.1.2.3 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> using **cl::sycl::buffer**< T, Dimensions, Allocator >**::implementation_t** = **detail::shared_ptr_implementation**< **buffer**<T, Dimensions, Allocator>, **detail::buffer_waiter**<T, Dimensions, Allocator>> `[private]`

Definition at line 81 of file buffer.hpp.

10.1.2.4 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> using **cl::sycl::buffer**< T, Dimensions, Allocator >**::reference** = **value_type&**

Definition at line 71 of file buffer.hpp.

10.1.2.5 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> using **cl::sycl::buffer**< T, Dimensions, Allocator >**::value_type** = **T**

The STL-like types.

Definition at line 70 of file buffer.hpp.

10.1.3 Constructor & Destructor Documentation

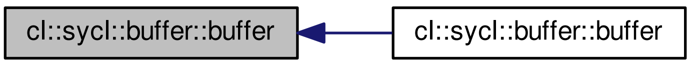
10.1.3.1 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> **cl::sycl::buffer**< T, Dimensions, Allocator >**::buffer** ( ) `[default]`

Use default constructors so that we can create a new buffer copy from another one, with either a l-value or an r-value (for std::move() for example).

Since we just copy the shared_ptr<> from the shared_ptr_implementation above, this is where/how the sharing magic is happening with reference counting in this case.

Referenced by cl::sycl::buffer< T, Dimensions, Allocator >::buffer().

Here is the caller graph for this function:



10.1.3.2 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> **cl::sycl::buffer**< T, Dimensions, Allocator >**::buffer** (const **range**< Dimensions > & *r*, Allocator *allocator* = { }) `[inline]`

Create a new buffer of the given size with storage managed by the SYCL runtime.

The default behavior is to use the default host buffer allocator, in order to allow for host accesses. If the type of the buffer, has the const qualifier, then the default allocator will remove the qualifier to allow host access to the data.

Parameters

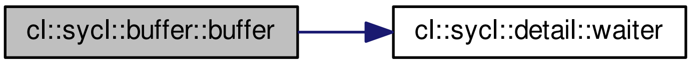
in	*r*	defines the size
in	*allocator*	is to be used by the SYCL runtime

Definition at line 111 of file buffer.hpp.

References cl::sycl::detail::waiter().

```
                                                            {})
     : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
                                            { r }) }
       {}
```

Here is the call graph for this function:



10.1.3.3 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> cl::sycl::buffer< T, Dimensions, Allocator >::buffer (const T ∗ *host_data*, const range< Dimensions > & *r*, Allocator *allocator* = {}) [inline]

Create a new buffer with associated host memory.

Parameters

in	*host_data*	points to the storage and values used by the buffer
in	*r*	defines the size
in	*allocator*	is to be used by the SYCL runtime, of type cl::sycl::buffer_allocator<T> by default

The host address is const T, so the host accesses can be read-only.

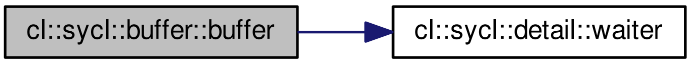
However, the typename T is not const so the device accesses can be both read and write accesses. Since, the host_data is const, this buffer is only initialized with this memory and there is no write after its destruction, unless there is another final data address given after construction of the buffer.

Definition at line 136 of file buffer.hpp.

References cl::sycl::detail::waiter().

```
                              {})
     : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
             { host_data, r }) }
   {}
```

Here is the call graph for this function:



10.1.3.4 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> **cl::sycl::buffer**< T, Dimensions, Allocator >::**buffer** (T * *host_data*, const **range**< Dimensions > & *r*, Allocator *allocator* = { }) [inline]

Create a new buffer with associated host memory.

Parameters

in,out	*host_data*	points to the storage and values used by the buffer
in	*r*	defines the size
in	*allocator*	is to be used by the SYCL runtime, of type cl::sycl::buffer_allocator<T> by default

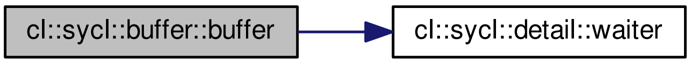
The memory is owned by the runtime during the lifetime of the object. Data is copied back to the host unless the user overrides the behavior using the set_final_data method. host_data points to the storage and values used by the buffer and range<dimensions> defines the size.

Definition at line 160 of file buffer.hpp.

References cl::sycl::detail::waiter().

```
                                                                            {})
    : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
            { host_data, r }) }
  {}
```

Here is the call graph for this function:



10.1.3.5 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> **cl::sycl::buffer**< T, Dimensions, Allocator >::**buffer** (**shared_ptr_class**< T > & *host_data*, const **range**< Dimensions > & *buffer_range*, **cl::sycl::mutex_class** & *m*, Allocator *allocator* = { }) [inline]

Create a new buffer with associated memory, using the data in host_data.

Parameters

`in,out`	*host_data*	points to the storage and values used by the buffer
`in`	*r*	defines the size
`in`	*allocator*	is to be used by the SYCL runtime, of type cl::sycl::buffer_allocator<T> by default

The ownership of the host_data is shared between the runtime and the user. In order to enable both the user application and the SYCL runtime to use the same pointer, a cl::sycl::mutex_class is used. The mutex m is locked by the runtime whenever the data is in use and unlocked otherwise. Data is synchronized with host_data, when the mutex is unlocked by the runtime.

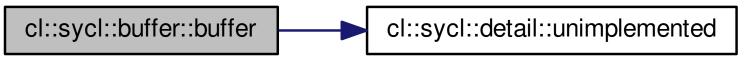
Todo update the specification to replace the pointer by a reference and provide the constructor with and without a mutex

Definition at line 187 of file buffer.hpp.

References cl::sycl::detail::unimplemented().

```
00190                                    {}) {
00191      detail::unimplemented();
00192    }
```

Here is the call graph for this function:



10.1.3.6 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> cl::sycl::buffer< T, Dimensions, Allocator >::buffer (shared_ptr_class< T > *host_data*, const range< Dimensions > & *buffer_range*, Allocator *allocator* = {}) [inline]

Create a new buffer with associated memory, using the data in host_data.

Parameters

`in,out`	*host_data*	points to the storage and values used by the buffer
`in`	*r*	defines the size
`in,out`	*m*	is the mutex used to protect the data access
`in`	*allocator*	is to be used by the SYCL runtime, of type cl::sycl::buffer_allocator<T> by default

The ownership of the host_data is shared between the runtime and the user. In order to enable both the user application and the SYCL runtime to use the same pointer, a cl::sycl::mutex_class is used.

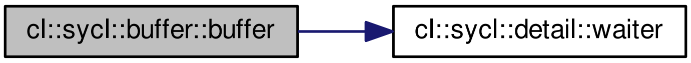
Todo add this mutex-less constructor to the specification

Definition at line 215 of file buffer.hpp.

References cl::sycl::detail::waiter().

```
                               {})
  : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
          { host_data, buffer_range }) }
{}
```

Here is the call graph for this function:



10.1.3.7 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> template<typename D = std::default_delete<T>> cl::sycl::buffer< T, Dimensions, Allocator >::buffer (unique_ptr_class< T, D > && *host_data*, const range< Dimensions > & *buffer_range*, Allocator *allocator* = {}) [inline]

Create a new buffer which is initialized by host_data.

Parameters

in,out	*host_data*	points to the storage and values used to initialize the buffer
in	*r*	defines the size
in	*allocator*	is to be used by the SYCL runtime, of type cl::sycl::buffer_allocator<T> by default

The SYCL runtime receives full ownership of the host_data unique_ptr and there in effect there is no synchronization with the application code using host_data.

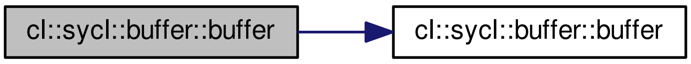
Todo Update the API to add template <typename D = std::default_delete<T>> because the unique_ptr_↩ class/std::unique_ptr have the destructor type as dependent

Definition at line 243 of file buffer.hpp.

References cl::sycl::buffer< T, Dimensions, Allocator >::buffer().

```
                               {})
  // Just delegate to the constructor with normal pointer
    : buffer(host_data.get(), buffer_range, allocator) {
    // Then release the host_data memory
    host_data.release();
  }
```

Here is the call graph for this function:



10.1.3.8 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> template<typename InputIterator , typename ValueType = typename std::iterator_traits<InputIterator>::value_type> cl::sycl::buffer< T, Dimensions, Allocator >::buffer (InputIterator *start_iterator,* InputIterator *end_iterator,* Allocator *allocator* = { }) [inline]

Create a new allocated 1D buffer initialized from the given elements ranging from first up to one before last.

The data is copied to an intermediate memory position by the runtime. Data is written back to the same iterator set if the iterator is not a const iterator.

Parameters

in,out	*start_iterator*	points to the first element to copy
in	*end_iterator*	points to just after the last element to copy
in	*allocator*	is to be used by the SYCL runtime, of type cl::sycl::buffer_allocator<T> by default

Todo Implement the copy back at buffer destruction

Todo Generalize this for n-D and provide column-major and row-major initialization

Todo a reason to have this nD is that set_final_data(weak_ptr_class<T> & finalData) is actually doing this linearization anyway

Todo Allow read-only buffer construction too

Todo update the specification to deal with forward iterators instead and rewrite back only when it is non const and output iterator at least

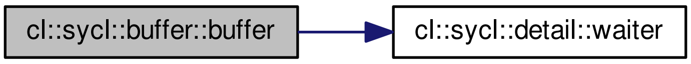
Todo Allow initialization from ranges and collections à la STL

Definition at line 290 of file buffer.hpp.

References cl::sycl::detail::waiter().

```
                                   {}) :
    implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
            { start_iterator, end_iterator }) }
  {}
```

Here is the call graph for this function:



10.1.3.9 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> cl::sycl::buffer< T, Dimensions, Allocator >::buffer (buffer< T, Dimensions, Allocator > & *b*, const id< Dimensions > & *base_index*, const range< Dimensions > & *sub_range*, Allocator *allocator* = { }) [inline]

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

Parameters

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

Todo To be implemented

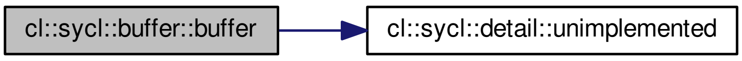
Todo Update the specification to replace index by id

Definition at line 312 of file buffer.hpp.

References cl::sycl::detail::unimplemented().

```
00315                              {}) { detail::unimplemented(); }
```

Here is the call graph for this function:



10.1.3.10 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> cl::sycl::buffer< T, Dimensions, Allocator >::buffer (cl_mem *mem_object*, queue *from_queue*, event *available_event* = { }, Allocator *allocator* = { }) [inline]

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

Parameters

`in,out`	*mem_object*	is the OpenCL memory object to use
`in,out`	*from_queue*	is the queue associated to the memory object
`in`	*available_event*	specifies the event to wait for if non null

Note that a buffer created from a cl_mem object will only have one underlying cl_mem for the lifetime of the buffer and use on an incompatible queue constitues an error.

Todo To be implemented

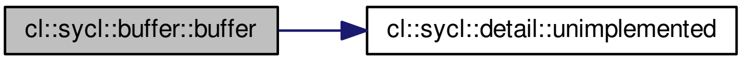
Todo Improve the specification to allow CLHPP objects too

Definition at line 339 of file buffer.hpp.

References cl::sycl::access::global_buffer, and cl::sycl::detail::unimplemented().

```
00341                                {},
00342          Allocator allocator = {}) { detail::unimplemented();  }
```

Here is the call graph for this function:



10.1.4 Member Function Documentation

10.1.4.1 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> template<access::mode Mode, access::target Target = access::target::global_buffer> accessor<T, Dimensions, Mode, Target> cl::sycl::buffer< T, Dimensions, Allocator >::get_access (handler & *command_group_handler*) [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
`in`	*command_group_handler*	is the command group handler in which the kernel is to be executed

Todo Do we need for an accessor to increase the reference count of a buffer object? It does make more sense for a host-side accessor.

Todo Implement the modes and targets

Definition at line 365 of file buffer.hpp.

References cl::sycl::access::constant_buffer, cl::sycl::access::global_buffer, and cl::sycl::access::host_buffer.

```
                                          {
    static_assert(Target == access::target::global_buffer
                  || Target == access::target::constant_buffer,
                  "get_access(handler) can only deal with access::global_buffer"
                  " or access::constant_buffer (for host_buffer accessor"
                  " do not use a command group handler");
    return { *this, command_group_handler };
  }
```

10.1.4.2 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> template<access::mode Mode, access::target Target = access::target::host_buffer> accessor<T, Dimensions, Mode, Target> cl::sycl::buffer< T, Dimensions, Allocator >::get_access ( ) [inline]

Get a host accessor to the buffer with the required mode.

Parameters

Mode	is the requested access mode

Todo Implement the modes

Todo More elegant solution

Definition at line 386 of file buffer.hpp.

References cl::sycl::access::host_buffer.

```
                  {
    static_assert(Target == access::target::host_buffer,
                  "get_access() without a command group handler is only"
                  " for host_buffer accessor");
    return { *this };
  }
```

10.1.4.3 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> auto cl::sycl::buffer< T, Dimensions, Allocator >::get_count ( ) const [inline]

Returns the total number of elements in the buffer.

Equal to get_range()[0] ∗ ... ∗ get_range()[dimensions-1].

Definition at line 415 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
                           {
    return implementation->implementation->get_count();
  }
```

10.1.4.4 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> auto cl::sycl::buffer< T, Dimensions, Allocator >::get_range ( ) const [inline]

Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.

Todo rename to the equivalent from array_ref proposals? Such as size() in http://www.open-std.↩ org/jtc1/sc22/wg21/docs/papers/2016/p0009r2.html

Definition at line 402 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
                          {
    /* Interpret the shape which is a pointer to the first element as an
       array of Dimensions elements so that the range<Dimensions>
       constructor is happy with this collection
    */
    return implementation->implementation->get_range();
  }
```

10.1.4.5 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> size_t cl::sycl::buffer< T, Dimensions, Allocator >::get_size ( ) const [inline]

Returns the size of the buffer storage in bytes.

Equal to get_count()∗sizeof(T).

Todo rename to something else. In http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0. pdf it is named bytes() for example

Definition at line 428 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
                           {
    return implementation->implementation->get_size();
  }
```

10.1.4.6 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> bool cl::sycl::buffer< T, Dimensions, Allocator >::is_read_only ( ) const [inline]

Ask for read-only status of the buffer.

Todo Add to specification

Definition at line 455 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
                               {
    return implementation->implementation->read_only;
  }
```

10.1.4.7 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> void cl::sycl::buffer< T, Dimensions, Allocator >::set_final_data (weak_ptr_class< T > *finalData*) [inline]

Set destination of buffer data on destruction.

The finalData points to the host memory to which, the outcome of all the buffer processing is going to be copied to.

This is the final pointer, which is going to be accessible after the destruction of the buffer and in the case where this is a valid pointer, the data are going to be copied to this host address.

finalData is different from the original host address, if the buffer was created associated with one. This is mainly to be used when a shared_ptr is given in the constructor and the output data will reside in a different location from the initialization data.

It is defined as a weak_ptr referring to a shared_ptr that is not associated with the cl::sycl::buffer, and so the cl↩ ::sycl::buffer will have no ownership of finalData.

Todo Update the API to take finalData by value instead of by reference. This way we can have an implicit conversion possible at the API call from a shared_ptr<>, avoiding an explicit weak_ptr<> creation

Todo figure out how set_final_data() interact with the other way to write back some data or with some data sharing with the host that can not be undone

Definition at line 487 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
                                                   {
    implementation->implementation->set_final_data(std::move(finalData));
  }
```

10.1.4.8 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> auto cl::sycl::buffer< T, Dimensions, Allocator >::use_count ( ) const [inline]

Returns the number of buffers that are shared/referenced.

For example

```
cl::sycl::buffer<int> b { 1000 };
// Here b.use_count() should return 1
cl::sycl::buffer<int> c { b };
// Here b.use_count() and b.use_count() should return 2
```

Todo Add to the specification, useful for validation

Definition at line 445 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
                             {
    // Rely on the shared_ptr<> use_count()
    return implementation.use_count();
  }
```

The documentation for this class was generated from the following files:

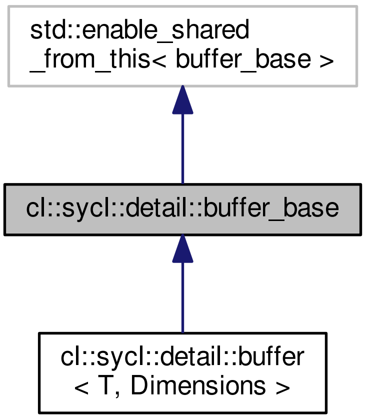
- include/CL/sycl/accessor.hpp
- include/CL/sycl/buffer.hpp

10.2 cl::sycl::detail::buffer_base Struct Reference

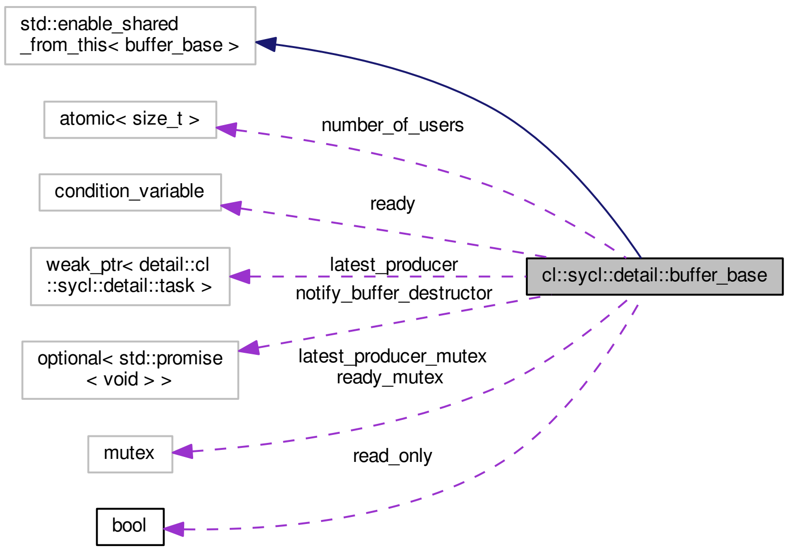
Factorize some template independent buffer aspects in a base class.

```
#include <buffer_base.hpp>
```

Inheritance diagram for cl::sycl::detail::buffer_base:



Collaboration diagram for cl::sycl::detail::buffer_base:



Public Member Functions

- buffer_base (bool read_only)

  Create a buffer base.
- ∼buffer_base ()

  The destructor wait for not being used anymore.
- void wait ()

  Wait for this buffer to be ready, which is no longer in use.
- void use ()

  Mark this buffer in use by a task.
- void release ()

  A task has released the buffer.
- std::shared_ptr< detail::task > get_latest_producer ()

  Return the latest producer for the buffer.
- std::shared_ptr< detail::task > set_latest_producer (std::weak_ptr< detail::task > newer_latest_producer)

  Return the latest producer for the buffer and set another future producer.
- std::shared_ptr< detail::task > add_to_task (handler ∗command_group_handler, bool is_write_mode)

  Add a buffer to the task running the command group.

Public Attributes

- bool read_only

  If the data are read-only, store the information for later optimization.
- std::atomic< size_t > number_of_users
- std::weak_ptr< detail::task > latest_producer

  Track the latest task to produce this buffer.
- std::mutex latest_producer_mutex

  To protect the access to latest_producer.
- std::condition_variable ready

  To signal when this buffer ready.
- std::mutex ready_mutex

  To protect the access to the condition variable.
- boost::optional< std::promise< void > > notify_buffer_destructor

  If the SYCL user buffer destructor is blocking, use this to block until this buffer implementation is destroyed.

10.2.1 Detailed Description

Factorize some template independent buffer aspects in a base class.

Definition at line 41 of file buffer_base.hpp.

10.2.2 Constructor & Destructor Documentation

10.2.2.1 cl::sycl::detail::buffer_base::buffer_base (bool *read_only*) [inline]

Create a buffer base.

Definition at line 68 of file buffer_base.hpp.

```
00068                                  : read_only { read_only },
00069                                    number_of_users { 0 } {}
```

10.2.2.2 cl::sycl::detail::buffer_base::∼buffer_base ( ) [inline]

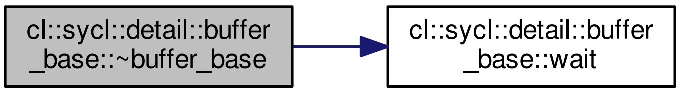
The destructor wait for not being used anymore.

Definition at line 73 of file buffer_base.hpp.

References wait().

```
                      {
    wait();
    // If there is the last SYCL user buffer waiting, notify it
    if (notify_buffer_destructor)
      notify_buffer_destructor->set_value();
  }
```

Here is the call graph for this function:


cl::sycl::detail::buffer_base::~buffer_base
cl::sycl::detail::buffer_base::wait

10.2.3 Member Function Documentation

10.2.3.1 std::shared_ptr<detail::task> cl::sycl::detail::buffer_base::add_to_task (handler ∗ *command_group_handler,* bool *is_write_mode*) [inline]

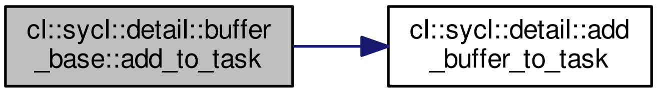
Add a buffer to the task running the command group.

Definition at line 130 of file buffer_base.hpp.

References cl::sycl::detail::add_buffer_to_task().

```
                                                                          {
    return add_buffer_to_task(command_group_handler,
                              shared_from_this(),
                              is_write_mode);
  }
```

Here is the call graph for this function:


cl::sycl::detail::buffer_base::add_to_task
cl::sycl::detail::add_buffer_to_task

10.2.3.2 std::shared_ptr<detail::task> cl::sycl::detail::buffer_base::get_latest_producer ( ) [inline]

Return the latest producer for the buffer.

Definition at line 107 of file buffer_base.hpp.

```
                                                        {
    std::lock_guard<std::mutex> lg { latest_producer_mutex };
    // Return the valid shared_ptr to the task, if any
    return latest_producer.lock();
  }
```

10.2.3.3 void cl::sycl::detail::buffer_base::release ( ) [inline]

A task has released the buffer.

Definition at line 99 of file buffer_base.hpp.

```
                     {
    if (--number_of_users == 0)
      // Notify the host consumers or the buffer destructor that it is ready
      ready.notify_all();
  }
```

10.2.3.4 std::shared_ptr<detail::task> cl::sycl::detail::buffer_base::set_latest_producer (std::weak_ptr< detail::task > *newer_latest_producer*) [inline]

Return the latest producer for the buffer and set another future producer.

Definition at line 118 of file buffer_base.hpp.

```
                                                                              {
    std::lock_guard<std::mutex> lg { latest_producer_mutex };
    using std::swap;

    swap(newer_latest_producer, latest_producer);
    // Return the valid shared_ptr to the previous producing task, if any
    return newer_latest_producer.lock();
  }
```

10.2.3.5 void cl::sycl::detail::buffer_base::use ( ) [inline]

Mark this buffer in use by a task.

Definition at line 92 of file buffer_base.hpp.

References number_of_users.

```
                 {
    // Increment the use count
    ++number_of_users;
  }
```

10.2.3.6 void cl::sycl::detail::buffer_base::wait ( ) [inline]

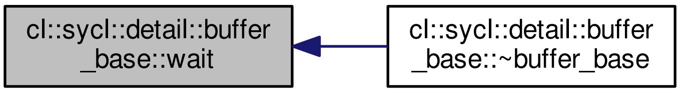
Wait for this buffer to be ready, which is no longer in use.

Definition at line 82 of file buffer_base.hpp.

Referenced by ∼buffer_base().

```
                 {
    std::unique_lock<std::mutex> ul { ready_mutex };
    ready.wait(ul, [&] {
        // When there is no producer for this buffer, we are ready to use it
        return number_of_users == 0;
      });
  }
```

Here is the caller graph for this function:


cl::sycl::detail::buffer_base::wait
cl::sycl::detail::buffer_base::~buffer_base

10.2.4 Member Data Documentation

10.2.4.1 std::weak_ptr< detail::task > cl::sycl::detail::buffer_base::latest_producer

Track the latest task to produce this buffer.

Definition at line 50 of file buffer_base.hpp.

10.2.4.2 std::mutex cl::sycl::detail::buffer_base::latest_producer_mutex

To protect the access to latest_producer.

Definition at line 52 of file buffer_base.hpp.

10.2.4.3 boost::optional< std::promise< void > > cl::sycl::detail::buffer_base::notify_buffer_destructor

If the SYCL user buffer destructor is blocking, use this to block until this buffer implementation is destroyed.

Use a void promise since there is no value to send, only waiting

Definition at line 64 of file buffer_base.hpp.

Referenced by cl::sycl::detail::buffer< T, Dimensions >::get_destructor_future().

10.2.4.4 std::atomic<size_t> cl::sycl::detail::buffer_base::number_of_users

Definition at line 47 of file buffer_base.hpp.

Referenced by use().

10.2.4.5 bool cl::sycl::detail::buffer_base::read_only

If the data are read-only, store the information for later optimization.

Todo Replace this by a static read-only type for the buffer

Definition at line 44 of file buffer_base.hpp.

10.2.4.6 std::condition_variable cl::sycl::detail::buffer_base::ready

To signal when this buffer ready.

Definition at line 55 of file buffer_base.hpp.

10.2.4.7 std::mutex cl::sycl::detail::buffer_base::ready_mutex

To protect the access to the condition variable.

Definition at line 57 of file buffer_base.hpp.

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

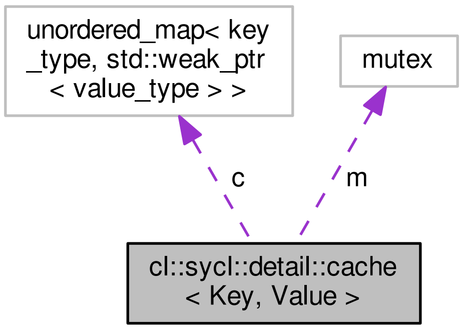
- include/CL/sycl/buffer/detail/buffer_base.hpp

10.3 cl::sycl::detail::cache< Key, Value > Class Template Reference

A simple thread safe cache mechanism to cache std::shared_ptr of values indexed by keys.

`#include <cache.hpp>`

Collaboration diagram for cl::sycl::detail::cache< Key, Value >:


unordered_map< key _type, std::weak_ptr < value_type > >
mutex
c
m
cl::sycl::detail::cache < Key, Value >

Public Types

- using key_type = Key

  The type of the keys used to indexed the cache.
- using value_type = Value

  The base type of the values stored in the cache.

Public Member Functions

- template<typename Functor >
  std::shared_ptr< value_type > get_or_register (const key_type &k, Functor &&create_element)

  Get a value stored in the cache if present or insert by calling a generator function.
- void remove (const key_type &k)

  Remove an entry from the cache.

Private Attributes

- std::unordered_map< key_type, std::weak_ptr< value_type > > c

  The caching storage.
- std::mutex m

  To make the cache thread-safe.

10.3.1 Detailed Description

template<typename Key, typename Value>
class cl::sycl::detail::cache< Key, Value >

A simple thread safe cache mechanism to cache std::shared_ptr of values indexed by keys.

Since internally only std::weak_ptr are stored, this does not prevent object deletion but it is up to the programmer not to use this cache to retrieve deleted objects.

Definition at line 29 of file cache.hpp.

10.3.2 Member Typedef Documentation

10.3.2.1 template<typename Key, typename Value> using cl::sycl::detail::cache< Key, Value >::key_type = Key

The type of the keys used to indexed the cache.

Definition at line 34 of file cache.hpp.

10.3.2.2 template<typename Key, typename Value> using cl::sycl::detail::cache< Key, Value >::value_type = Value

The base type of the values stored in the cache.

Definition at line 37 of file cache.hpp.

10.3.3 Member Function Documentation

10.3.3.1 template<typename Key, typename Value> template<typename Functor > std::shared_ptr<value_type> cl::sycl::detail::cache< Key, Value >::get_or_register (const key_type & *k*, Functor && *create_element*) [inline]

Get a value stored in the cache if present or insert by calling a generator function.

Parameters

in	*k*	is the key used to retrieve the value
in	*create_element*	is the function to be called if the key is not found in the cache to generate a value which is inserted for the key. This function has to produce a value convertible to a shared_ptr

Returns

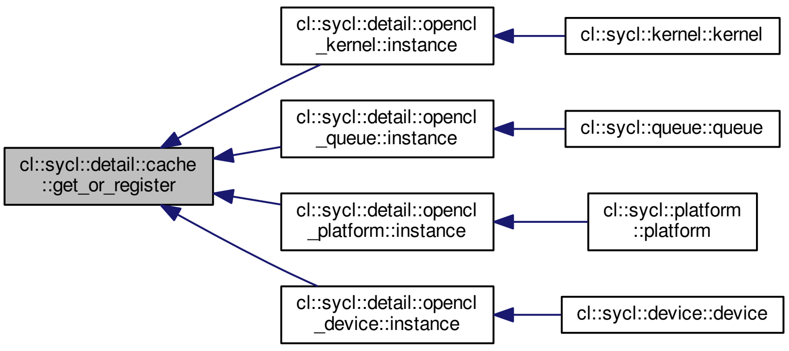
a shared_ptr to the value retrieved or inserted

Definition at line 62 of file cache.hpp.

Referenced by cl::sycl::detail::opencl_kernel::instance(), cl::sycl::detail::opencl_queue::instance(), cl::sycl::detail↩ ::opencl_platform::instance(), and cl::sycl::detail::opencl_device::instance().

```
                                                                          {
    std::lock_guard<std::mutex> lg { m };

    auto i = c.find(k);
    if (i != c.end())
      // Return the found element
      return std::shared_ptr<value_type>{ i->second };

    // Otherwise create and insert a new element
    std::shared_ptr<value_type> e { create_element() };
    c.insert({ k, e });
    return e;
  }
```

Here is the caller graph for this function:


cl::sycl::detail::cache ::get_or_register
cl::sycl::detail::opencl _kernel::instance
cl::sycl::detail::opencl _queue::instance
cl::sycl::detail::opencl _platform::instance
cl::sycl::detail::opencl _device::instance
cl::sycl::kernel::kernel
cl::sycl::queue::queue
cl::sycl::platform ::platform
cl::sycl::device::device

10.3.3.2 template<typename Key, typename Value> void cl::sycl::detail::cache< Key, Value >::remove (const key_type & *k*) [inline]

Remove an entry from the cache.

Parameters

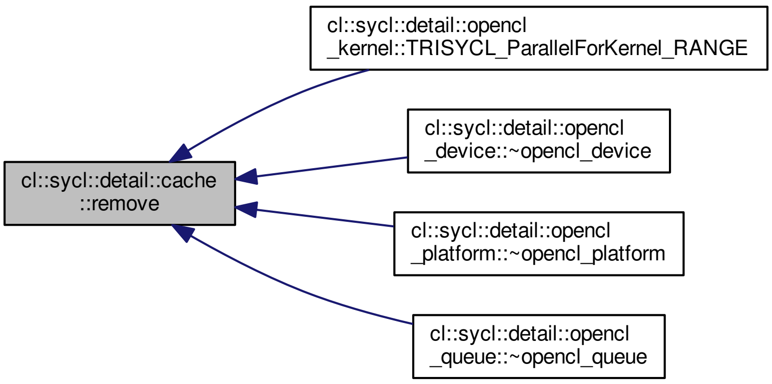
in	*k*	is the key associated to the value to remove from the cache

Definition at line 83 of file cache.hpp.

Referenced by cl::sycl::detail::opencl_kernel::TRISYCL_ParallelForKernel_RANGE(), cl::sycl::detail::opencl_↩
device::∼opencl_device(), cl::sycl::detail::opencl_platform::∼opencl_platform(), and cl::sycl::detail::opencl_↩
queue::∼opencl_queue().

```
                              {
    std::lock_guard<std::mutex> lg { m };
    c.erase(k);
  }
```

Here is the caller graph for this function:



10.3.4 Member Data Documentation

10.3.4.1 template< typename Key, typename Value > std::unordered_map< key_type, std::weak_ptr< value_type > > cl::sycl::detail::cache< Key, Value >::c [private]

The caching storage.

Definition at line 42 of file cache.hpp.

10.3.4.2 template< typename Key, typename Value > std::mutex cl::sycl::detail::cache< Key, Value >::m [private]

To make the cache thread-safe.

Definition at line 45 of file cache.hpp.

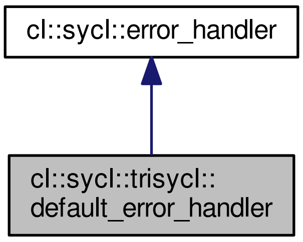
The documentation for this class was generated from the following file:

- include/CL/sycl/detail/cache.hpp

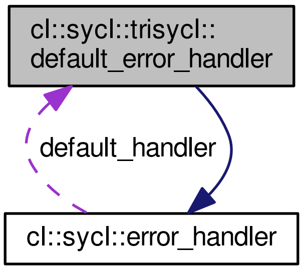
10.4 cl::sycl::trisycl::default_error_handler Struct Reference

`#include <error_handler.hpp>`

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



Collaboration diagram for cl::sycl::trisycl::default_error_handler:



Public Member Functions

- void report_error (exception &) override

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

Additional Inherited Members

10.4.1 Detailed Description

Definition at line 49 of file error_handler.hpp.

10.4.2 Member Function Documentation

10.4.2.1 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.

Definition at line 51 of file error_handler.hpp.

```
00051                                           {
00052     }
```

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

- include/CL/sycl/error_handler.hpp

10.5 cl::sycl::event Class Reference

`#include <event.hpp>`

Public Member Functions

- event ()=default

10.5.1 Detailed Description

Definition at line 14 of file event.hpp.

10.5.2 Constructor & Destructor Documentation

10.5.2.1 cl::sycl::event::event ( ) [default]

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

- include/CL/sycl/event.hpp

10.6 handler_event Class Reference

Handler event.

`#include <handler_event.hpp>`

10.6.1 Detailed Description

Handler event.

Todo To be implemented

Todo To be implemented

Definition at line 19 of file handler_event.hpp.

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

- include/CL/sycl/handler_event.hpp

10.7 std::hash< cl::sycl::buffer< T, Dimensions, Allocator > > Struct Template Reference

`#include <buffer.hpp>`

Public Member Functions

- auto operator() (const cl::sycl::buffer< T, Dimensions, Allocator > &b) const

10.7.1 Detailed Description

template<typename T, std::size_t Dimensions, typename Allocator>
struct std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >

Definition at line 508 of file buffer.hpp.

10.7.2 Member Function Documentation

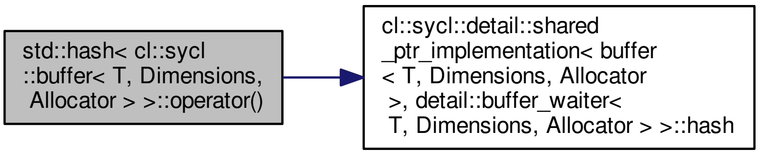
10.7.2.1 template<typename T , std::size_t Dimensions, typename Allocator > auto std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >::operator() (const cl::sycl::buffer< T, Dimensions, Allocator > & *b*) const `[inline]`

Definition at line 510 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::hash().

```
                                                               {
    // Forward the hashing to the implementation
    return b.hash();
  }
```

Here is the call graph for this function:


std::hash< cl::sycl ::buffer< T, Dimensions, Allocator > >::operator()
cl::sycl::detail::shared _ptr_implementation< buffer < T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::hash

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

- include/CL/sycl/buffer.hpp

10.8 std::hash< cl::sycl::device > Struct Template Reference

`#include <device.hpp>`

Public Member Functions

- auto operator() (const cl::sycl::device &d) const

10.8.1 Detailed Description

template<>
struct std::hash< cl::sycl::device >

Definition at line 256 of file device.hpp.

10.8.2 Member Function Documentation

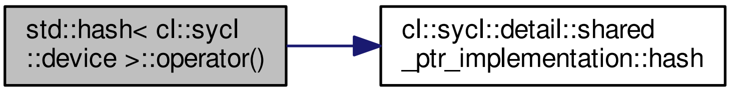
10.8.2.1 auto std::hash< cl::sycl::device >::operator() (const cl::sycl::device & *d*) const [inline]

Definition at line 258 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash().

```
                                            {
    // Forward the hashing to the implementation
    return d.hash();
  }
```

Here is the call graph for this function:


std::hash< cl::sycl ::device >::operator()
cl::sycl::detail::shared _ptr_implementation::hash

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

- include/CL/sycl/device.hpp

10.9 std::hash< cl::sycl::kernel > Struct Template Reference

```
#include <kernel.hpp>
```

Public Member Functions

- auto operator() (const cl::sycl::kernel &k) const

10.9.1 Detailed Description

template<>
struct std::hash< cl::sycl::kernel >

Definition at line 123 of file kernel.hpp.

10.9.2 Member Function Documentation

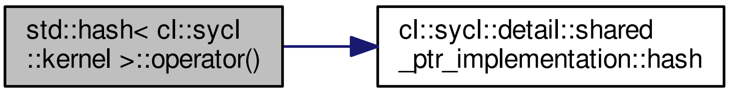
10.9.2.1 auto std::hash< cl::sycl::kernel >::operator() (const cl::sycl::kernel & *k*) const [inline]

Definition at line 125 of file kernel.hpp.

References cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash().

```
                                               {
    // Forward the hashing to the implementation
    return k.hash();
  }
```

Here is the call graph for this function:


std::hash< cl::sycl ::kernel >::operator()
cl::sycl::detail::shared _ptr_implementation::hash

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

- include/CL/sycl/kernel.hpp

10.10 std::hash< cl::sycl::platform > Struct Template Reference

`#include <platform.hpp>`

Public Member Functions

- auto operator() (const cl::sycl::platform &p) const

10.10.1 Detailed Description

template<>
struct std::hash< cl::sycl::platform >

Definition at line 192 of file platform.hpp.

10.10.2 Member Function Documentation

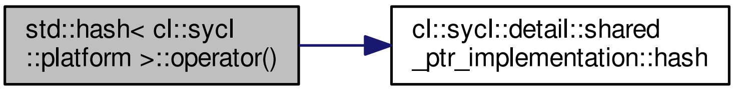
10.10.2.1 auto std::hash< cl::sycl::platform >::operator() (const cl::sycl::platform & *p*) const [inline]

Definition at line 194 of file platform.hpp.

References cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash().

```
                                                  {
      // Forward the hashing to the implementation
      return p.hash();
    }
```

Here is the call graph for this function:


std::hash< cl::sycl ::platform >::operator()
cl::sycl::detail::shared _ptr_implementation::hash

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

- include/CL/sycl/platform.hpp

10.11 std::hash< cl::sycl::queue > Struct Template Reference

```
#include <queue.hpp>
```

Public Member Functions

- auto operator() (const cl::sycl::queue &q) const

10.11.1 Detailed Description

template<>
struct std::hash< cl::sycl::queue >

Definition at line 360 of file queue.hpp.

10.11.2 Member Function Documentation

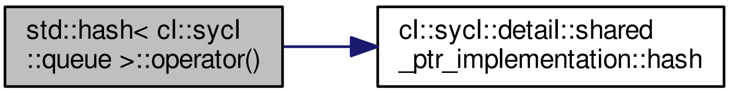
10.11.2.1 auto std::hash< cl::sycl::queue >::operator() (const cl::sycl::queue & *q*) const [inline]

Definition at line 362 of file queue.hpp.

References cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash().

```
                                                           {
    // Forward the hashing to the implementation
    return q.hash();
  }
```

Here is the call graph for this function:


std::hash< cl::sycl ::queue >::operator()
cl::sycl::detail::shared _ptr_implementation::hash

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

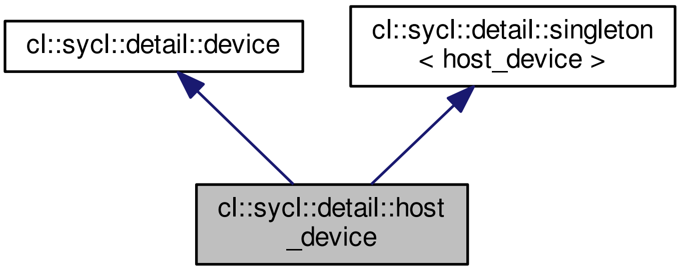
- include/CL/sycl/queue.hpp

10.12 cl::sycl::detail::host_device Class Reference

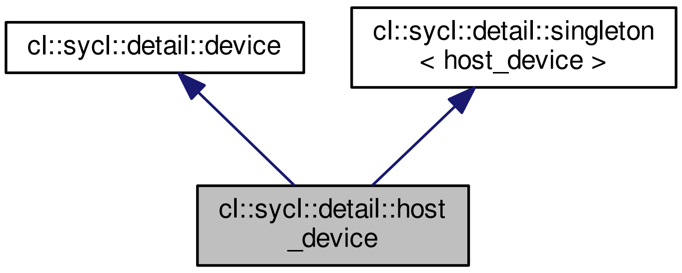
SYCL host device.

```
#include <host_device.hpp>
```

Inheritance diagram for cl::sycl::detail::host_device:


cl::sycl::detail::device
cl::sycl::detail::singleton < host_device >
cl::sycl::detail::host _device

Collaboration diagram for cl::sycl::detail::host_device:



Public Member Functions

- cl_device_id get () const override

    Return the cl_device_id of the underlying OpenCL platform.
- bool is_host () const override

    Return true since the device is a SYCL host device.
- bool is_cpu () const override

    Return false since the host device is not an OpenCL CPU device.
- bool is_gpu () const override

    Return false since the host device is not an OpenCL GPU device.
- bool is_accelerator () const override

    Return false since the host device is not an OpenCL accelerator device.
- cl::sycl::platform get_platform () const override

    Return the platform of device.
- bool has_extension (const string_class &extension) const override

    Specify whether a specific extension is supported on the device.

Additional Inherited Members

10.12.1 Detailed Description

SYCL host device.

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

Definition at line 31 of file host_device.hpp.

10.12.2 Member Function Documentation

10.12.2.1 cl_device_id cl::sycl::detail::host_device::get ( ) const [inline], [override], [virtual]

Return the cl_device_id of the underlying OpenCL platform.

This throws an error since there is no OpenCL device associated to the host device.

Implements cl::sycl::detail::device.

Definition at line 42 of file host_device.hpp.

```
                                     {
    throw non_cl_error("The host device has no OpenCL device");
  }
```

10.12.2.2 cl::sycl::platform cl::sycl::detail::host_device::get_platform ( ) const [inline], [override], [virtual]

Return the platform of device.

Return synchronous errors via the SYCL exception class.

Todo To be implemented

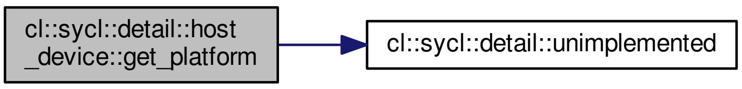
Implements cl::sycl::detail::device.

Definition at line 78 of file host_device.hpp.

References cl::sycl::detail::unimplemented().

```
                                              {
    detail::unimplemented();
    return {};
  }
```

Here is the call graph for this function:


cl::sycl::detail::host_device::get_platform
cl::sycl::detail::unimplemented

10.12.2.3 bool cl::sycl::detail::host_device::has_extension (const string_class & *extension*) const [inline], [override], [virtual]

Specify whether a specific extension is supported on the device.

Todo To be implemented

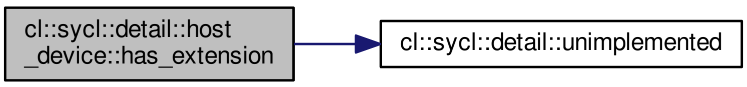
Implements cl::sycl::detail::device.

Definition at line 102 of file host_device.hpp.

References cl::sycl::detail::unimplemented().

```
                                                                  {
    detail::unimplemented();
    return {};
  }
```

Here is the call graph for this function:



10.12.2.4 bool cl::sycl::detail::host_device::is_accelerator ( ) const [inline], [override], [virtual]

Return false since the host device is not an OpenCL accelerator device.

Implements cl::sycl::detail::device.

Definition at line 67 of file host_device.hpp.

```
                                      {
    return false;
  }
```

10.12.2.5 bool cl::sycl::detail::host_device::is_cpu ( ) const [inline], [override], [virtual]

Return false since the host device is not an OpenCL CPU device.

Implements cl::sycl::detail::device.

Definition at line 55 of file host_device.hpp.

```
                              {
    return false;
  }
```

10.12.2.6 bool cl::sycl::detail::host_device::is_gpu ( ) const [inline], [override], [virtual]

Return false since the host device is not an OpenCL GPU device.

Implements cl::sycl::detail::device.

Definition at line 61 of file host_device.hpp.

```
00061                              {
00062     return false;
00063   }
```

10.12.2.7 bool cl::sycl::detail::host_device::is_host ( ) const [inline], [override], [virtual]

Return true since the device is a SYCL host device.

Implements cl::sycl::detail::device.

Definition at line 49 of file host_device.hpp.

```
00049                               {
00050     return true;
00051   }
```

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

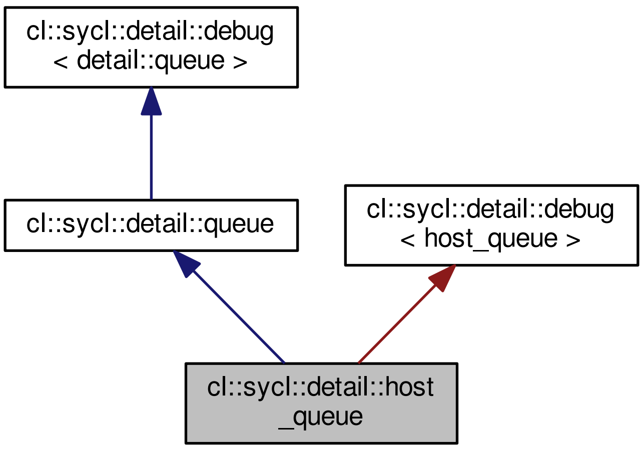
- include/CL/sycl/device/detail/host_device.hpp

10.13 cl::sycl::detail::host_queue Class Reference

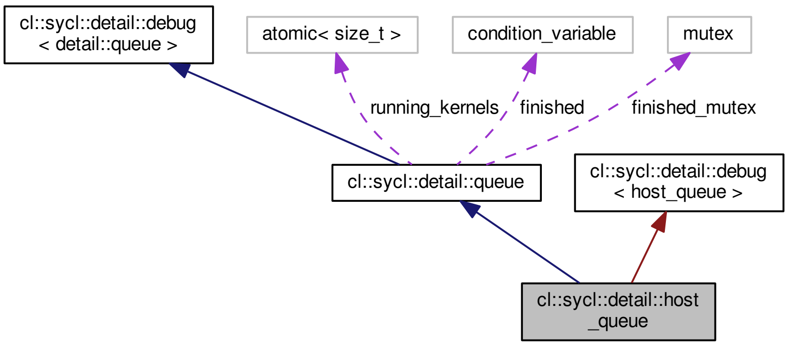
Some implementation details about the SYCL queue.

`#include <host_queue.hpp>`

Inheritance diagram for cl::sycl::detail::host_queue:


cl::sycl::detail::debug < detail::queue >
cl::sycl::detail::queue
cl::sycl::detail::debug < host_queue >
cl::sycl::detail::host _queue

Collaboration diagram for cl::sycl::detail::host_queue:


cl::sycl::detail::debug < detail::queue >
atomic< size_t >
condition_variable
mutex
running_kernels
finished
finished_mutex
cl::sycl::detail::queue
cl::sycl::detail::debug < host_queue >
cl::sycl::detail::host _queue

Private Member Functions

- cl_command_queue get () const override

  Return the cl_command_queue of the underlying OpenCL queue.
- boost::compute::command_queue & get_boost_compute () override

  Return the underlying Boost.Compute command queue.
- cl::sycl::context get_context () const override

  Return the SYCL host queue's host context.
- cl::sycl::device get_device () const override

  Return the SYCL host device the host queue is associated with.
- bool is_host () const override

  Claim proudly that the queue is executing on the SYCL host device.

Additional Inherited Members

10.13.1 Detailed Description

Some implementation details about the SYCL queue.

Todo Once a triSYCL queue is no longer blocking, make this a singleton

Definition at line 29 of file host_queue.hpp.

10.13.2 Member Function Documentation

10.13.2.1 cl_command_queue cl::sycl::detail::host_queue::get ( ) const [inline], [override], [private], [virtual]

Return the cl_command_queue of the underlying OpenCL queue.

This throws an error since there is no OpenCL queue associated to the host queue.

Implements cl::sycl::detail::queue.

Definition at line 38 of file host_queue.hpp.

```
                                         {
    throw non_cl_error("The host queue has no OpenCL command queue");
  }
```

10.13.2.2 boost::compute::command_queue& cl::sycl::detail::host_queue::get_boost_compute ( ) [inline], [override], [private], [virtual]

Return the underlying Boost.Compute command queue.

This throws an error since there is no OpenCL queue associated to the host queue.

Implements cl::sycl::detail::queue.

Definition at line 48 of file host_queue.hpp.

```
                                                          {
    throw non_cl_error("The host queue has no OpenCL command queue");
  }
```

10.13.2.3 cl::sycl::context cl::sycl::detail::host_queue::get_context ( ) const [inline], [override], [private], [virtual]

Return the SYCL host queue's host context.

Implements cl::sycl::detail::queue.

Definition at line 55 of file host_queue.hpp.

```
                                         {
    // Return the default context which is the host context
    return {};
  }
```

10.13.2.4 cl::sycl::device cl::sycl::detail::host_queue::get_device ( ) const [inline], [override], [private], [virtual]

Return the SYCL host device the host queue is associated with.

Implements cl::sycl::detail::queue.

Definition at line 62 of file host_queue.hpp.

```
                                       {
    // Return the default device which is the host device
    return {};
  }
```

10.13.2.5 bool cl::sycl::detail::host_queue::is_host ( ) const [inline], [override], [private], [virtual]

Claim proudly that the queue is executing on the SYCL host device.

Implements cl::sycl::detail::queue.

Definition at line 69 of file host_queue.hpp.

```
                            {
    return true;
  }
```

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

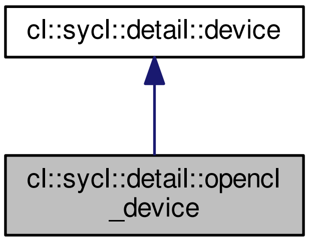
- include/CL/sycl/queue/detail/host_queue.hpp

10.14 cl::sycl::detail::opencl_device Class Reference

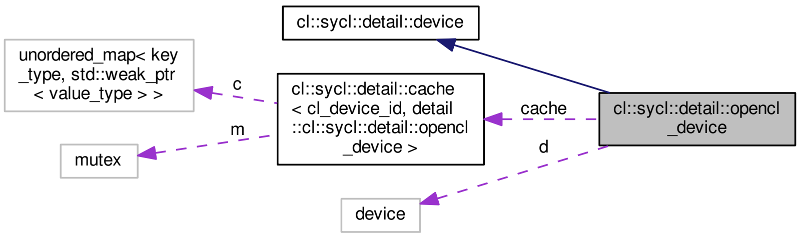
SYCL OpenCL device.

`#include <opencl_device.hpp>`

Inheritance diagram for cl::sycl::detail::opencl_device:



Collaboration diagram for cl::sycl::detail::opencl_device:


cl::sycl::detail::device
unordered_map< key_type, std::weak_ptr< value_type > >
c
cl::sycl::detail::cache< cl_device_id, detail::cl::sycl::detail::opencl_device >
cache
cl::sycl::detail::opencl_device
m
mutex
d
device

Public Member Functions

- cl_device_id get () const override

  Return the cl_device_id of the underlying OpenCL device.
- bool is_host () const override

  Return false since an OpenCL device is not the SYCL host device.
- bool is_cpu () const override

  Test if the OpenCL is a CPU device.
- bool is_gpu () const override

  Test if the OpenCL is a GPU device.
- bool is_accelerator () const override

  Test if the OpenCL is an accelerator device.
- cl::sycl::platform get_platform () const override

  Return the platform of device.
- bool has_extension (const string_class &extension) const override

  Specify whether a specific extension is supported on the device.
- ∼opencl_device () override

  Unregister from the cache on destruction.

Static Public Member Functions

- static std::shared_ptr< opencl_device > instance (const boost::compute::device &d)

Private Member Functions

- opencl_device (const boost::compute::device &d)

  Only the instance factory can built it.

Private Attributes

- boost::compute::device d

  Use the Boost Compute abstraction of the OpenCL device.

Static Private Attributes

- static detail::cache< cl_device_id, detail::opencl_device > cache

  A cache to always return the same alive device for a given OpenCL device.

10.14.1 Detailed Description

SYCL OpenCL device.

Definition at line 30 of file opencl_device.hpp.

10.14.2 Constructor & Destructor Documentation

10.14.2.1 cl::sycl::detail::opencl_device::opencl_device (const boost::compute::device & *d*) [inline], [private]

Only the instance factory can built it.

Definition at line 120 of file opencl_device.hpp.

```
00120 : d { d } {}
```

10.14.2.2 cl::sycl::detail::opencl_device::∼opencl_device ( ) [inline], [override]

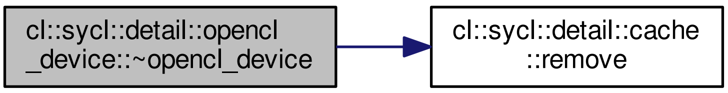
Unregister from the cache on destruction.

Definition at line 125 of file opencl_device.hpp.

References cl::sycl::detail::__attribute__, cache, and cl::sycl::detail::cache< Key, Value >::remove().

```
00125                              {
00126     cache.remove(d.id());
00127   }
```

Here is the call graph for this function:



10.14.3 Member Function Documentation

10.14.3.1 cl_device_id cl::sycl::detail::opencl_device::get ( ) const [inline], [override], [virtual]

Return the cl_device_id of the underlying OpenCL device.

Implements cl::sycl::detail::device.

Definition at line 45 of file opencl_device.hpp.

```
                                                {
    return d.id();
  }
```

10.14.3.2 cl::sycl::platform cl::sycl::detail::opencl_device::get_platform ( ) const [inline], [override], [virtual]

Return the platform of device.

Return synchronous errors via the SYCL exception class.

Todo To be implemented

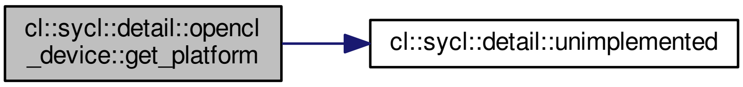
Implements cl::sycl::detail::device.

Definition at line 80 of file opencl_device.hpp.

References cl::sycl::detail::unimplemented().

```
                                                           {
    detail::unimplemented();
    return {};
  }
```

Here is the call graph for this function:



10.14.3.3 bool cl::sycl::detail::opencl_device::has_extension (const string_class & *extension*) const [inline], [override], [virtual]

Specify whether a specific extension is supported on the device.

Todo To be implemented

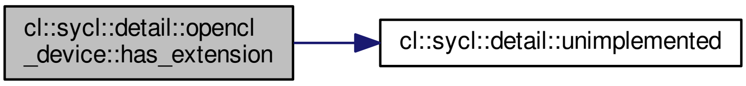
Implements cl::sycl::detail::device.

Definition at line 104 of file opencl_device.hpp.

References cl::sycl::detail::unimplemented().

```
                                                                          {
    detail::unimplemented();
    return {};
  }
```

Here is the call graph for this function:



10.14.3.4 static std::shared_ptr<opencl_device> cl::sycl::detail::opencl_device::instance (const boost::compute::device & *d*) [inline], [static]

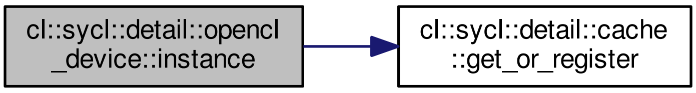
Definition at line 112 of file opencl_device.hpp.

References cl::sycl::detail::cache< Key, Value >::get_or_register().

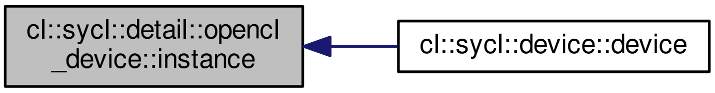
Referenced by cl::sycl::device::device().

```
                                            {
    return cache.get_or_register(d.id(),
                                 [&] { return new opencl_device { d }; });
  }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.14.3.5 bool cl::sycl::detail::opencl_device::is_accelerator ( ) const [inline], [override], [virtual]

Test if the OpenCL is an accelerator device.

Implements cl::sycl::detail::device.

Definition at line 69 of file opencl_device.hpp.

```
                                        {
    return d.type() == boost::compute::device::accelerator;
  }
```

10.14.3.6 bool cl::sycl::detail::opencl_device::is_cpu ( ) const [inline], [override], [virtual]

Test if the OpenCL is a CPU device.

Implements cl::sycl::detail::device.

Definition at line 57 of file opencl_device.hpp.

```
                                {
    return d.type() == boost::compute::device::cpu;
  }
```

10.14.3.7 bool cl::sycl::detail::opencl_device::is_gpu ( ) const [inline], [override], [virtual]

Test if the OpenCL is a GPU device.

Implements cl::sycl::detail::device.

Definition at line 63 of file opencl_device.hpp.

```
                                {
    return d.type() == boost::compute::device::gpu;
  }
```

10.14.3.8 bool cl::sycl::detail::opencl_device::is_host ( ) const [inline], [override], [virtual]

Return false since an OpenCL device is not the SYCL host device.

Implements cl::sycl::detail::device.

Definition at line 51 of file opencl_device.hpp.

```
                                    {
    return false;
  }
```

10.14.4 Member Data Documentation

10.14.4.1 detail::cache<cl_device_id, detail::opencl_device> cl::sycl::detail::opencl_device::cache [static], [private]

A cache to always return the same alive device for a given OpenCL device.

C++11 guaranties the static construction is thread-safe

Definition at line 40 of file opencl_device.hpp.

Referenced by ∼opencl_device().

10.14.4.2 boost::compute::device cl::sycl::detail::opencl_device::d [private]

Use the Boost Compute abstraction of the OpenCL device.

Definition at line 33 of file opencl_device.hpp.

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

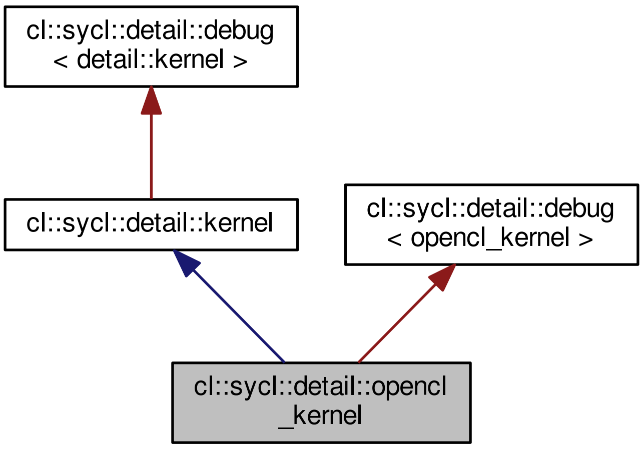
- include/CL/sycl/device/detail/opencl_device.hpp

10.15 cl::sycl::detail::opencl_kernel Class Reference

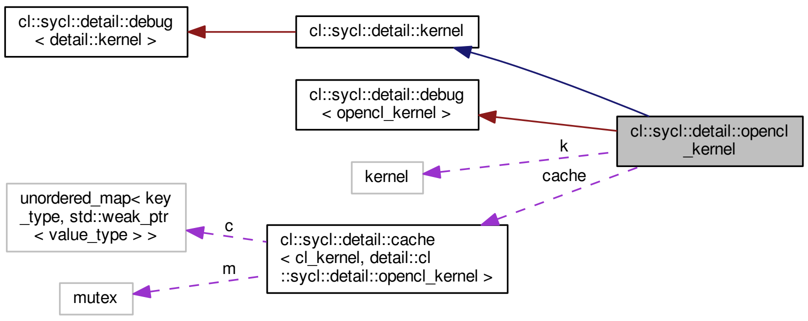
An abstraction of the OpenCL kernel.

`#include <opencl_kernel.hpp>`

Inheritance diagram for cl::sycl::detail::opencl_kernel:


cl::sycl::detail::debug < detail::kernel >
cl::sycl::detail::kernel
cl::sycl::detail::debug < opencl_kernel >
cl::sycl::detail::opencl _kernel

Collaboration diagram for cl::sycl::detail::opencl_kernel:


cl::sycl::detail::debug < detail::kernel >
cl::sycl::detail::kernel
cl::sycl::detail::debug < opencl_kernel >
cl::sycl::detail::opencl _kernel
k
kernel
cache
unordered_map< key _type, std::weak_ptr < value_type > >
c
cl::sycl::detail::cache < cl_kernel, detail::cl ::sycl::detail::opencl_kernel >
m
mutex

Public Member Functions

- cl_kernel get () const override

  Return the underlying OpenCL object.
- boost::compute::kernel get_boost_compute () const override

  Return the Boost.Compute OpenCL kernel object for this kernel.
- TRISYCL_ParallelForKernel_RANGE (1) TRISYCL_ParallelForKernel_RANGE(2) TRISYCL_ParallelFor↩ Kernel_RANGE(3)∼opencl_kernel() override

  Unregister from the cache on destruction.

Static Public Member Functions

- static std::shared_ptr< opencl_kernel > instance (const boost::compute::kernel &k)

Private Member Functions

- opencl_kernel (const boost::compute::kernel &k)

Private Attributes

- boost::compute::kernel k

  Use the Boost Compute abstraction of the OpenCL kernel.

Static Private Attributes

- static detail::cache< cl_kernel, detail::opencl_kernel > cache

  A cache to always return the same alive kernel for a given OpenCL kernel.

10.15.1 Detailed Description

An abstraction of the OpenCL kernel.

Definition at line 28 of file opencl_kernel.hpp.

10.15.2 Constructor & Destructor Documentation

10.15.2.1 cl::sycl::detail::opencl_kernel::opencl_kernel (const boost::compute::kernel & *k*) [inline], [private]

Definition at line 41 of file opencl_kernel.hpp.

```
: k { k } {}
```

10.15.3 Member Function Documentation

10.15.3.1 cl_kernel cl::sycl::detail::opencl_kernel::get ( ) const [inline], [override], [virtual]

Return the underlying OpenCL object.

Todo Improve the spec to deprecate C OpenCL host API and move to C++ instead to avoid this ugly ownership management

Todo Test error and throw. Externalize this feature in Boost.Compute?

Implements cl::sycl::detail::kernel.

Definition at line 57 of file opencl_kernel.hpp.

```
                                 {
    /// \todo Test error and throw. Externalize this feature in Boost.Compute?
    clRetainKernel(k);
    return k.get();
  }
```

10.15.3.2 boost::compute::kernel cl::sycl::detail::opencl_kernel::get_boost_compute ( ) const [inline], [override], [virtual]

Return the Boost.Compute OpenCL kernel object for this kernel.

This is an extension.

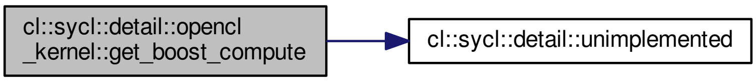
Implements cl::sycl::detail::kernel.

Definition at line 68 of file opencl_kernel.hpp.

References k, and cl::sycl::detail::unimplemented().

```
                                                        {
    return k;
  }
```

Here is the call graph for this function:


cl::sycl::detail::opencl_kernel::get_boost_compute
cl::sycl::detail::unimplemented

10.15.3.3 static std::shared_ptr< opencl_kernel > cl::sycl::detail::opencl_kernel::instance (const boost::compute::kernel & *k*) [inline], [static]

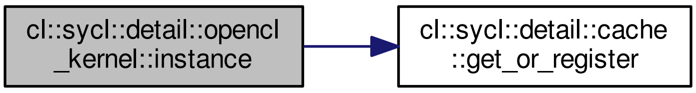
Definition at line 47 of file opencl_kernel.hpp.

References cl::sycl::detail::cache< Key, Value >::get_or_register().

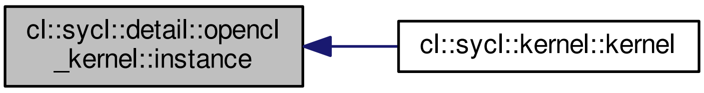
Referenced by cl::sycl::kernel::kernel().

```
                                      {
    return cache.get_or_register(k.get(),
                                 [&] { return new opencl_kernel { k }; });
  }
```

Here is the call graph for this function:


cl::sycl::detail::opencl_kernel::instance
cl::sycl::detail::cache::get_or_register

Here is the caller graph for this function:



10.15.3.4 cl::sycl::detail::opencl_kernel::TRISYCL_ParallelForKernel_RANGE (1) [inline], [override]

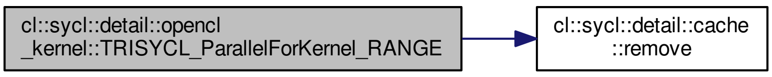
Unregister from the cache on destruction.

Definition at line 110 of file opencl_kernel.hpp.

References cl::sycl::detail::__attribute__, cache, and cl::sycl::detail::cache< Key, Value >::remove().

```
                                {
    cache.remove(k.get());
  }
```

Here is the call graph for this function:



10.15.4 Member Data Documentation

10.15.4.1 detail::cache<cl_kernel, detail::opencl_kernel> cl::sycl::detail::opencl_kernel::cache [static], [private]

A cache to always return the same alive kernel for a given OpenCL kernel.

C++11 guaranties the static construction is thread-safe

Definition at line 39 of file opencl_kernel.hpp.

Referenced by TRISYCL_ParallelForKernel_RANGE().

10.15.4.2 boost::compute::kernel cl::sycl::detail::opencl_kernel::k [private]

Use the Boost Compute abstraction of the OpenCL kernel.

Definition at line 32 of file opencl_kernel.hpp.

Referenced by get_boost_compute().

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

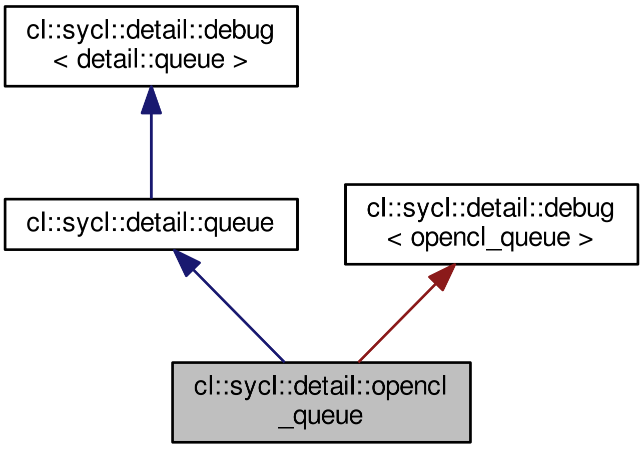
- include/CL/sycl/kernel/detail/opencl_kernel.hpp

10.16 cl::sycl::detail::opencl_queue Class Reference

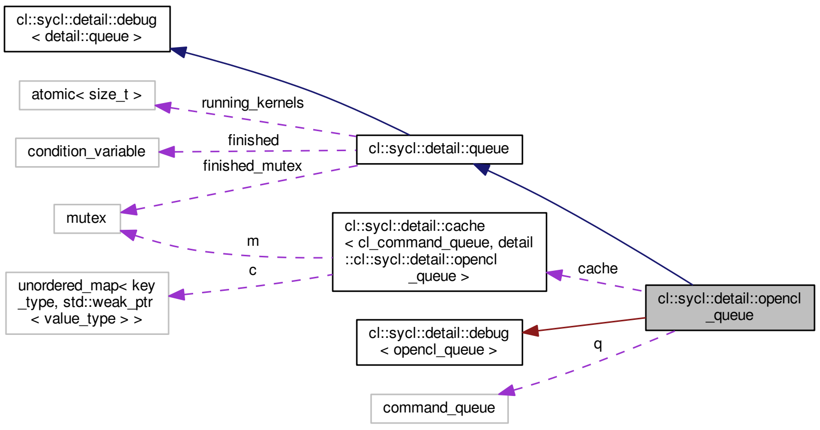
Some implementation details about the SYCL queue.

```
#include <opencl_queue.hpp>
```

Inheritance diagram for cl::sycl::detail::opencl_queue:


cl::sycl::detail::debug < detail::queue >
cl::sycl::detail::queue
cl::sycl::detail::debug < opencl_queue >
cl::sycl::detail::opencl _queue

Collaboration diagram for cl::sycl::detail::opencl_queue:


cl::sycl::detail::debug < detail::queue >
atomic< size_t >
running_kernels
condition_variable
finished
finished_mutex
cl::sycl::detail::queue
mutex
m
cl::sycl::detail::cache < cl_command_queue, detail ::cl::sycl::detail::opencl _queue >
cache
c
unordered_map< key _type, std::weak_ptr < value_type > >
cl::sycl::detail::opencl _queue
cl::sycl::detail::debug < opencl_queue >
q
command_queue

Public Member Functions

- ∼opencl_queue () override

    Unregister from the cache on destruction.

Static Public Member Functions

- static std::shared_ptr< opencl_queue > instance (const boost::compute::command_queue &q)

Private Member Functions

- cl_command_queue get () const override

    Return the cl_command_queue of the underlying OpenCL queue.
- boost::compute::command_queue & get_boost_compute () override

    Return the underlying Boost.Compute command queue.
- cl::sycl::context get_context () const override

    Return the SYCL context associated to the queue.
- cl::sycl::device get_device () const override

    Return the SYCL device associated to the queue.
- bool is_host () const override

    Claim proudly that an OpenCL queue cannot be the SYCL host queue.
- opencl_queue (const boost::compute::command_queue &q)

    Only the instance factory can built it.

Private Attributes

- boost::compute::command_queue q

    Use the Boost Compute abstraction of the OpenCL command queue.

Static Private Attributes

- static detail::cache< cl_command_queue, detail::opencl_queue > cache

    A cache to always return the same alive queue for a given OpenCL command queue.

Additional Inherited Members

10.16.1 Detailed Description

Some implementation details about the SYCL queue.

Definition at line 23 of file opencl_queue.hpp.

10.16.2 Constructor & Destructor Documentation

10.16.2.1 cl::sycl::detail::opencl_queue::opencl_queue (const boost::compute::command_queue & *q*) [inline], [private]

Only the instance factory can built it.

Definition at line 69 of file opencl_queue.hpp.

```
00069 : q { q } {}
```

10.16.2.2 cl::sycl::detail::opencl_queue::∼opencl_queue ( ) [inline], [override]

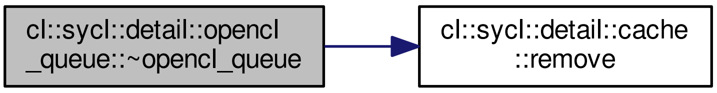
Unregister from the cache on destruction.

Definition at line 82 of file opencl_queue.hpp.

References cl::sycl::detail::__attribute__, cache, and cl::sycl::detail::cache< Key, Value >::remove().

```
00082                            {
00083     cache.remove(q.get());
00084   }
```

Here is the call graph for this function:



10.16.3 Member Function Documentation

10.16.3.1 cl_command_queue cl::sycl::detail::opencl_queue::get ( ) const [inline], [override], [private], [virtual]

Return the cl_command_queue of the underlying OpenCL queue.

Implements cl::sycl::detail::queue.

Definition at line 36 of file opencl_queue.hpp.

```
00036                                           {
00037     return q.get();
00038   }
```

10.16.3.2 boost::compute::command_queue& cl::sycl::detail::opencl_queue::get_boost_compute ( ) [inline], [override], [private], [virtual]

Return the underlying Boost.Compute command queue.

Implements cl::sycl::detail::queue.

Definition at line 42 of file opencl_queue.hpp.

References q.

```
00042                                                               {
00043     return q;
00044   }
```

10.16.3.3 cl::sycl::context cl::sycl::detail::opencl_queue::get_context ( ) const [inline], [override], [private], [virtual]

Return the SYCL context associated to the queue.

Todo Finish context

Implements cl::sycl::detail::queue.

Definition at line 49 of file opencl_queue.hpp.

```
00049                                               {
00050 //    return q.get_context();
00051     return {};
00052   }
```

10.16.3.4 cl::sycl::device cl::sycl::detail::opencl_queue::get_device ( ) const [inline], [override], [private], [virtual]

Return the SYCL device associated to the queue.

Implements cl::sycl::detail::queue.

Definition at line 56 of file opencl_queue.hpp.

```
                                              {
    return q.get_device();
  }
```

10.16.3.5 static std::shared_ptr<opencl_queue> cl::sycl::detail::opencl_queue::instance (const boost::compute::command_queue & *q*) [inline], [static]

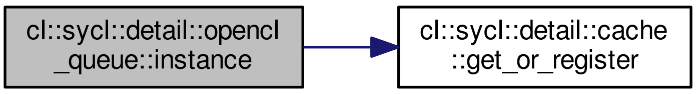
Definition at line 75 of file opencl_queue.hpp.

References cl::sycl::detail::cache< Key, Value >::get_or_register().

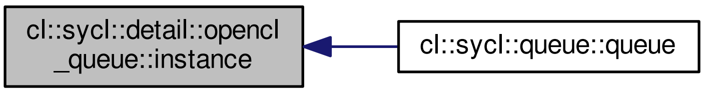
Referenced by cl::sycl::queue::queue().

```
                                                    {
    return cache.get_or_register(q.get(),
                                [&] { return new opencl_queue { q }; });
  }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.16.3.6 bool cl::sycl::detail::opencl_queue::is_host ( ) const [inline], [override], [private], [virtual]

Claim proudly that an OpenCL queue cannot be the SYCL host queue.

Implements cl::sycl::detail::queue.

Definition at line 62 of file opencl_queue.hpp.

```
00062                                     {
00063     return false;
00064   }
```

10.16.4 Member Data Documentation

10.16.4.1 detail::cache<cl_command_queue, detail::opencl_queue> cl::sycl::detail::opencl_queue::cache [static], [private]

A cache to always return the same alive queue for a given OpenCL command queue.

C++11 guaranties the static construction is thread-safe

Definition at line 33 of file opencl_queue.hpp.

Referenced by ∼opencl_queue().

10.16.4.2 boost::compute::command_queue cl::sycl::detail::opencl_queue::q [private]

Use the Boost Compute abstraction of the OpenCL command queue.

Definition at line 26 of file opencl_queue.hpp.

Referenced by get_boost_compute().

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

- include/CL/sycl/queue/detail/opencl_queue.hpp

10.17 cl::sycl::info::param_traits< T, Param > Struct Template Reference

Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value)

`#include <param_traits.hpp>`

10.17.1 Detailed Description

template<typename T, T Param>
struct cl::sycl::info::param_traits< T, Param >

Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value)

Definition at line 20 of file param_traits.hpp.

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

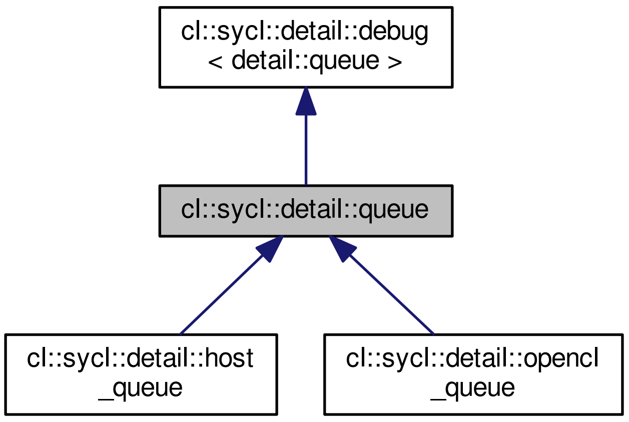
- include/CL/sycl/info/param_traits.hpp

10.18 cl::sycl::detail::queue Struct Reference

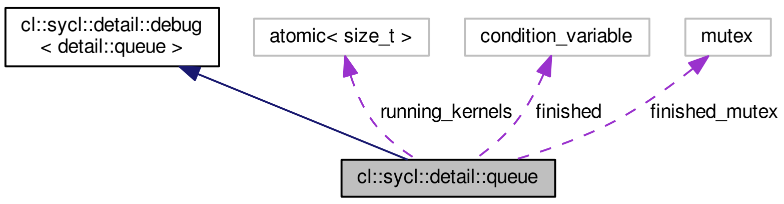
Some implementation details about the SYCL queue.

`#include <queue.hpp>`

Inheritance diagram for cl::sycl::detail::queue:


cl::sycl::detail::debug < detail::queue >
cl::sycl::detail::queue
cl::sycl::detail::host _queue
cl::sycl::detail::opencl _queue

Collaboration diagram for cl::sycl::detail::queue:



Public Member Functions

- queue ()

  Initialize the queue with 0 running kernel.
- void wait_for_kernel_execution ()

  Wait for all kernel completion.
- void kernel_start ()

  Signal that a new kernel started on this queue.
- void kernel_end ()

  Signal that a new kernel finished on this queue.
- virtual cl_command_queue get () const =0

  Return the underlying OpenCL command queue after doing a retain.
- virtual boost::compute::command_queue & get_boost_compute ()=0

  Return the underlying Boost.Compute command queue.
- virtual cl::sycl::context get_context () const =0

  Return the SYCL queue's context.
- virtual cl::sycl::device get_device () const =0

  Return the SYCL device the queue is associated with.
- virtual bool is_host () const =0

  Return whether the queue is executing on a SYCL host device.
- virtual ∼queue ()

  Wait for all kernel completion before the queue destruction.

Public Attributes

- std::atomic< size_t > running_kernels

  Track the number of kernels still running to wait for their completion.
- std::condition_variable finished

  To signal when all the kernels have completed.
- std::mutex finished_mutex

  To protect the access to the condition variable.

10.18.1 Detailed Description

Some implementation details about the SYCL queue.

Definition at line 30 of file queue.hpp.

10.18.2 Constructor & Destructor Documentation

10.18.2.1 cl::sycl::detail::queue::queue ( ) [inline]

Initialize the queue with 0 running kernel.

Definition at line 41 of file queue.hpp.

```
            {
    running_kernels = 0;
  }
```

10.18.2.2 virtual cl::sycl::detail::queue::∼queue ( ) [inline], [virtual]

Wait for all kernel completion before the queue destruction.

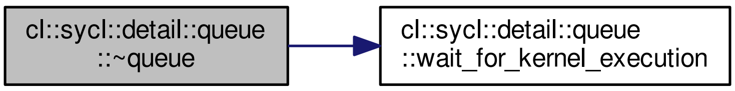
Todo Update according spec since queue destruction is non blocking

Definition at line 114 of file queue.hpp.

References wait_for_kernel_execution().

```
                        {
    wait_for_kernel_execution();
  }
```

Here is the call graph for this function:


cl::sycl::detail::queue ::~queue
cl::sycl::detail::queue ::wait_for_kernel_execution

10.18.3 Member Function Documentation

10.18.3.1 virtual cl_command_queue cl::sycl::detail::queue::get ( ) const [pure virtual]

Return the underlying OpenCL command queue after doing a retain.

This memory object is expected to be released by the developer.

Retain a reference to the returned cl_command_queue object.

Caller should release it when finished.

If the queue is a SYCL host queue then an exception is thrown.

Implemented in cl::sycl::detail::host_queue, and cl::sycl::detail::opencl_queue.

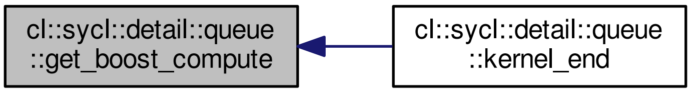
10.18.3.2 virtual boost::compute::command_queue& cl::sycl::detail::queue::get_boost_compute ( ) [pure virtual]

Return the underlying Boost.Compute command queue.

Implemented in cl::sycl::detail::host_queue, and cl::sycl::detail::opencl_queue.

Referenced by kernel_end().

Here is the caller graph for this function:


cl::sycl::detail::queue ::get_boost_compute
cl::sycl::detail::queue ::kernel_end

10.18.3.3 virtual cl::sycl::context cl::sycl::detail::queue::get_context ( ) const [pure virtual]

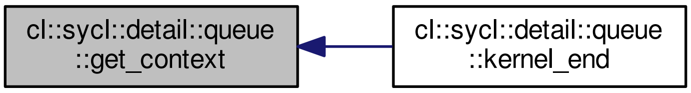
Return the SYCL queue's context.

Report errors using SYCL exception classes.

Implemented in cl::sycl::detail::host_queue, and cl::sycl::detail::opencl_queue.

Referenced by kernel_end().

Here is the caller graph for this function:


cl::sycl::detail::queue ::get_context
cl::sycl::detail::queue ::kernel_end

10.18.3.4 virtual **cl::sycl::device** cl::sycl::detail::queue::get_device ( ) const `[pure virtual]`

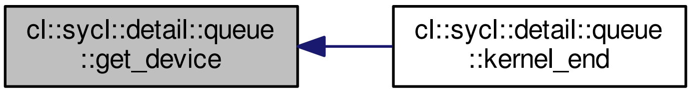
Return the SYCL device the queue is associated with.

Report errors using SYCL exception classes.

Implemented in cl::sycl::detail::host_queue, and cl::sycl::detail::opencl_queue.

Referenced by kernel_end().

Here is the caller graph for this function:


cl::sycl::detail::queue ::get_device
cl::sycl::detail::queue ::kernel_end

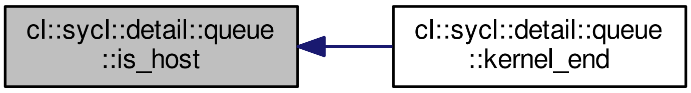
10.18.3.5 virtual **bool** cl::sycl::detail::queue::is_host ( ) const `[pure virtual]`

Return whether the queue is executing on a SYCL host device.

Implemented in cl::sycl::detail::host_queue, and cl::sycl::detail::opencl_queue.

Referenced by kernel_end().

Here is the caller graph for this function:


cl::sycl::detail::queue ::is_host
cl::sycl::detail::queue ::kernel_end

10.18.3.6 void cl::sycl::detail::queue::kernel_end ( ) [inline]

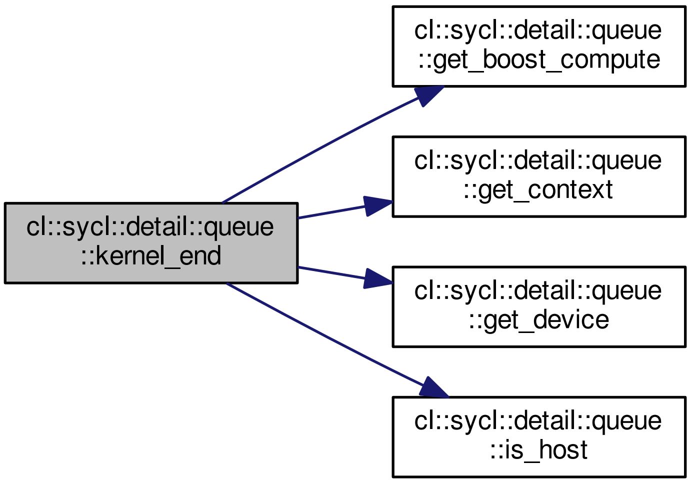
Signal that a new kernel finished on this queue.

Definition at line 66 of file queue.hpp.

References get_boost_compute(), get_context(), get_device(), is_host(), and TRISYCL_DUMP_T.

```
                    {
    TRISYCL_DUMP_T("A kernel of the queue ended");
    if (--running_kernels == 0) {
      /* It was the last kernel running, so signal the queue just in
         case it was working for it for completion */
      finished.notify_one();
    }
  }
```

Here is the call graph for this function:


cl::sycl::detail::queue ::kernel_end
cl::sycl::detail::queue ::get_boost_compute
cl::sycl::detail::queue ::get_context
cl::sycl::detail::queue ::get_device
cl::sycl::detail::queue ::is_host

10.18.3.7 void cl::sycl::detail::queue::kernel_start ( ) [inline]

Signal that a new kernel started on this queue.

Definition at line 58 of file queue.hpp.

References running_kernels, and TRISYCL_DUMP_T.

```
                      {
    TRISYCL_DUMP_T("A kernel has been added to the queue");
    // One more kernel
    ++running_kernels;
  }
```

10.18.3.8 void cl::sycl::detail::queue::wait_for_kernel_execution ( ) [inline]

Wait for all kernel completion.

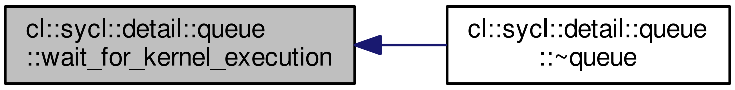
Definition at line 47 of file queue.hpp.

References TRISYCL_DUMP_T.

Referenced by ∼queue().

```
                                    {
    TRISYCL_DUMP_T("Queue waiting for kernel completion");
    std::unique_lock<std::mutex> ul { finished_mutex };
    finished.wait(ul, [&] {
        // When there is no kernel running in this queue, we are ready to go
        return running_kernels == 0;
      });
  }
```

Here is the caller graph for this function:



10.18.4 Member Data Documentation

10.18.4.1 std::condition_variable cl::sycl::detail::queue::finished

To signal when all the kernels have completed.

Definition at line 35 of file queue.hpp.

10.18.4.2 std::mutex cl::sycl::detail::queue::finished_mutex

To protect the access to the condition variable.

Definition at line 37 of file queue.hpp.

10.18.4.3 std::atomic<size_t> cl::sycl::detail::queue::running_kernels

Track the number of kernels still running to wait for their completion.

Definition at line 32 of file queue.hpp.

Referenced by kernel_start().

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

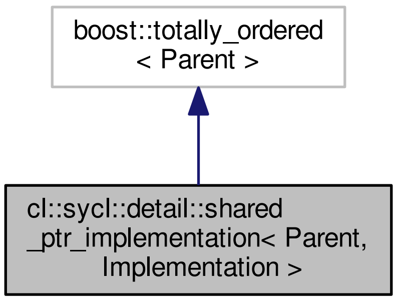
- include/CL/sycl/queue/detail/queue.hpp

10.19 cl::sycl::detail::shared_ptr_implementation< Parent, Implementation > Struct Template Reference

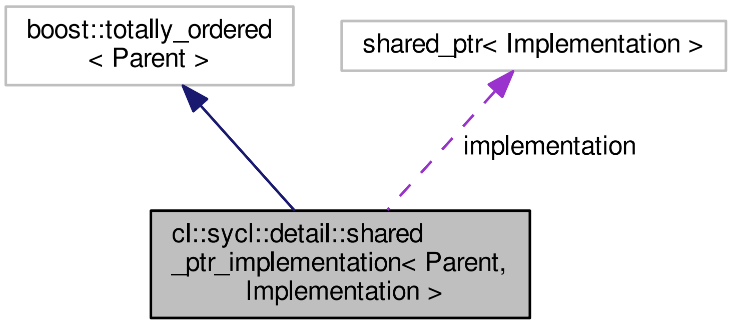
Provide an implementation as shared_ptr with total ordering and hashing to be used with algorithms and in (un)ordered containers.

`#include <shared_ptr_implementation.hpp>`

Inheritance diagram for cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >:


boost::totally_ordered < Parent >
cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >

Collaboration diagram for cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >:


boost::totally_ordered < Parent >
shared_ptr< Implementation >
implementation
cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >

Public Member Functions

- shared_ptr_implementation (std::shared_ptr< Implementation > i)

  The implementation directly as a shared pointer.
- shared_ptr_implementation (Implementation ∗i)

  The implementation takes the ownership from a raw pointer.

- shared_ptr_implementation ()=default

  Keep all other constructors to have usual shared_ptr behaviour.
- bool operator== (const Parent &other) const

  Equality operator.
- bool operator< (const Parent &other) const

  Inferior operator.
- auto hash () const

  Forward the hashing for unordered containers to the implementation.

Public Attributes

- std::shared_ptr< Implementation > implementation

  The implementation forward everything to this... implementation.

10.19.1 Detailed Description

template<typename Parent, typename Implementation>
struct cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >

Provide an implementation as shared_ptr with total ordering and hashing to be used with algorithms and in (un)ordered containers.

To be used, a Parent class wanting an Implementation needs to inherit from.

The implementation ends up in a member really named "implementation".

```
public detail::shared_ptr_implementation<Parent, Implementation>
```

and also inject in std namespace a specialization for

```
hash<Parent>
```

Definition at line 40 of file shared_ptr_implementation.hpp.

10.19.2 Constructor & Destructor Documentation

10.19.2.1 template<typename Parent, typename Implementation> cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::shared_ptr_implementation (std::shared_ptr< Implementation > *i*) [inline]

The implementation directly as a shared pointer.

Definition at line 48 of file shared_ptr_implementation.hpp.

```
00049     : implementation { i } {}
```

10.19.2.2 template<typename Parent, typename Implementation> cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::shared_ptr_implementation (Implementation ∗ *i*) [inline]

The implementation takes the ownership from a raw pointer.

Definition at line 53 of file shared_ptr_implementation.hpp.

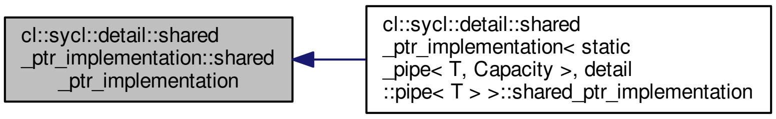
```
00053 : implementation { i } {}
```

10.19.2.3 template<typename Parent, typename Implementation> cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::shared_ptr_implementation ( ) [default]

Keep all other constructors to have usual shared_ptr behaviour.

Referenced by cl::sycl::detail::shared_ptr_implementation< static_pipe< T, Capacity >, detail::pipe< T > >↩ ::shared_ptr_implementation().

Here is the caller graph for this function:


cl::sycl::detail::shared
_ptr_implementation::shared
_ptr_implementation
cl::sycl::detail::shared
_ptr_implementation< static
_pipe< T, Capacity >, detail
::pipe< T > >::shared_ptr_implementation

10.19.3 Member Function Documentation

10.19.3.1 template<typename Parent, typename Implementation> auto cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash ( ) const [inline]

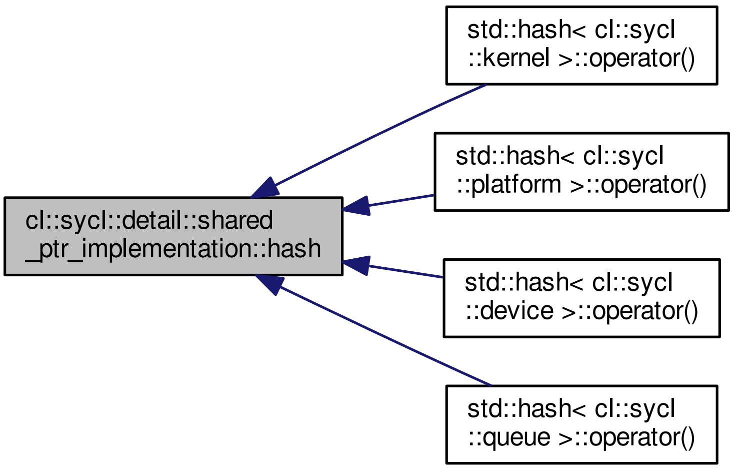
Forward the hashing for unordered containers to the implementation.

Definition at line 85 of file shared_ptr_implementation.hpp.

Referenced by std::hash< cl::sycl::kernel >::operator()(), std::hash< cl::sycl::platform >::operator()(), std::hash< cl::sycl::device >::operator()(), and std::hash< cl::sycl::queue >::operator()().

```
00085                      {
00086    return std::hash<decltype(implementation)>{}(implementation);
00087  }
```

Here is the caller graph for this function:


std::hash< cl::sycl ::kernel >::operator()
std::hash< cl::sycl ::platform >::operator()
cl::sycl::detail::shared _ptr_implementation::hash
std::hash< cl::sycl ::device >::operator()
std::hash< cl::sycl ::queue >::operator()

10.19.3.2 template<typename Parent, typename Implementation> bool cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::operator< (const Parent & *other*) const [inline]

Inferior operator.

This is generalized by boost::less_than_comparable from boost::totally_ordered to implement the equality comparable concept

Todo Add this to the spec

Definition at line 79 of file shared_ptr_implementation.hpp.

```
                                                 {
    return implementation < other.implementation;
  }
```

10.19.3.3 template<typename Parent, typename Implementation> bool cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::operator== (const Parent & *other*) const [inline]

Equality operator.

This is generalized by boost::equality_comparable from boost::totally_ordered to implement the equality comparable concept

Definition at line 66 of file shared_ptr_implementation.hpp.

```
                                                  {
    return implementation == other.implementation;
  }
```

10.19.4 Member Data Documentation

10.19.4.1 template<typename Parent, typename Implementation> std::shared_ptr<Implementation> cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::implementation

The implementation forward everything to this... implementation.

Definition at line 43 of file shared_ptr_implementation.hpp.

Referenced by cl::sycl::detail::shared_ptr_implementation< static_pipe< T, Capacity >, detail::pipe< T > >↩
::hash().

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

- include/CL/sycl/detail/shared_ptr_implementation.hpp

10.20 cl::sycl::detail::singleton< T > Struct Template Reference

Provide a singleton factory.

```
#include <singleton.hpp>
```

Static Public Member Functions

- static std::shared_ptr< T > instance ()

  Get a singleton instance of T.

10.20.1 Detailed Description

template<typename T>
struct cl::sycl::detail::singleton< T >

Provide a singleton factory.

Definition at line 25 of file singleton.hpp.

10.20.2 Member Function Documentation

10.20.2.1 template<typename T> static std::shared_ptr<T> cl::sycl::detail::singleton< T >::instance ( ) [inline], [static]

Get a singleton instance of T.

Use a null_deleter since the singleton should not be deleted, as allocated in the static area

Definition at line 28 of file singleton.hpp.

```
00028                                     {
00029     // C++11 guaranties the static construction is thread-safe
00030     static T single;
00031     /** Use a null_deleter since the singleton should not be deleted,
00032         as allocated in the static area */
00033     static std::shared_ptr<T> sps { &single,
00034                                     boost::null_deleter {} };
00035 
00036     return sps;
00037   }
```

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

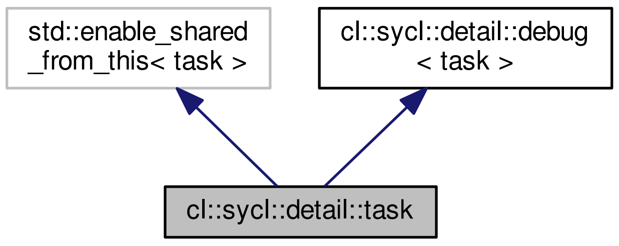
- include/CL/sycl/detail/singleton.hpp

10.21 cl::sycl::detail::task Struct Reference

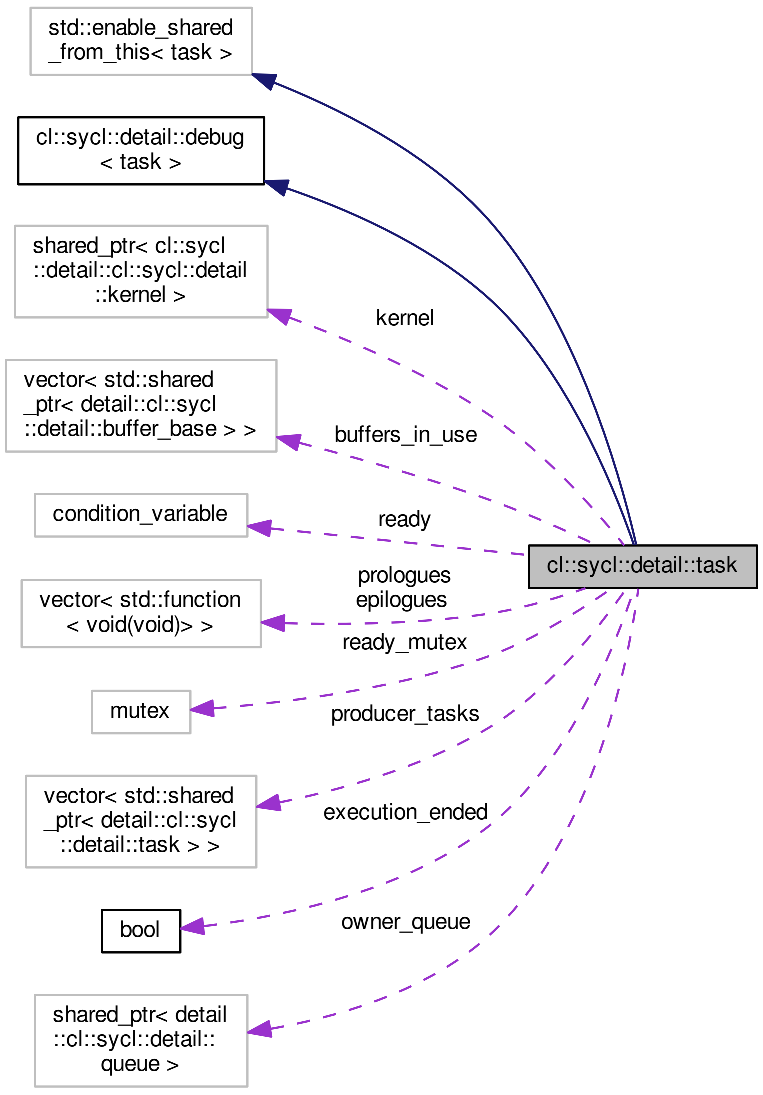
The abstraction to represent SYCL tasks executing inside command_group.

```
#include <task.hpp>
```

Inheritance diagram for cl::sycl::detail::task:



Collaboration diagram for cl::sycl::detail::task:


std::enable_shared _from_this< task >
cl::sycl::detail::debug < task >
shared_ptr< cl::sycl ::detail::cl::sycl::detail ::kernel >
kernel
vector< std::shared _ptr< detail::cl::sycl ::detail::buffer_base > >
buffers_in_use
condition_variable
ready
cl::sycl::detail::task
vector< std::function < void(void)> >
prologues
epilogues
ready_mutex
mutex
producer_tasks
vector< std::shared _ptr< detail::cl::sycl ::detail::task > >
execution_ended
bool
owner_queue
shared_ptr< detail ::cl::sycl::detail:: queue >

Public Member Functions

- task (const std::shared_ptr< detail::queue > &q)

  Create a task from a submitting queue.
- void schedule (std::function< void(void)> f)

  Add a new task to the task graph and schedule for execution.
- void wait_for_producers ()

  Wait for the required producer tasks to be ready.
- void release_buffers ()

  Release the buffers that have been used by this task.
- void notify_consumers ()

  Notify the waiting tasks that we are done.

- void wait ()

  Wait for this task to be ready.
- void add_buffer (std::shared_ptr< detail::buffer_base > &buf, bool is_write_mode)

  Register a buffer to this task.
- void prelude ()

  Execute the prologues.
- void postlude ()

  Execute the epilogues.
- void add_prelude (const std::function< void(void)> &f)

  Add a function to the prelude to run before kernel execution.
- void add_postlude (const std::function< void(void)> &f)

  Add a function to the postlude to run after kernel execution.
- auto get_queue ()

  Get the queue behind the task to run a kernel on.
- void set_kernel (const std::shared_ptr< cl::sycl::detail::kernel > &k)

  Set the kernel running this task if any.
- cl::sycl::detail::kernel & get_kernel ()

  Get the kernel running if any.

Public Attributes

- std::vector< std::shared_ptr< detail::buffer_base > > buffers_in_use

  List of the buffers used by this task.
- std::vector< std::shared_ptr< detail::task > > producer_tasks

  The tasks producing the buffers used by this task.
- std::vector< std::function< void(void)> > prologues

  Keep track of any prologue to be executed before the kernel.
- std::vector< std::function< void(void)> > epilogues

  Keep track of any epilogue to be executed after the kernel.
- bool execution_ended = false

  Store if the execution ended, to be notified by task_ready.
- std::condition_variable ready

  To signal when this task is ready.
- std::mutex ready_mutex

  To protect the access to the condition variable.
- std::shared_ptr< detail::queue > owner_queue

  Keep track of the queue used to submission to notify kernel completion or to run OpenCL kernels on.
- std::shared_ptr< cl::sycl::detail::kernel > kernel

10.21.1 Detailed Description

The abstraction to represent SYCL tasks executing inside command_group.

"enable_shared_from_this" allows to access the shared_ptr behind the scene.

Definition at line 34 of file task.hpp.

10.21.2 Constructor & Destructor Documentation

10.21.2.1 cl::sycl::detail::task::task (const std::shared_ptr< **detail::queue** > & *q*) [inline]

Create a task from a submitting queue.

Definition at line 70 of file task.hpp.

```
    : owner_queue { q } {}
```

10.21.3 Member Function Documentation

10.21.3.1 void cl::sycl::detail::task::add_buffer (std::shared_ptr< **detail::buffer_base** > & *buf*, **bool** *is_write_mode*) [inline]

Register a buffer to this task.

This is how the dependency graph is incrementally built.

Definition at line 167 of file task.hpp.

References TRISYCL_DUMP_T.

```
                                              {
    TRISYCL_DUMP_T("Add buffer " << buf << " in task " << this);
    /* Keep track of the use of the buffer to notify its release at
       the end of the execution */
    buffers_in_use.push_back(buf);
    // To be sure the buffer does not disappear before the kernel can run
    buf->use();

    std::shared_ptr<detail::task> latest_producer;
    if (is_write_mode) {
      /* Set this task as the latest producer of the buffer so that
         another kernel may wait on this task */
      latest_producer = buf->set_latest_producer(shared_from_this());
    }
    else
      latest_producer = buf->get_latest_producer();

    /* If the buffer is to be produced by a task, add the task in the
       producer list to wait on it before running the task core */
    if (latest_producer)
      producer_tasks.push_back(latest_producer);
  }
```

10.21.3.2 void cl::sycl::detail::task::add_postlude (const std::function< void(void)> & *f*) [inline]

Add a function to the postlude to run after kernel execution.

Definition at line 219 of file task.hpp.

```
                                                             {
    epilogues.push_back(f);
  }
```

10.21.3.3 void cl::sycl::detail::task::add_prelude (const std::function< void(void)> & f) [inline]

Add a function to the prelude to run before kernel execution.

Definition at line 213 of file task.hpp.

```
                                                        {
      prologues.push_back(f);
    }
```

10.21.3.4 cl::sycl::detail::kernel& cl::sycl::detail::task::get_kernel ( ) [inline]

Get the kernel running if any.

Todo Specify this error in the spec

Definition at line 240 of file task.hpp.

References kernel.

```
                                      {
      if (!kernel)
        throw non_cl_error("Cannot use an OpenCL kernel in this context");
      return *kernel;
    }
```

10.21.3.5 auto cl::sycl::detail::task::get_queue ( ) [inline]

Get the queue behind the task to run a kernel on.

Definition at line 225 of file task.hpp.

References owner_queue.

```
                      {
      return owner_queue;
    }
```

10.21.3.6 void cl::sycl::detail::task::notify_consumers ( ) [inline]

Notify the waiting tasks that we are done.

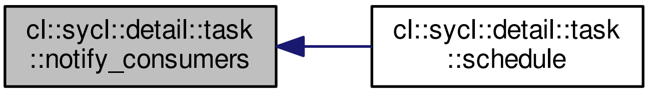
Definition at line 143 of file task.hpp.

References TRISYCL_DUMP_T.

Referenced by schedule().

```
                            {
    TRISYCL_DUMP_T("Notify all the task waiting for this task " << this);
    execution_ended = true;
    /* \todo Verify that the memory model with the notify does not
       require some fence or atomic */
    ready.notify_all();
  }
```

Here is the caller graph for this function:


cl::sycl::detail::task ::notify_consumers
cl::sycl::detail::task ::schedule

10.21.3.7 void cl::sycl::detail::task::postlude ( ) [inline]

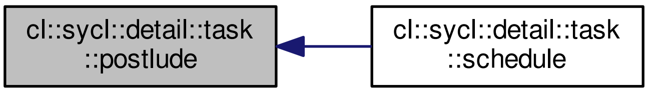
Execute the epilogues.

Definition at line 203 of file task.hpp.

Referenced by schedule().

```
                    {
    for (const auto &p : epilogues)
      p();
    /* Free the functors that may own an accessor owning a buffer
       preventing the command group to complete */
    epilogues.clear();
  }
```

Here is the caller graph for this function:


cl::sycl::detail::task ::postlude
cl::sycl::detail::task ::schedule

10.21.3.8 void cl::sycl::detail::task::prelude ( ) [inline]

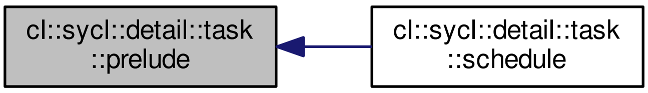
Execute the prologues.

Definition at line 193 of file task.hpp.

Referenced by schedule().

```
                  {
    for (const auto &p : prologues)
      p();
    /* Free the functors that may own an accessor owning a buffer
       preventing the command group to complete */
    prologues.clear();
  }
```

Here is the caller graph for this function:



cl::sycl::detail::task ::prelude
cl::sycl::detail::task ::schedule

10.21.3.9 void cl::sycl::detail::task::release_buffers ( ) [inline]

Release the buffers that have been used by this task.

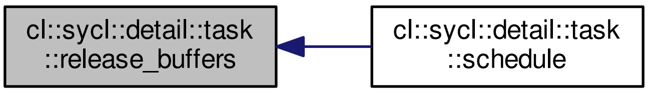
Definition at line 134 of file task.hpp.

References TRISYCL_DUMP_T.

Referenced by schedule().

```
                          {
    TRISYCL_DUMP_T("Task " << this << " releases the written buffers");
    for (auto b: buffers_in_use)
      b->release();
    buffers_in_use.clear();
  }
```

Here is the caller graph for this function:



cl::sycl::detail::task ::release_buffers
cl::sycl::detail::task ::schedule

10.21.3.10 void cl::sycl::detail::task::schedule (std::function< void(void)> *f*) [inline]

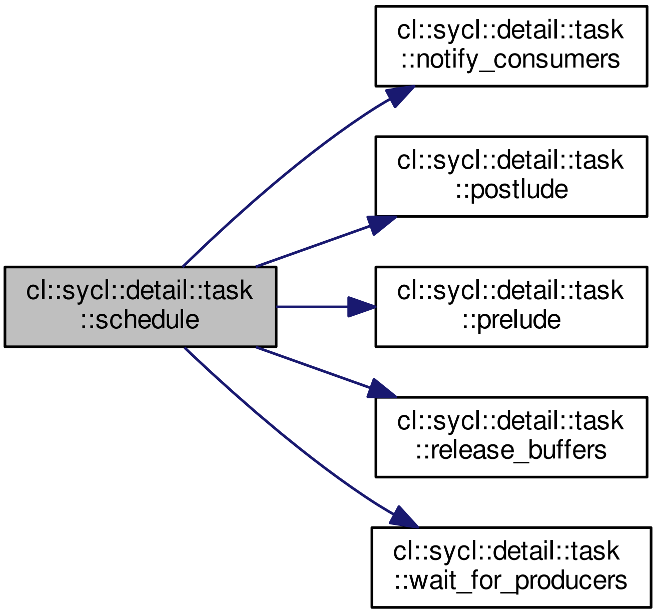
Add a new task to the task graph and schedule for execution.

Definition at line 75 of file task.hpp.

References notify_consumers(), postlude(), prelude(), release_buffers(), TRISYCL_DUMP_T, and wait_for_producers().

```
                                              {
    /* To keep a copy of the task shared_ptr after the end of the
       command group, capture it by copy in the following lambda. This
       should be easier in C++17 with move semantics on capture
    */
    auto task = shared_from_this();
    auto execution = [=] {
      // Wait for the required tasks to be ready
      task->wait_for_producers();
      task->prelude();
      TRISYCL_DUMP_T("Execute the kernel");
      // Execute the kernel
      f();
      task->postlude();
      // Release the buffers that have been written by this task
      task->release_buffers();
      // Notify the waiting tasks that we are done
      task->notify_consumers();
      // Notify the queue we are done
      owner_queue->kernel_end();
      TRISYCL_DUMP_T("Task thread exit");
    };
    /* Notify the queue that there is a kernel submitted to the
       queue. Do not do it in the task contructor so that we can deal
       with command group without kernel and if we put it inside the
       thread, the queue may have finished before the thread is
       scheduled */
    owner_queue->kernel_start();
    /* \todo it may be implementable with packaged_task that would
       deal with exceptions in kernels
    */
#if TRISYCL_ASYNC
    /* If in asynchronous execution mode, execute the functor in a new
       thread */
    std::thread thread(execution);
    TRISYCL_DUMP_T("Task thread started");
    /** Detach the thread since it will synchronize by its own means

        \todo This is an issue if there is an exception in the kernel
    */
    thread.detach();
#else
    // Just a synchronous execution otherwise
    execution();
#endif
  }
```

Here is the call graph for this function:


cl::sycl::detail::task ::schedule
cl::sycl::detail::task ::notify_consumers
cl::sycl::detail::task ::postlude
cl::sycl::detail::task ::prelude
cl::sycl::detail::task ::release_buffers
cl::sycl::detail::task ::wait_for_producers

10.21.3.11 void cl::sycl::detail::task::set_kernel (const std::shared_ptr< cl::sycl::detail::kernel > & *k*) [inline]

Set the kernel running this task if any.

Definition at line 231 of file task.hpp.

```
                                                            {
    kernel = k;
  }
```

10.21.3.12 void cl::sycl::detail::task::wait ( ) [inline]

Wait for this task to be ready.

This is to be called from another thread

Definition at line 156 of file task.hpp.

References execution_ended, and TRISYCL_DUMP_T.

```
               {
    TRISYCL_DUMP_T("The task wait for task " << this << " to end");
    std::unique_lock<std::mutex> ul { ready_mutex };
    ready.wait(ul, [&] { return execution_ended; });
  }
```

10.21.3.13 void cl::sycl::detail::task::wait_for_producers ( ) [inline]

Wait for the required producer tasks to be ready.

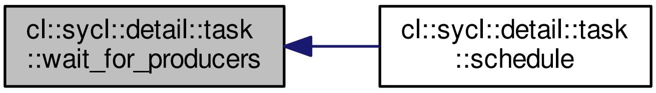
Definition at line 124 of file task.hpp.

References TRISYCL_DUMP_T.

Referenced by schedule().

```
                                {
    TRISYCL_DUMP_T("Task " << this << " waits for the producer tasks");
    for (auto &t : producer_tasks)
      t->wait();
    // We can let the producers rest in peace
    producer_tasks.clear();
  }
```

Here is the caller graph for this function:



10.21.4 Member Data Documentation

10.21.4.1 std::vector< std::shared_ptr< detail::buffer_base > > cl::sycl::detail::task::buffers_in_use

List of the buffers used by this task.

Todo Use a set to check that some buffers are not used many times at least on writing

Definition at line 42 of file task.hpp.

10.21.4.2 std::vector< std::function< void(void)> > cl::sycl::detail::task::epilogues

Keep track of any epilogue to be executed after the kernel.

Definition at line 51 of file task.hpp.

10.21.4.3 bool cl::sycl::detail::task::execution_ended = false

Store if the execution ended, to be notified by task_ready.

Definition at line 54 of file task.hpp.

Referenced by wait().

10.21.4.4 std::shared_ptr<cl::sycl::detail::kernel> cl::sycl::detail::task::kernel

Definition at line 66 of file task.hpp.

Referenced by get_kernel().

10.21.4.5 std::shared_ptr<detail::queue> cl::sycl::detail::task::owner_queue

Keep track of the queue used to submission to notify kernel completion or to run OpenCL kernels on.

Definition at line 64 of file task.hpp.

Referenced by get_queue().

10.21.4.6 std::vector<std::shared_ptr<detail::task> > cl::sycl::detail::task::producer_tasks

The tasks producing the buffers used by this task.

Definition at line 45 of file task.hpp.

10.21.4.7 std::vector<std::function<void(void)> > cl::sycl::detail::task::prologues

Keep track of any prologue to be executed before the kernel.

Definition at line 48 of file task.hpp.

10.21.4.8 std::condition_variable cl::sycl::detail::task::ready

To signal when this task is ready.

Definition at line 57 of file task.hpp.

10.21.4.9 std::mutex cl::sycl::detail::task::ready_mutex

To protect the access to the condition variable.

Definition at line 60 of file task.hpp.

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

- include/CL/sycl/command_group/detail/task.hpp

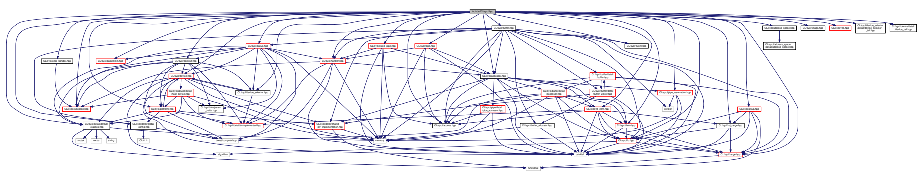
Chapter 11

File Documentation

11.1 include/CL/sycl.hpp File Reference

```
#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/address_space.hpp"
#include "CL/sycl/buffer.hpp"
#include "CL/sycl/context.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/error_handler.hpp"
#include "CL/sycl/event.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/group.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/image.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/parallelism.hpp"
#include "CL/sycl/pipe.hpp"
#include "CL/sycl/pipe_reservation.hpp"
#include "CL/sycl/platform.hpp"
#include "CL/sycl/queue.hpp"
#include "CL/sycl/range.hpp"
#include "CL/sycl/static_pipe.hpp"
#include "CL/sycl/vec.hpp"
#include "CL/sycl/device_selector/detail/device_selector_tail.hpp"
#include "CL/sycl/device/detail/device_tail.hpp"
```

Include dependency graph for sycl.hpp:

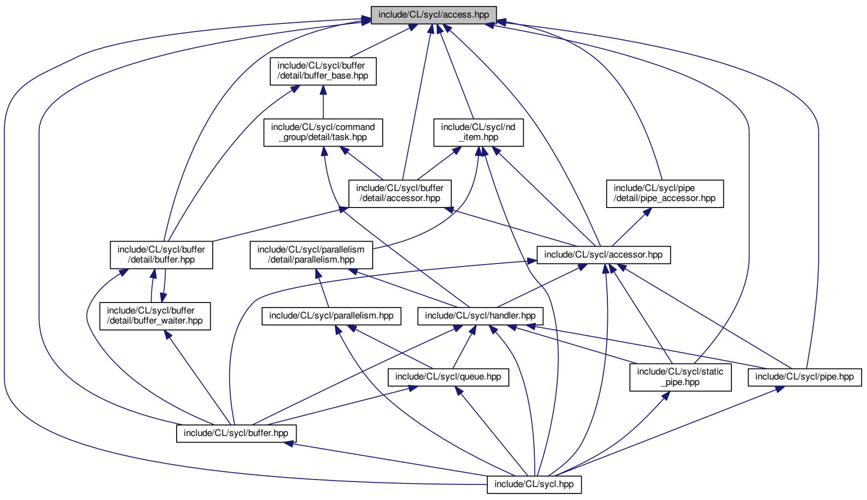


11.2 sycl.hpp

```
/** \file

    \mainpage

    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/sycl/

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

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


    Ronan at keryell dot FR

    Copyright 2014--2015 Advanced Micro Devices, Inc.

    Copyright 2015--2016 Xilinx, Inc.

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/


/** Some global triSYCL configuration */
#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/detail/default_classes.hpp"


/* All the SYCL components, one per file */
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/address_space.hpp"
#include "CL/sycl/buffer.hpp"
#include "CL/sycl/context.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/error_handler.hpp"
#include "CL/sycl/event.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/group.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/image.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/parallelism.hpp"
#include "CL/sycl/pipe.hpp"
#include "CL/sycl/pipe_reservation.hpp"
#include "CL/sycl/platform.hpp"
#include "CL/sycl/queue.hpp"
#include "CL/sycl/range.hpp"
#include "CL/sycl/static_pipe.hpp"
#include "CL/sycl/vec.hpp"

// Some includes at the end to break some dependencies
#include "CL/sycl/device_selector/detail/device_selector_tail.hpp
"
#include "CL/sycl/device/detail/device_tail.hpp"

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/
```

11.3 include/CL/sycl/access.hpp File Reference

This graph shows which files directly or indirectly include this file:


include/CL/sycl/access.hpp
include/CL/sycl/buffer/detail/buffer_base.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/nd_item.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/pipe/detail/pipe_accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/parallelism/detail/parallelism.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/parallelism.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::access

  Describe the type of access by kernels.

Enumerations

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

  This describes the type of the access mode to be used via accessor.
- enum cl::sycl::access::target {
  cl::sycl::access::target::global_buffer = 2014, cl::sycl::access::target::constant_buffer, cl::sycl::access↩
  ::target::local, cl::sycl::access::target::image,
  cl::sycl::access::target::host_buffer, cl::sycl::access::target::host_image, cl::sycl::access::target::image_↩
  array, cl::sycl::access::target::pipe,
  cl::sycl::access::target::blocking_pipe }

  The target enumeration describes the type of object to be accessed via the accessor.
- enum cl::sycl::access::fence_space : char { cl::sycl::access::fence_space::local_space, cl::sycl::access↩
  ::fence_space::global_space, cl::sycl::access::fence_space::global_and_local }

  Precise the address space a barrier needs to act on.

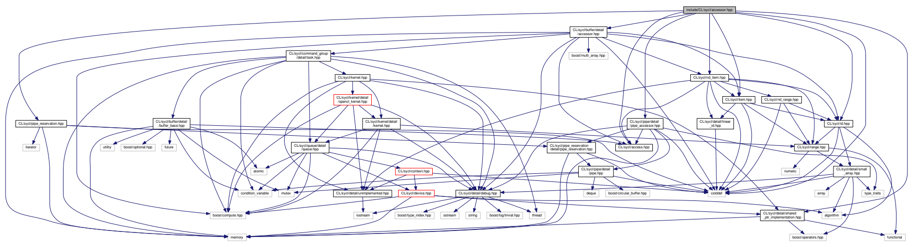
11.4 access.hpp

```
#ifndef TRISYCL_SYCL_ACCESS_HPP
#define TRISYCL_SYCL_ACCESS_HPP

/** \file The OpenCL SYCL access naming space

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

// SYCL dwells in the cl::sycl namespace
namespace cl {
namespace sycl {

/** \addtogroup data Data access and storage in SYCL

    @{
*/

/** Describe the type of access by kernels.

    \todo This values should be normalized to allow separate compilation
    with different implementations?
*/
namespace access {
  /* By using "enum mode" here instead of "enum struct mode", we have for
     example "write" appearing both as cl::sycl::access::mode::write and
     cl::sycl::access::write, instead of only the last one. This seems
     more conform to the specification. */

  /// This describes the type of the access mode to be used via accessor
  enum class mode {
    read = 42, /**< Read-only access. Insist on the fact that
                    read_write != read + write */
    write, ///< Write-only access, but previous content *not* discarded
    read_write, ///< Read and write access
    discard_write, ///< Write-only access and previous content discarded
    discard_read_write, /**< Read and write access and previous
                             content discarded*/
    atomic ///< Atomic access
  };


  /** The target enumeration describes the type of object to be accessed
      via the accessor
  */
  enum class target {
    global_buffer = 2014, //< Just pick a random number...
    constant_buffer,
    local,
    image,
    host_buffer,
    host_image,
    image_array,
    pipe,
    blocking_pipe
  };


  /** Precise the address space a barrier needs to act on
  */
  enum class fence_space : char {
    local_space,
    global_space,
    global_and_local
  };

}

/// @} End the data Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ACCESS_HPP
```

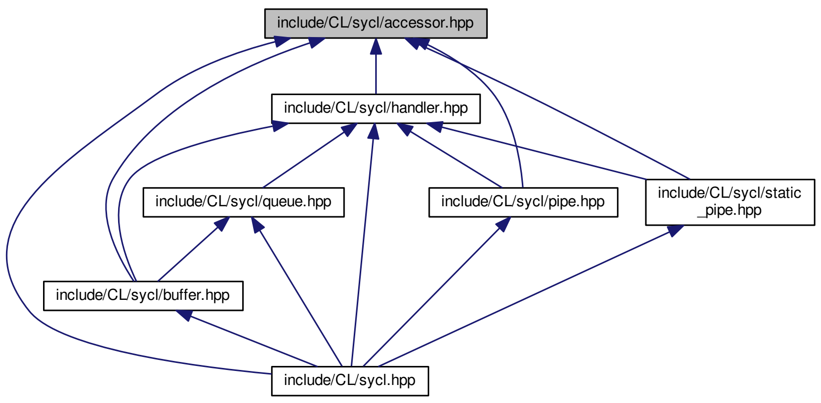
11.5 include/CL/sycl/accessor.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/access.hpp"
#include "CL/sycl/buffer/detail/accessor.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
#include "CL/sycl/pipe_reservation.hpp"
#include "CL/sycl/pipe/detail/pipe_accessor.hpp"
```

Include dependency graph for accessor.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/accessor.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static
_pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::buffer< T, Dimensions, Allocator >

  < T, Dimensions, Mode, Target> up data Data access and storage in SYCL
- class cl::sycl::pipe< T >

  A SYCL pipe. More...
- class cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >

  The accessor abstracts the way buffer or pipe data are accessed inside a kernel in a multidimensional variable length array way. More...
- class cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >

  The pipe accessor abstracts the way pipe data are accessed inside a kernel. More...
- class cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >

  The pipe accessor abstracts the way pipe data are accessed inside a kernel. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Functions

- template<typename Accessor >
  static auto & cl::sycl::get_pipe_detail (Accessor &a)

  Top-level function to break circular dependencies on the the types to get the pipe implementation.

11.6 accessor.hpp

```
#ifndef TRISYCL_SYCL_ACCESSOR_HPP
#define TRISYCL_SYCL_ACCESSOR_HPP

/** \file The OpenCL SYCL accessor<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>

#include "CL/sycl/access.hpp"
#include "CL/sycl/buffer/detail/accessor.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
#include "CL/sycl/pipe_reservation.hpp"
#include "CL/sycl/pipe/detail/pipe_accessor.hpp"

namespace cl {
namespace sycl {

template <typename T, std::size_t Dimensions, typename Allocator>
class buffer;
template <typename T>
class pipe;
class handler;

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** The accessor abstracts the way buffer or pipe 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
*/
template <typename DataType,
          std::size_t Dimensions,
          access::mode AccessMode,
          access::target Target = access::target::global_buffer>
class accessor :
    public detail::shared_ptr_implementation<accessor<DataType,
                                                      Dimensions,
                                                      AccessMode,
                                                      Target>,
                                             detail::accessor<DataType,
                                                              Dimensions,
                                                              AccessMode,
                                                              Target>> {
public:

  /// \todo in the specification: store the dimension for user request
  static constexpr auto dimensionality = Dimensions;
  using value_type = DataType;
  using reference = value_type&;
  using const_reference = const value_type&;
```

```

private:

  using accessor_detail = detail::accessor<DataType,
                                           Dimensions,
                                           AccessMode,
                                           Target>;

  // The type encapsulating the implementation
  using implementation_t =
    detail::shared_ptr_implementation<accessor<DataType,
                                               Dimensions,
                                               AccessMode,
                                               Target>,
                                      accessor_detail>;

 public:

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

  /** Construct a buffer accessor from a buffer using a command group
      handler object from the command group scope

      Constructor only available for global_buffer or constant_buffer
      target.

      access_target defines the form of access being obtained.

      \todo Add template allocator type in all the accessor
      constructors in the specification or just use a more opaque
      Buffer type?

      \todo fix specification where access mode should be target
      instead
  */
  template <typename Allocator>
  accessor(buffer<DataType, Dimensions, Allocator> &
target_buffer,
           handler &command_group_handler) : implementation_t {
    new detail::accessor<DataType, Dimensions, AccessMode, Target>
 {
      target_buffer.implementation->implementation, command_group_handler }
  } {
    static_assert(Target == access::target::global_buffer
                  || Target == access::target::constant_buffer,
                  "access target should be global_buffer or constant_buffer "
                  "when a handler is used");
  }


  /** Construct a buffer accessor from a buffer using a command group
      handler object from the command group scope

      Constructor only available for host_buffer target.

      access_target defines the form of access being obtained.

      \todo add this lacking constructor to specification
  */
  template <typename Allocator>
  accessor(buffer<DataType, Dimensions, Allocator> &
target_buffer)
    : implementation_t {
    new detail::accessor<DataType, Dimensions, AccessMode, Target>
 {
      target_buffer.implementation->implementation }
  } {
    static_assert(Target == access::target::host_buffer,
                  "without a handler, access target should be host_buffer");
  }


  /** Construct a buffer accessor from a buffer given a specific range for
      access permissions and an offset that provides the starting point
      for the access range using a command group handler object from the
      command group scope

      This accessor limits the processing of the buffer to the [offset,
      offset+range[ for every dimension. Any other parts of the buffer
      will be unaffected.

      Constructor only available for access modes global_buffer,
      host_buffer or constant_buffer (see Table 3.25). access_target
      defines the form of access being obtained (see Table 3.26).

      This accessor is recommended for discard-write and discard read
```

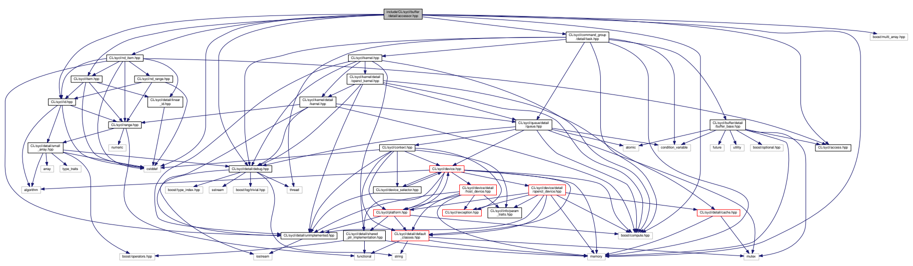
```
     write access modes, when the unaffected parts of the processing
     should be retained.
  */
  template <typename Allocator>
  accessor(buffer<DataType, Dimensions, Allocator> &
target_buffer,
           handler &command_group_handler,
           range<Dimensions> offset,
           range<Dimensions> range) {
    detail::unimplemented();
  }


  /** Construct an accessor of dimensions Dimensions with elements of type
      DataType using the passed range to specify the size in each
      dimension

      It needs as a parameter a command group handler object from the
      command group scope. Constructor only available if AccessMode is
      local, see Table 3.25.
  */
  accessor(range<Dimensions> allocation_size,
           handler &command_group_handler) {
    detail::unimplemented();
  }


  /** Use the accessor with integers à la [][][]

      Use array_view_type::reference instead of auto& because it does not
      work in some dimensions.
   */
  typename accessor_detail::reference operator[](std::size_t index) {
    return (*implementation)[index];
  }


  /** Use the accessor with integers à la [][][]

      Use array_view_type::reference instead of auto& because it does not
      work in some dimensions.
   */
  typename accessor_detail::reference operator[](std::size_t index)
 const {
    return (*implementation)[index];
  }


  /// To use the accessor with [id<>]
  auto &operator[](id<dimensionality> index) {
    return (*implementation)[index];
  }


  /// To use the accessor with [id<>]
  auto &operator[](id<dimensionality> index) const {
    return (*implementation)[index];
  }


  /// To use an accessor with [item<>]
  auto &operator[](item<dimensionality> index) {
    return (*this)[index.get()];
  }


  /// To use an accessor with [item<>]
  auto &operator[](item<dimensionality> index) const {
    return (*this)[index.get()];
  }


  /** To use an accessor with an [nd_item<>]

      \todo Add in the specification because used by HPC-GPU slide 22
  */
  auto &operator[](nd_item<dimensionality> index) {
    return (*this)[index.get_global()];
  }

  /** To use an accessor with an [nd_item<>]

      \todo Add in the specification because used by HPC-GPU slide 22
  */
  auto &operator[](nd_item<dimensionality> index) const {
    return (*this)[index.get_global()];
  }
```

```


    /** Get the first element of the accessor

      Useful with an accessor on a scalar for example.

      \todo Add in the specification
  */
  typename accessor_detail::reference operator*() {
    return **implementation;
  }


  /** Get the first element of the accessor

      Useful with an accessor on a scalar for example.

      \todo Add in the specification?

      \todo Add the concept of 0-dim buffer and accessor for scalar
      and use an implicit conversion to value_type reference to access
      the value with the accessor?
  */
  typename accessor_detail::reference operator*() const {
    return **implementation;
  }

  /** Forward all the iterator functions to the implementation

      \todo Add these functions to the specification

      \todo The fact that the lambda capture make a const copy of the
      accessor is not yet elegantly managed... The issue is that
      begin()/end() dispatch is made according to the accessor
      constness and not from the array member constness...

      \todo try to solve it by using some enable_if on array
      constness?

      \todo The issue is that the end may not be known if it is
      implemented by a raw OpenCL cl_mem... So only provide on the
      device the iterators related to the start? Actually the accessor
      needs to know a part of the shape to have the multidimentional
      addressing. So this only require a size_t more...

      \todo Factor out these in a template helper
  */


  // iterator begin() { return array.begin(); }
  typename accessor_detail::iterator begin() const {
    return implementation->begin();
  }


  // iterator end() { return array.end(); }
  typename accessor_detail::iterator end() const {
    return implementation->end();
  }


  // const_iterator begin() const { return implementation->begin(); }


  // const_iterator end() const { return implementation->end(); }


  typename accessor_detail::const_iterator cbegin() const {
    return implementation->cbegin();
  }


  typename accessor_detail::const_iterator cend() const {
    return implementation->cend();
  }


  typename accessor_detail::reverse_iterator
rbegin() const {
    return implementation->rbegin();
  };


  typename accessor_detail::reverse_iterator
rend() const {
    return implementation->rend();
  }


  // const_reverse_iterator rbegin() const { return array.rbegin(); }


  // const_reverse_iterator rend() const { return array.rend(); }


  typename accessor_detail::const_reverse_iterator
crbegin() const {
    return implementation->rbegin();
  }


  typename accessor_detail::const_reverse_iterator
crend() const {
    return implementation->rend();
  }

};


/** The pipe accessor abstracts the way pipe data are accessed inside
    a kernel

    A specialization for an non-blocking pipe
*/
template <typename DataType,
          access::mode AccessMode>
class accessor<DataType, 1, AccessMode, access::target::pipe> :
    public detail::pipe_accessor<DataType, AccessMode, access::target::pipe> {
public:

  using accessor_detail =
    detail::pipe_accessor<DataType, AccessMode, access::target::pipe>
;
  // Inherit of the constructors to have accessor constructor from detail
  using accessor_detail::accessor_detail;

  /** Construct a pipe accessor from a pipe using a command group
      handler object from the command group scope

      access_target defines the form of access being obtained.
  */
  accessor(pipe<DataType> &p, handler &command_group_handler)
    : accessor_detail { p.implementation, command_group_handler } { }

  /// Make a reservation inside the pipe
  pipe_reservation<accessor> reserve(std::size_t size) const {
    return accessor_detail::reserve(size);
  }


  /// Get the underlying pipe implementation
  auto &get_pipe_detail() {
    return accessor_detail::get_pipe_detail();
  }

};


/** The pipe accessor abstracts the way pipe data are accessed inside
    a kernel

    A specialization for a blocking pipe
*/
template <typename DataType,
          access::mode AccessMode>
class accessor<DataType, 1, AccessMode, access::target::blocking_pipe> :
    public detail::pipe_accessor<DataType, AccessMode, access::target::blocking_pipe>
 {
public:

  using accessor_detail =
    detail::pipe_accessor<DataType, AccessMode, access::target::blocking_pipe>
;
  // Inherit of the constructors to have accessor constructor from detail
  using accessor_detail::accessor_detail;

  /** Construct a pipe accessor from a pipe using a command group
      handler object from the command group scope

      access_target defines the form of access being obtained.
  */
  accessor(pipe<DataType> &p, handler &command_group_handler)
    : accessor_detail { p.implementation, command_group_handler } { }


  /// Make a reservation inside the pipe
  pipe_reservation<accessor> reserve(std::size_t size) const {
    return accessor_detail::reserve(size);
  }


  /// Get the underlying pipe implementation
  auto &get_pipe_detail() {
    return accessor_detail::get_pipe_detail();
  }

};


/** Top-level function to break circular dependencies on the the types
    to get the pipe implementation */
template <typename Accessor>
static inline auto &get_pipe_detail(Accessor &a) {
  return a.get_pipe_detail();
  }

/// @} End the data Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ACCESSOR_HPP
```

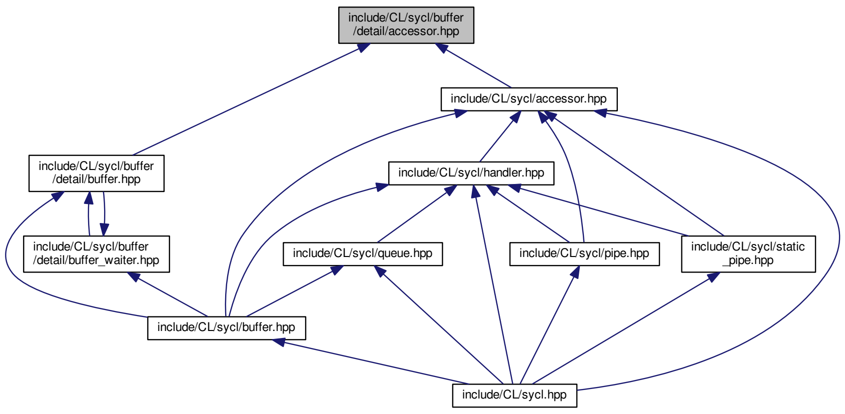
11.7 include/CL/sycl/buffer/detail/accessor.hpp File Reference

```
#include <cstddef>
#include <memory>
#include <boost/compute.hpp>
#include <boost/multi_array.hpp>
#include "CL/sycl/access.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
```

Include dependency graph for accessor.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::detail::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...
- class cl::sycl::detail::accessor< T, Dimensions, Mode, Target >

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

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.8 accessor.hpp

```
#ifndef TRISYCL_SYCL_ACCESSOR_DETAIL_ACCESSOR_HPP
#define TRISYCL_SYCL_ACCESSOR_DETAIL_ACCESSOR_HPP

/** \file The OpenCL SYCL buffer accessor<> detail behind the scene

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <memory>

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif
#include <boost/multi_array.hpp>

#include "CL/sycl/access.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/debug.hpp"
```

```
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"

namespace cl {
namespace sycl {

class handler;

namespace detail {

// Forward declaration of detail::buffer for use in accessor
template <typename T, std::size_t Dimensions> class buffer;

/** \addtogroup data Data access and storage in SYCL
    @{
*/

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

    This implementation relies on boost::multi_array to provide this
    nice syntax and behaviour.

    Right now the aim of this class is just to access to the buffer in
    a read-write mode, even if capturing the multi_array_ref from a
    lambda make it const (since in examples we have lambda with [=]
    without mutable lambda).

    \todo Use the access::mode
*/
template <typename T,
          std::size_t Dimensions,
          access::mode Mode,
          access::target Target /* = access::global_buffer */>
class accessor : public detail::debug<accessor<T,
                                               Dimensions,
                                               Mode,
                                               Target>> {
  /** Keep a reference to the accessed buffer

      Beware that it owns the buffer, which means that the accessor
      has to be destroyed to release the buffer and potentially
      unblock a kernel at the end of its execution
  */
  std::shared_ptr<detail::buffer<T, Dimensions>> buf;

  /// The implementation is a multi_array_ref wrapper
  using array_view_type = boost::multi_array_ref<T, Dimensions>;

  // The same type but writable
  using writable_array_view_type =
    typename std::remove_const<array_view_type>::type;

  /** The way the buffer is really accessed

      Use a mutable member because the accessor needs to be captured
      by value in the lambda which is then read-only. This is to avoid
      the user to use mutable lambda or have a lot of const_cast as
      previously done in this implementation
   */
  mutable array_view_type array;

  /// The task where the accessor is used in
  std::shared_ptr<detail::task> task;

#ifdef TRISYCL_OPENCL
  /// The OpenCL buffer used by an OpenCL accessor
  boost::optional<boost::compute::buffer> cl_buf;
#endif

public:

  /** \todo in the specification: store the dimension for user request

      \todo Use another name, such as from C++17 committee discussions.
   */
  static constexpr auto dimensionality = Dimensions;

  /** \todo in the specification: store the types for user request as STL
      or C++AMP */
  using value_type = T;
  using element = T;
  using reference = typename array_view_type::reference;
  using const_reference = typename array_view_type::const_reference;

  /** Inherit the iterator types from the implementation
```

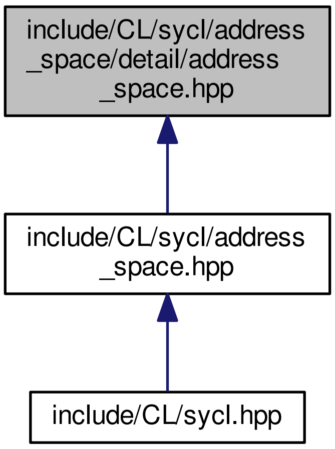
```

     \todo Add iterators to accessors in the specification
  */
  using iterator = typename array_view_type::iterator;
  using const_iterator = typename array_view_type::const_iterator;
  using reverse_iterator = typename array_view_type::reverse_iterator;
  using const_reverse_iterator =
    typename array_view_type::const_reverse_iterator;


  /** Construct a host accessor from an existing buffer

      \todo fix the specification to rename target that shadows
      template parm
  */
  accessor(std::shared_ptr<detail::buffer<T, Dimensions>>
target_buffer) :
    buf { target_buffer }, array { target_buffer->access } {
    TRISYCL_DUMP_T("Create a host accessor write = " <<
is_write_access());
    static_assert(Target == access::target::host_buffer,
                  "without a handler, access target should be host_buffer");
    /* The host needs to wait for all the producers of the buffer to
       have finished */
    buf->wait();
  }


  /** Construct a device accessor from an existing buffer

      \todo fix the specification to rename target that shadows
      template parm
  */
  accessor(std::shared_ptr<detail::buffer<T, Dimensions>>
target_buffer,
           handler &command_group_handler) :
    buf { target_buffer }, array { target_buffer->access } {
    TRISYCL_DUMP_T("Create a kernel accessor write = " <<
is_write_access());
    static_assert(Target == access::target::global_buffer
                  || Target == access::target::constant_buffer,
                  "access target should be global_buffer or constant_buffer "
                  "when a handler is used");
    // Register the buffer to the task dependencies
    task = buffer_add_to_task(buf, &command_group_handler,
is_write_access());
  }


  /** Returns the size of the underlying buffer in number of elements

      \todo It is incompatible with buffer get_size() in the spec
  */
  std::size_t get_size() const {
    return array.num_elements();
  }


  /** Use the accessor with integers à la [][][]

      Use array_view_type::reference instead of auto& because it does not
      work in some dimensions.
   */
  reference operator[](std::size_t index) {
    return array[index];
  }


  /** Use the accessor with integers à la [][][]

      Use array_view_type::reference instead of auto& because it does not
      work in some dimensions.
   */
  reference operator[](std::size_t index) const {
    return array[index];
  }


  /// To use the accessor with [id<>]
  auto &operator[](id<dimensionality> index) {
    return array(index);
  }


  /// To use the accessor with [id<>]
  auto &operator[](id<dimensionality> index) const {
    return array(index);
  }


  /// To use an accessor with [item<>]
  auto &operator[](item<dimensionality> index) {
    return (*this)[index.get()];
  }


  /// To use an accessor with [item<>]
  auto &operator[](item<dimensionality> index) const {
    return (*this)[index.get()];
  }


  /** To use an accessor with an [nd_item<>]

      \todo Add in the specification because used by HPC-GPU slide 22
  */
  auto &operator[](nd_item<dimensionality> index) {
    return (*this)[index.get_global()];
  }

  /** To use an accessor with an [nd_item<>]

      \todo Add in the specification because used by HPC-GPU slide 22
  */
  auto &operator[](nd_item<dimensionality> index) const {
    return (*this)[index.get_global()];
  }


  /** Get the first element of the accessor

      Useful with an accessor on a scalar for example.

      \todo Add in the specification
  */
  reference operator*() {
    return *array.data();
  }


  /** Get the first element of the accessor

      Useful with an accessor on a scalar for example.

      \todo Add in the specification?

      \todo Add the concept of 0-dim buffer and accessor for scalar
      and use an implicit conversion to value_type reference to access
      the value with the accessor?
  */
  reference operator*() const {
    return *array.data();
  }


  /// Get the buffer used to create the accessor
  detail::buffer<T, Dimensions> &get_buffer() {
    return *buf;
  }


  /** Test if the accessor has a read access right

      \todo Strangely, it is not really constexpr because it is not a
      static method...

      \todo to move in the access::mode enum class and add to the
      specification ?
  */
  constexpr bool is_read_access() const {
    return Mode == access::mode::read
      || Mode == access::mode::read_write
      || Mode == access::mode::discard_read_write;
  }


  /** Test if the accessor has a write access right

      \todo Strangely, it is not really constexpr because it is not a
      static method...

      \todo to move in the access::mode enum class and add to the
      specification ?
  */
  constexpr bool is_write_access() const {
    return Mode == access::mode::write
      || Mode == access::mode::read_write
      || Mode == access::mode::discard_write
      || Mode == access::mode::discard_read_write;
  }


  /** Forward all the iterator functions to the implementation

      \todo Add these functions to the specification

      \todo The fact that the lambda capture make a const copy of the
      accessor is not yet elegantly managed... The issue is that
      begin()/end() dispatch is made according to the accessor
      constness and not from the array member constness...

      \todo try to solve it by using some enable_if on array
      constness?

      \todo The issue is that the end may not be known if it is
      implemented by a raw OpenCL cl_mem... So only provide on the
      device the iterators related to the start? Actually the accessor
      needs to know a part of the shape to have the multidimentional
      addressing. So this only require a size_t more...

      \todo Factor out these in a template helper

      \todo Do we need this in detail::accessor too or only in accessor?
  */


  // iterator begin() { return array.begin(); }
  iterator begin() const {
    return const_cast<writable_array_view_type &>(array).
begin();
  }


  // iterator end() { return array.end(); }
  iterator end() const {
    return const_cast<writable_array_view_type &>(array).
end();
  }


  // const_iterator begin() const { return array.begin(); }


  // const_iterator end() const { return array.end(); }


  const_iterator cbegin() const { return array.begin(); }


  const_iterator cend() const { return array.end(); }


  // reverse_iterator rbegin() { return array.rbegin(); }
  reverse_iterator rbegin() const {
    return const_cast<writable_array_view_type &>(array).
rbegin();
  }


  // reverse_iterator rend() { return array.rend(); }
  reverse_iterator rend() const {
    return const_cast<writable_array_view_type &>(array).
rend();
  }


  // const_reverse_iterator rbegin() const { return array.rbegin(); }


  // const_reverse_iterator rend() const { return array.rend(); }


  const_reverse_iterator crbegin() const { return array.rbegin(); }


  const_reverse_iterator crend() const { return array.rend(); }

private:

  // The following function are used from handler
  friend handler;
```

```

#ifdef TRISYCL_OPENCL
  /// Get the boost::compute::buffer or throw if unset
  auto get_cl_buffer() const {
    // This throws if not set
    return cl_buf.value();
  }


  /** Lazily associate a CL buffer to the SYCL buffer and copy data in
      if required

      \todo Move this into the buffer with queue/device-based caching
  */
  void copy_in_cl_buffer() {
    // This should be a constexpr
    cl_mem_flags flags = is_read_access() && is_write_access() ?
      CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR
      : is_read_access() ? CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR
                         : CL_MEM_WRITE_ONLY;

    /* Create the OpenCL buffer and copy in data from the host if in
       read mode */
    cl_buf = { task->get_queue()->get_boost_compute().get_context(),
               get_size()*sizeof(value_type),
               flags,
               is_read_access() ? array.data() : 0 };
  }


  /** Copy back the CL buffer to the SYCL if required

      \todo Move this into the buffer with queue/device-based caching
  */
  void copy_back_cl_buffer() {
    // \todo Use if constexpr in C++17
    if (is_write_access())
      task->get_queue()->get_boost_compute()
        .enqueue_read_buffer(get_cl_buffer(),
                             0 /*< Offset */,
                             get_size()*sizeof(value_type),
                             array.data());
  }
#endif

};

/// @} End the data Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ACCESSOR_DETAIL_ACCESSOR_HPP
```

11.9 include/CL/sycl/address_space/detail/address_space.hpp File Reference

Implement OpenCL address spaces in SYCL with C++-style.

This graph shows which files directly or indirectly include this file:



Classes

- struct cl::sycl::detail::opencl_type< T, AS >

    Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device. More...
- struct cl::sycl::detail::opencl_type< T, constant_address_space >

    Add an attribute for __constant address space. More...
- struct cl::sycl::detail::opencl_type< T, generic_address_space >

    Add an attribute for __generic address space. More...
- struct cl::sycl::detail::opencl_type< T, global_address_space >

    Add an attribute for __global address space. More...
- struct cl::sycl::detail::opencl_type< T, local_address_space >

    Add an attribute for __local address space. More...
- struct cl::sycl::detail::opencl_type< T, private_address_space >

    Add an attribute for __private address space. More...
- struct cl::sycl::detail::address_space_array< T, AS >

    Implementation of an array variable with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_fundamental< T, AS >

    Implementation of a fundamental type with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_object< T, AS >

    Implementation of an object type with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_ptr< T, AS >

    Implementation for an OpenCL address space pointer. More...
- struct cl::sycl::detail::address_space_base< T, AS >

    Implementation of the base infrastructure to wrap something in an OpenCL address space. More...
- struct cl::sycl::detail::address_space_variable< T, AS >

    Implementation of a variable with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_fundamental< T, AS >

    Implementation of a fundamental type with an OpenCL address space. More...

- struct cl::sycl::detail::address_space_ptr< T, AS >

  Implementation for an OpenCL address space pointer. More...
- struct cl::sycl::detail::address_space_array< T, AS >

  Implementation of an array variable with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_object< T, AS >

  Implementation of an object type with an OpenCL address space. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Typedefs

- template<typename T , address_space AS>
  using cl::sycl::detail::addr_space = typename std::conditional< std::is_pointer< T >::value, address_↩ space_ptr< T, AS >, typename std::conditional< std::is_class< T >::value, address_space_object< T, AS >, typename std::conditional< std::is_array< T >::value, address_space_array< T, AS >, address_space↩ _fundamental< T, AS > >::type >::type >::type

  Dispatch the address space implementation according to the requested type.

11.9.1 Detailed Description

Implement OpenCL address spaces in SYCL with C++-style.

Ronan at Keryell point FR

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file address_space.hpp.

11.10 address_space.hpp

```
#ifndef TRISYCL_SYCL_ADDRESS_SPACES_DETAIL_ADDRESS_SPACES_HPP
#define TRISYCL_SYCL_ADDRESS_SPACES_DETAIL_ADDRESS_SPACES_HPP

/** \file

    Implement OpenCL address spaces in SYCL with C++-style.

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup address_spaces
    @{
*/

/** Generate a type with some real OpenCL 2 attribute if we are on an
```

```
    OpenCL device

    In the general case, do not add any OpenCL address space qualifier */
template <typename T, address_space AS>
struct opencl_type {
  using type = T;
};

/// Add an attribute for __constant address space
template <typename T>
struct opencl_type<T, constant_address_space> {
  using type = T
#ifdef __SYCL_DEVICE_ONLY__
    /* Put the address space qualifier after the type so that we can
       construct pointer type with qualifier */
    __constant
#endif
    ;
};

/// Add an attribute for __generic address space
template <typename T>
struct opencl_type<T, generic_address_space> {
  using type = T
#ifdef __SYCL_DEVICE_ONLY__
    /* Put the address space qualifier after the type so that we can
       construct pointer type with qualifier */
    __generic
#endif
    ;
};

/// Add an attribute for __global address space
template <typename T>
struct opencl_type<T, global_address_space> {
  using type = T
#ifdef __SYCL_DEVICE_ONLY__
    /* Put the address space qualifier after the type so that we can
       construct pointer type with qualifier */
    __global
#endif
    ;
};

/// Add an attribute for __local address space
template <typename T>
struct opencl_type<T, local_address_space> {
  using type = T
#ifdef __SYCL_DEVICE_ONLY__
    /* Put the address space qualifier after the type so that we can
       construct pointer type with qualifier */
    __local
#endif
    ;
};

/// Add an attribute for __private address space
template <typename T>
struct opencl_type<T, private_address_space> {
  using type = T
#ifdef __SYCL_DEVICE_ONLY__
    /* Put the address space qualifier after the type so that we can
       construct pointer type with qualifier */
    __private
#endif
    ;
};


/* Forward declare some classes to allow some recursion in conversion
   operators */
template <typename SomeType, address_space SomeAS>
struct address_space_array;

template <typename SomeType, address_space SomeAS>
struct address_space_fundamental;

template <typename SomeType, address_space SomeAS>
struct address_space_object;

template <typename SomeType, address_space SomeAS>
struct address_space_ptr;

/** Dispatch the address space implementation according to the requested type

    \param T is the type of the object to be created

   \param AS is the address space to place the object into or to point to
   in the case of a pointer type
*/
template <typename T, address_space AS>
using addr_space =
  typename std::conditional<std::is_pointer<T>::value,
                            address_space_ptr<T, AS>,
  typename std::conditional<std::is_class<T>::value,
                            address_space_object<T, AS>,
  typename std::conditional<std::is_array<T>::value,
                            address_space_array<T, AS>,
                            address_space_fundamental<T, AS>
  >::type>::type>::type;


/** Implementation of the base infrastructure to wrap something in an
    OpenCL address space

    \param T is the type of the basic stuff to be created

    \param AS is the address space to place the object into

    \todo Verify/improve to deal with const/volatile?
*/
template <typename T, address_space AS>
struct address_space_base {
  /** Store the base type of the object

      \todo Add to the specification
  */
  using type = T;

  /** Store the base type of the object with OpenCL address space modifier

      \todo Add to the specification
  */
  using opencl_type = typename opencl_type<T, AS>::type;

  /** Set the address_space identifier that can be queried to know the
      pointer type */
  static auto constexpr address_space = AS;

};


/** Implementation of a variable with an OpenCL address space

    \param T is the type of the basic object to be created

    \param AS is the address space to place the object into
*/
template <typename T, address_space AS>
struct address_space_variable : public address_space_base<T, AS> {
  /** Store the base type of the object with OpenCL address space modifier

      \todo Add to the specification
  */
  using opencl_type = typename opencl_type<T, AS>::type;

  /// Keep track of the base class as a short-cut
  using super = address_space_base<T, AS>;

protected:

  /* C++11 helps a lot to be able to have the same constructors as the
     parent class here

     \todo Add this to the list of required C++11 features needed for SYCL
  */
  opencl_type variable;

public:

  /** Allow to create an address space version of an object or to convert
      one to be used by the classes inheriting by this one because it is
      not possible to directly initialize a base class member in C++ */
  address_space_variable(const T & v) : variable(v) { }


  /// Put back the default constructors canceled by the previous definition
  address_space_variable() = default;


  /** Conversion operator to allow a address_space_object<T> to be used
      as a T so that all the methods of a T and the built-in operators for
      T can be used on a address_space_object<T> too.

      Use opencl_type so that if we take the address of it, the address
      space is kept.
  */
  operator opencl_type & () { return variable; }

  /// Return the address of the value to implement pointers
  opencl_type * get_address() { return &variable; }

};


/** Implementation of a fundamental type with an OpenCL address space

    \param T is the type of the basic object to be created

    \param AS is the address space to place the object into

    \todo Verify/improve to deal with const/volatile?
*/
template <typename T, address_space AS>
struct address_space_fundamental : public
address_space_variable<T, AS> {
  /// Keep track of the base class as a short-cut
  using super = address_space_variable<T, AS>;

  /// Inherit from base class constructors
  using super::address_space_variable;


  /** Also request for the default constructors that have been disabled by
      the declaration of another constructor

      This ensures for example that we can write
      \code
        generic<float *> q;
      \endcode
      without initialization.
  */
  address_space_fundamental() = default;


  /** Allow for example assignment of a global<float> to a priv<double>
      for example

     Since it needs 2 implicit conversions, it does not work with the
     conversion operators already define, so add 1 more explicit
     conversion here so that the remaining implicit conversion can be
     found by the compiler.

     Strangely
     \code
     template <typename SomeType, address_space SomeAS>
     address_space_base(addr_space<SomeType, SomeAS>& v)
     : variable(SomeType(v)) { }
     \endcode
     cannot be used here because SomeType cannot be inferred. So use
     address_space_base<> instead

     Need to think further about it...
  */
  template <typename SomeType, cl::sycl::address_space SomeAS>
  address_space_fundamental(
address_space_fundamental<SomeType, SomeAS>& v)
  {
    /* Strangely I cannot have it working in the initializer instead, for
       some cases */
    super::variable = SomeType(v);
  }

};


/** Implementation for an OpenCL address space pointer

    \param T is the pointer type

    Note that if \a T is not a pointer type, it is an error.

    All the address space pointers inherit from it, which makes trivial
    the implementation of cl::sycl::multi_ptr<T, AS>
*/
template <typename T, address_space AS>
struct address_space_ptr : public address_space_fundamental<T, AS
> {
  // Verify that \a T is really a pointer
  static_assert(std::is_pointer<T>::value,
                "T must be a pointer type");

  /// Keep track of the base class as a short-cut
  using super = address_space_fundamental<T, AS>;

  /// Inherit from base class constructors
  using super::address_space_fundamental;

  /** Allow initialization of a pointer type from the address of an
      element with the same type and address space
  */
  address_space_ptr(address_space_fundamental<typename
std::pointer_traits<T>::element_type, AS> *p)
    : address_space_fundamental<T, AS> { p->get_address() } {}

  /// Put back the default constructors canceled by the previous definition
  address_space_ptr() = default;
};


/** Implementation of an array variable with an OpenCL address space

    \param T is the type of the basic object to be created

    \param AS is the address space to place the object into
*/
template <typename T, address_space AS>
struct address_space_array : public address_space_variable<T, AS>
 {
  /// Keep track of the base class as a short-cut
  using super = address_space_variable<T, AS>;

  /// Inherit from base class constructors
  using super::address_space_variable;


  /** Allow to create an address space array from an array
   */
  address_space_array(const T &array) {
    std::copy(std::begin(array), std::end(array), std::begin(super::variable));
  };


  /** Allow to create an address space array from an initializer list

      \todo Extend to more than 1 dimension
  */
  address_space_array(std::initializer_list<std::remove_extent_t<T>> list) {
    std::copy(std::begin(list), std::end(list), std::begin(super::variable));
  };

};


/** Implementation of an object type with an OpenCL address space

    \param T is the type of the basic object to be created

    \param AS is the address space to place the object into

    The class implementation is just inheriting of T so that all methods
    and non-member operators on T work also on address_space_object<T>

    \todo Verify/improve to deal with const/volatile?

    \todo what about T having some final methods?
*/
template <typename T, address_space AS>
struct address_space_object : public opencl_type<T, AS>::type,
                              public address_space_base<T, AS> {
  /** Store the base type of the object with OpenCL address space modifier

      \todo Add to the specification
  */
  using opencl_type = typename opencl_type<T, AS>::type;

  /* C++11 helps a lot to be able to have the same constructors as the
     parent class here but with an OpenCL address space

     \todo Add this to the list of required C++11 features needed for SYCL
  */
  using opencl_type::opencl_type;

  /** Allow to create an address space version of an object or to
      convert one */
  address_space_object(T && v) : opencl_type(v) { }

  /** Conversion operator to allow a address_space_object<T> to be used
```

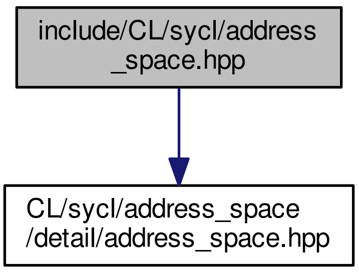
```
      as a T so that all the methods of a T and the built-in operators for
      T can be used on a address_space_object<T> too.

      Use opencl_type so that if we take the address of it, the address
      space is kept. */
  operator opencl_type & () { return *this; }

};

/// @} End the address_spaces Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ADDRESS_SPACES_DETAIL_ADDRESS_SPACES_HPP
```

11.11 include/CL/sycl/address_space.hpp File Reference

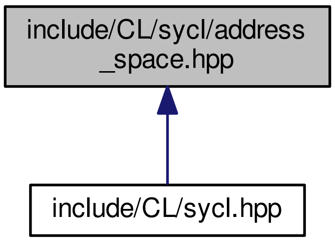
Implement OpenCL address spaces in SYCL with C++-style.

`#include "CL/sycl/address_space/detail/address_space.hpp"`

Include dependency graph for address_space.hpp:


include/CL/sycl/address_space.hpp
CL/sycl/address_space/detail/address_space.hpp

This graph shows which files directly or indirectly include this file:



Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Typedefs

- template< typename T >
  using cl::sycl::constant = detail::addr_space< T, constant_address_space >

  Declare a variable to be in the OpenCL constant address space.
- template< typename T >
  using cl::sycl::generic = detail::addr_space< T, generic_address_space >

  Declare a variable to be in the OpenCL 2 generic address space.
- template< typename T >
  using cl::sycl::global = detail::addr_space< T, global_address_space >

  Declare a variable to be in the OpenCL global address space.
- template< typename T >
  using cl::sycl::local = detail::addr_space< T, local_address_space >

  Declare a variable to be in the OpenCL local address space.
- template< typename T >
  using cl::sycl::priv = detail::addr_space< T, private_address_space >

  Declare a variable to be in the OpenCL private address space.
- template< typename Pointer , address_space AS>
  using cl::sycl::multi_ptr = detail::address_space_ptr< Pointer, AS >

  A pointer that can be statically associated to any address-space.

Enumerations

- enum cl::sycl::address_space {
  cl::sycl::constant_address_space, cl::sycl::generic_address_space, cl::sycl::global_address_space, cl↵
  ::sycl::local_address_space,
  cl::sycl::private_address_space }

  Enumerate the different OpenCL 2 address spaces.

Functions

- template<typename T , address_space AS>
  multi_ptr< T, AS > cl::sycl::make_multi (multi_ptr< T, AS > pointer)

  Construct a cl::sycl::multi_ptr<> with the right type.

11.11.1 Detailed Description

Implement OpenCL address spaces in SYCL with C++-style.

Note that in SYCL 1.2, only pointer types should be specified but in this implementation we generalize the concept to any type.

Todo Add the alias ..._ptr<T> = ...<T *>

Ronan at Keryell point FR

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file address_space.hpp.

11.12 address_space.hpp

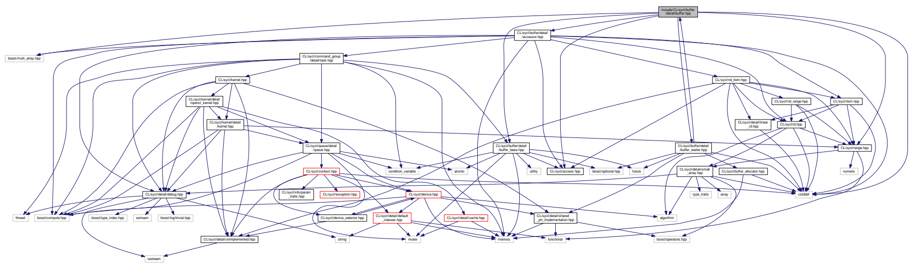
```
#ifndef TRISYCL_SYCL_ADDRESS_SPACE_HPP
#define TRISYCL_SYCL_ADDRESS_SPACE_HPP

/** \file

    Implement OpenCL address spaces in SYCL with C++-style.

    Note that in SYCL 1.2, only pointer types should be specified but
    in this implementation we generalize the concept to any type.

    \todo Add the alias ..._ptr<T> = ...<T *>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

namespace cl {
namespace sycl {

/** \addtogroup address_spaces Dealing with OpenCL address spaces
    @{
*/

/** Enumerate the different OpenCL 2 address spaces */
enum address_space {
  constant_address_space,
  generic_address_space,
  global_address_space,
  local_address_space,
  private_address_space,
};

}
}
/// @} End the address_spaces Doxygen group


#include "CL/sycl/address_space/detail/address_space.hpp"


namespace cl {
namespace sycl {
```

```

/** \addtogroup address_spaces
    @{
*/

/** Declare a variable to be in the OpenCL constant address space

    \param T is the type of the object
*/
template <typename T>
using constant = detail::addr_space<T, constant_address_space>
;


/** Declare a variable to be in the OpenCL 2 generic address space

    \param T is the type of the object
*/
template <typename T>
using generic = detail::addr_space<T, generic_address_space>;


/** Declare a variable to be in the OpenCL global address space

    \param T is the type of the object
*/
template <typename T>
using global = detail::addr_space<T, global_address_space>
;


/** Declare a variable to be in the OpenCL local address space

    \param T is the type of the object
*/
template <typename T>
using local = detail::addr_space<T, local_address_space>;


/** Declare a variable to be in the OpenCL private address space

    \param T is the type of the object
*/
template <typename T>
using priv = detail::addr_space<T, private_address_space>;


/** A pointer that can be statically associated to any address-space

    \param Pointer is the pointer type

    \param AS is the address space to point to

    Note that if \a Pointer is not a pointer type, it is an error.
*/
template <typename Pointer, address_space AS>
using multi_ptr = detail::address_space_ptr<Pointer, AS>;


/** Construct a cl::sycl::multi_ptr<> with the right type

    \param pointer is the address with its address space to point to

    \todo Implement the case with a plain pointer
*/
template <typename T, address_space AS>
multi_ptr<T, AS> make_multi(multi_ptr<T, AS> pointer) {
  return pointer;
}

}
}
/// @} End the parallelism Doxygen group

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ADDRESS_SPACE_HPP
```

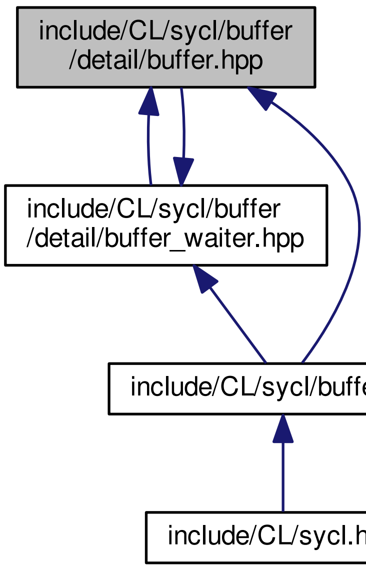
11.13 include/CL/sycl/buffer/detail/buffer.hpp File Reference

```
#include <cstddef>
#include <boost/multi_array.hpp>
#include "CL/sycl/access.hpp"
#include "CL/sycl/buffer/detail/accessor.hpp"
#include "CL/sycl/buffer/detail/buffer_base.hpp"
#include "CL/sycl/buffer/detail/buffer_waiter.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for buffer.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/buffer /detail/buffer.hpp
include/CL/sycl/buffer /detail/buffer_waiter.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::detail::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

    The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<typename BufferDetail >
  static std::shared_ptr< detail::task > cl::sycl::detail::buffer_add_to_task (BufferDetail buf, handler ∗command_group_handler, bool is_write_mode)

    Proxy function to avoid some circular type recursion.

11.14 buffer.hpp

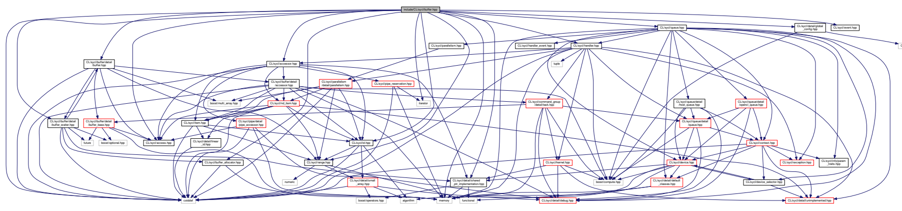
```
#ifndef TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_HPP
#define TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_HPP

/** \file The OpenCL SYCL buffer<> detail implementation

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>

#include <boost/multi_array.hpp>

#include "CL/sycl/access.hpp"
#include "CL/sycl/buffer/detail/accessor.hpp"
#include "CL/sycl/buffer/detail/buffer_base.hpp"
#include "CL/sycl/buffer/detail/buffer_waiter.hpp"
#include "CL/sycl/range.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** 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.
*/
template <typename T,
          std::size_t Dimensions = 1>
class buffer : public detail::buffer_base,
               public detail::debug<buffer<T, Dimensions>> {
public:

  // Extension to SYCL: provide pieces of STL container interface
  using element = T;
  using value_type = T;

private:

  /** If some allocation is requested, it is managed by this multi_array
      to ease initialization from data */
  boost::multi_array<T, Dimensions> allocation;

  // \todo Replace U and D somehow by T and Dimensions
  // To allow allocation access
  template <typename U,
            std::size_t D,
            access::mode Mode,
            access::target Target /* = access::global_buffer */>
    friend class detail::accessor;


  /** This is the multi-dimensional interface to the data that may point
      to either allocation in the case of storage managed by SYCL itself
      or to some other memory location in the case of host memory or
      storage<> abstraction use
  */
  boost::multi_array_ref<T, Dimensions> access;

  /// The weak pointer to copy back data on buffer deletion
  weak_ptr_class<T> final_data;

  /** The shared pointer in the case the buffer memory is shared with
      the host */
  shared_ptr_class<T> shared_data;

  // Track if the buffer memory is provided as host memory
  bool host_write_back = false;

public:

  /// Create a new read-write buffer of size \param r
  buffer(const range<Dimensions> &r) : buffer_base { false },
                                       allocation { r },
                                       access { allocation }
                                       {}


  /** Create a new read-write buffer from \param host_data of size
      \param r without further allocation */
  buffer(T *host_data, const range<Dimensions> &r) :
buffer_base { false },
                                                     access { host_data, r },
                                                     host_write_back { true }
                                                     {}


  /** Create a new read-only buffer from \param host_data of size \param r
      without further allocation

      \todo Clarify the semantics in the spec. What happens if the
      host change the host_data after buffer creation?
  */
  buffer(const T *host_data, const range<Dimensions> &r) :
    /* \todo Need to solve this const buffer issue in a clean way

       Just allocate memory? */
    buffer_base { true },
    access { const_cast<T *>(host_data), r }
    {}


  /** Create a new buffer with associated memory, using the data in
      host_data

      The ownership of the host_data is shared between the runtime and the
      user. In order to enable both the user application and the SYCL
      runtime to use the same pointer, a cl::sycl::mutex_class is
      used.
  */
  buffer(shared_ptr_class<T> &host_data,
         const range<Dimensions> &r)
    : buffer_base { false },
    access { host_data.get(), r },
    shared_data { host_data }
    {}


  /// Create a new allocated 1D buffer from the given elements
  template <typename Iterator>
  buffer(Iterator start_iterator, Iterator end_iterator) :
    buffer_base { false },
    // The size of a multi_array is set at creation time
    allocation { boost::extents[std::distance(start_iterator, end_iterator)] },
    access { allocation }
    {
      /* Then assign allocation since this is the only multi_array
         method with this iterator interface */
      allocation.assign(start_iterator, end_iterator);
    }


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

      \todo To implement and deal with reference counting
  buffer(buffer<T, Dimensions> b,
         index<Dimensions> base_index,
         range<Dimensions> sub_range)
  */

  /// \todo Allow CLHPP objects too?
  ///
  /*
  buffer(cl_mem mem_object,
         queue from_queue,
         event available_event)
  */

  /** The buffer content may be copied back on destruction to some
      final location */
  ~buffer() {
    /* If there is a final_data set and that points to something
       alive, copy back the data through the shared pointer */
    if (auto p = final_data.lock())
      std::copy_n(access.data(), access.num_elements(), p.get());
    /* If data are shared with the host but not concretely, we would
       have to copy back the data to the host */
    // else if (shared_data)
    //   std::copy_n(access.data(), access.num_elements(), shared_data.get());
  }

  // Use BOOST_DISABLE_ASSERTS at some time to disable range checking

  /// Return an accessor of the required mode \param M
  /// \todo Remove if not used
  template <access::mode Mode,
            access::target Target = access::target::global_buffer
>
  detail::accessor<T, Dimensions, Mode, Target>
get_access() {
    return { *this };
  }


 /** Return a range object representing the size of the buffer in
      terms of number of elements in each dimension as passed to the
      constructor
  */
  auto get_range() const {
    /* Interpret the shape which is a pointer to the first element as an
       array of Dimensions elements so that the range<Dimensions>
       constructor is happy with this collection

       \todo Add also a constructor in range<> to accept a const
       std::size_t *?
    */
    return range<Dimensions> {
      *(const std::size_t (*)[Dimensions])(allocation.shape())
        };
  }


  /** Returns the total number of elements in the buffer

      Equal to get_range()[0] * ... * get_range()[dimensions-1].
  */
  auto get_count() const {
    return allocation.num_elements();
  }


  /** Returns the size of the buffer storage in bytes

      Equal to get_count()*sizeof(T).

      \todo rename to something else. In
      http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf
      it is named bytes() for example
  */
  size_t get_size() const {
    return get_count()*sizeof(T);
  }


  /** Set the weak pointer to copy back data on buffer deletion

      \todo Add a write kernel dependency on the buffer so the buffer
      destructor has to wait for the kernel execution if the buffer is
      also accessed through a write accessor
  */
  void set_final_data(weak_ptr_class<T> && finalData) {
    final_data = finalData;
  }

private:

  /** Get a \c future to wait from inside the \c cl::sycl::buffer in
      case there is something to copy back to the host

      \return A \c future in the \c optional if there is something to
      wait for, otherwise an empty \c optional
  */
  boost::optional<std::future<void>> get_destructor_future() {
    boost::optional<std::future<void>> f;
    /* If there is only 1 shared_ptr user of the buffer, this is the
       caller of this function, the \c buffer_waiter, so there is no
       need to get a \ future otherwise there will be a dead-lock if
       there is only 1 thread waiting for itself.

       Since \c use_count() is applied to a \c shared_ptr just created
       for this purpose, it actually increase locally the count by 1,
       so check for 1 + 1 use count instead...
    */
    if (shared_from_this().use_count() > 2)
      // \todo Double check the specification and add unit tests
      if (host_write_back || !final_data.expired() || shared_data) {
        // Create a promise to wait for
        notify_buffer_destructor = std::promise<void> {};
        // And return the future to wait for it
        f = notify_buffer_destructor->get_future();
      }
    return f;
  }


  // Allow buffer_waiter destructor to access get_destructor_future()
  // friend detail::buffer_waiter<T, Dimensions>::~buffer_waiter();
  /* \todo Work around to Clang bug
     https://llvm.org/bugs/show_bug.cgi?id=28873 cannot use destructor
     here */
  friend detail::buffer_waiter<T, Dimensions>;

};


/** Proxy function to avoid some circular type recursion

    \return a shared_ptr<task>

    \todo To remove with some refactoring
*/
template <typename BufferDetail>
static std::shared_ptr<detail::task>
buffer_add_to_task(BufferDetail buf,
                   handler *command_group_handler,
                   bool is_write_mode) {
    return buf->add_to_task(command_group_handler, is_write_mode);
  }

/// @} End the data Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_HPP
```

11.15 include/CL/sycl/buffer.hpp File Reference

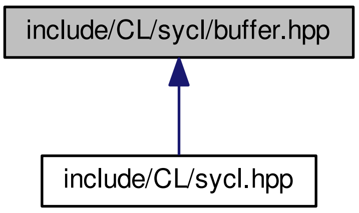
`#include <cstddef>`

```
#include <iterator>
#include <memory>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/buffer/detail/buffer.hpp"
#include "CL/sycl/buffer/detail/buffer_waiter.hpp"
#include "CL/sycl/buffer_allocator.hpp"
#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/event.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/queue.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for buffer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class cl::sycl::buffer< T, Dimensions, Allocator >

  < T, Dimensions, Mode, Target> up data Data access and storage in SYCL
- struct std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- std

11.16 buffer.hpp

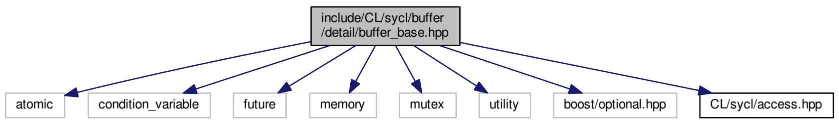
```
#ifndef TRISYCL_SYCL_BUFFER_HPP
#define TRISYCL_SYCL_BUFFER_HPP

/** \file The OpenCL SYCL buffer<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <iterator>
#include <memory>

#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/buffer/detail/buffer.hpp"
#include "CL/sycl/buffer/detail/buffer_waiter.hpp"
#include "CL/sycl/buffer_allocator.hpp"
#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/event.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/queue.hpp"
#include "CL/sycl/range.hpp"

namespace cl {
namespace sycl {

/** \addtogro<T, Dimensions, Mode, Target>up data Data access and storage in SYCL
    @{
*/

/** 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.

    \todo We have some read-write buffers and some read-only buffers,
    according to the constructor called. So we could have some static
    checking for correctness with the accessors used, but we do not have a
    way in the specification to have a read-only buffer type for this.

    \todo There is a naming inconsistency in the specification between
    buffer and accessor on T versus datatype

    \todo Finish allocator implementation

    \todo Think about the need of an allocator when constructing a buffer
    from other buffers

    \todo Add constructors from arrays so that in C++17 the range and
    type can be infered from the constructor

    \todo Add constructors from array_ref
*/
template <typename T,
          std::size_t Dimensions = 1,
          typename Allocator = buffer_allocator<T>>
class buffer
  /* Use the underlying buffer waiter implementation that can be
     shared in the SYCL model */
  : public detail::shared_ptr_implementation<
                             buffer<T, Dimensions, Allocator>,
                             detail::buffer_waiter<T, Dimensions, Allocator>>,
    detail::debug<buffer<T, Dimensions, Allocator>> {
public:

  /// The STL-like types
  using value_type = T;
  using reference = value_type&;
  using const_reference = const value_type&;
  using allocator_type = Allocator;

private:

  // The type encapsulating the implementation
  using implementation_t =
    detail::shared_ptr_implementation<
                             buffer<T, Dimensions, Allocator>,
                             detail::buffer_waiter<T, Dimensions, Allocator>
>;

public:

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

  /** Use default constructors so that we can create a new buffer copy
      from another one, with either a l-value or an r-value (for
      std::move() for example).

      Since we just copy the shared_ptr<> from the
      shared_ptr_implementation above, this is where/how the sharing
      magic is happening with reference counting in this case.
  */
  buffer() = default;


  /** Create a new buffer of the given size with
      storage managed by the SYCL runtime

      The default behavior is to use the default host buffer
      allocator, in order to allow for host accesses. If the type of
      the buffer, has the const qualifier, then the default allocator
      will remove the qualifier to allow host access to the data.

      \param[in] r defines the size

      \param[in] allocator is to be used by the SYCL runtime
  */
  buffer(const range<Dimensions> &r, Allocator allocator = {})
    : implementation_t { detail::waiter(new
detail::buffer<T, Dimensions>
                                         { r }) }
      {}


  /** Create a new buffer with associated host memory

      \param[in] host_data points to the storage and values used by
      the buffer

      \param[in] r defines the size

      \param[in] allocator is to be used by the SYCL runtime, of type
      cl::sycl::buffer_allocator<T> by default

      The host address is const T, so the host accesses can be
      read-only.

      However, the typename T is not const so the device accesses can
      be both read and write accesses. Since, the host_data is const,
      this buffer is only initialized with this memory and there is
      no write after its destruction, unless there is another final
      data address given after construction of the buffer.
  */
  buffer(const T *host_data,
         const range<Dimensions> &r,
         Allocator allocator = {})
    : implementation_t { detail::waiter(new
detail::buffer<T, Dimensions>
            { host_data, r }) }
  {}


  /** Create a new buffer with associated host memory

      \param[inout] host_data points to the storage and values used by
      the buffer

      \param[in] r defines the size

      \param[in] allocator is to be used by the SYCL runtime, of type
      cl::sycl::buffer_allocator<T> by default

      The memory is owned by the runtime during the lifetime of the
      object.  Data is copied back to the host unless the user
      overrides the behavior using the set_final_data method. host_data
      points to the storage and values used by the buffer and
      range<dimensions> defines the size.
  */
  buffer(T *host_data, const range<Dimensions> &r, Allocator allocator = {})
    : implementation_t { detail::waiter(new
detail::buffer<T, Dimensions>
            { host_data, r }) }
  {}


  /** Create a new buffer with associated memory, using the data in
      host_data

      \param[inout] host_data points to the storage and values used by
      the buffer

      \param[in] r defines the size

      \param[in] allocator is to be used by the SYCL runtime, of type
      cl::sycl::buffer_allocator<T> by default

      The ownership of the host_data is shared between the runtime and the
      user. In order to enable both the user application and the SYCL
      runtime to use the same pointer, a cl::sycl::mutex_class is
      used. The mutex m is locked by the runtime whenever the data is in
      use and unlocked otherwise. Data is synchronized with host_data, when
      the mutex is unlocked by the runtime.

      \todo update the specification to replace the pointer by a
      reference and provide the constructor with and without a mutex
  */
  buffer(shared_ptr_class<T> &host_data,
         const range<Dimensions> &buffer_range,
         cl::sycl::mutex_class &m,
         Allocator allocator = {}) {
    detail::unimplemented();
  }


  /** Create a new buffer with associated memory, using the data in
      host_data

      \param[inout] host_data points to the storage and values used by
      the buffer

      \param[in] r defines the size

      \param[inout] m is the mutex used to protect the data access

      \param[in] allocator is to be used by the SYCL runtime, of type
      cl::sycl::buffer_allocator<T> by default

      The ownership of the host_data is shared between the runtime and the
      user. In order to enable both the user application and the SYCL
      runtime to use the same pointer, a cl::sycl::mutex_class is
      used.

      \todo add this mutex-less constructor to the specification
  */
  buffer(shared_ptr_class<T> host_data,
         const range<Dimensions> &buffer_range,
         Allocator allocator = {})
    : implementation_t { detail::waiter(new
detail::buffer<T, Dimensions>
            { host_data, buffer_range }) }
  {}


  /** Create a new buffer which is initialized by host_data

      \param[inout] host_data points to the storage and values used to
      initialize the buffer

      \param[in] r defines the size

      \param[in] allocator is to be used by the SYCL runtime, of type
      cl::sycl::buffer_allocator<T> by default

      The SYCL runtime receives full ownership of the host_data unique_ptr
      and there in effect there is no synchronization with the application
      code using host_data.

      \todo Update the API to add template <typename D =
      std::default_delete<T>> because the
      unique_ptr_class/std::unique_ptr have the destructor type as
      dependent
  */
  template <typename D = std::default_delete<T>>
  buffer(unique_ptr_class<T, D> &&host_data,
         const range<Dimensions> &buffer_range,
         Allocator allocator = {})
  // Just delegate to the constructor with normal pointer
    : buffer(host_data.get(), buffer_range, allocator) {
    // Then release the host_data memory
    host_data.release();
  }


  /** Create a new allocated 1D buffer initialized from the given
```

```
      elements ranging from first up to one before last

      The data is copied to an intermediate memory position by the
      runtime. Data is written back to the same iterator set if the
      iterator is not a const iterator.

      \param[inout] start_iterator points to the first element to copy

      \param[in] end_iterator points to just after the last element to copy

      \param[in] allocator is to be used by the SYCL runtime, of type
      cl::sycl::buffer_allocator<T> by default

      \todo Implement the copy back at buffer destruction

      \todo Generalize this for n-D and provide column-major and row-major
      initialization

      \todo a reason to have this nD is that
            set_final_data(weak_ptr_class<T> & finalData) is actually
            doing this linearization anyway

      \todo Allow read-only buffer construction too

      \todo update the specification to deal with forward iterators
      instead and rewrite back only when it is non const and output
      iterator at least

      \todo Allow initialization from ranges and collections à la STL
  */
  template <typename InputIterator,
            /* To force some iterator concept checking to avoid GCC 4.9
               diving into this when initializing from ({ int, int })
               which is a range<> and and not an iterator... */
            typename ValueType =
            typename std::iterator_traits<InputIterator>::value_type>
  buffer(InputIterator start_iterator,
         InputIterator end_iterator,
         Allocator allocator = {}) :
    implementation_t { detail::waiter(new
detail::buffer<T, Dimensions>
            { start_iterator, end_iterator }) }
  {}


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

      \param[inout] b is the buffer with the real data

      \param[in] base_index specifies the origin of the sub-buffer inside the
      buffer b

      \param[in] sub_range specifies the size of the sub-buffer

      \todo To be implemented

      \todo Update the specification to replace index by id
  */
  buffer(buffer<T, Dimensions, Allocator> &b,
         const id<Dimensions> &base_index,
         const range<Dimensions> &sub_range,
         Allocator allocator = {}) { detail::unimplemented(); }


#ifdef TRISYCL_OPENCL
  /** Create a buffer from an existing OpenCL memory object associated
      with a context after waiting for an event signaling the
      availability of the OpenCL data

      \param[inout] mem_object is the OpenCL memory object to use

      \param[inout] from_queue is the queue associated to the memory
      object

      \param[in] available_event specifies the event to wait for if
      non null

      Note that a buffer created from a cl_mem object will only have
      one underlying cl_mem for the lifetime of the buffer and use on
      an incompatible queue constitues an error.

      \todo To be implemented

      \todo Improve the specification to allow CLHPP objects too
  */
  buffer(cl_mem mem_object,
        queue from_queue,
        event available_event = {},
        Allocator allocator = {}) { detail::unimplemented();  }
#endif


  // Use BOOST_DISABLE_ASSERTS at some time to disable range checking

  /** Get an accessor to the buffer with the required mode

      \param Mode is the requested access mode

      \param Target is the type of object to be accessed

      \param[in] command_group_handler is the command group handler in
      which the kernel is to be executed

      \todo Do we need for an accessor to increase the reference count of
      a buffer object? It does make more sense for a host-side accessor.

      \todo Implement the modes and targets
  */
  template <access::mode Mode,
            access::target Target = access::target::global_buffer
>
  accessor<T, Dimensions, Mode, Target>
  get_access(handler &command_group_handler) {
    static_assert(Target == access::target::global_buffer
                  || Target == access::target::constant_buffer,
                  "get_access(handler) can only deal with access::global_buffer"
                  " or access::constant_buffer (for host_buffer accessor"
                  " do not use a command group handler");
    return { *this, command_group_handler };
  }


  /** Get a host accessor to the buffer with the required mode

      \param Mode is the requested access mode

      \todo Implement the modes

      \todo More elegant solution
  */
  template <access::mode Mode,
            access::target Target = access::target::host_buffer>
  accessor<T, Dimensions, Mode, Target>
  get_access() {
    static_assert(Target == access::target::host_buffer,
                  "get_access() without a command group handler is only"
                  " for host_buffer accessor");
    return { *this };
  }


  /** Return a range object representing the size of the buffer in
      terms of number of elements in each dimension as passed to the
      constructor

      \todo rename to the equivalent from array_ref proposals? Such
      as size() in
      http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0009r2.html
  */
  auto get_range() const {
    /* Interpret the shape which is a pointer to the first element as an
       array of Dimensions elements so that the range<Dimensions>
       constructor is happy with this collection
    */
    return implementation->implementation->get_range();
  }


  /** Returns the total number of elements in the buffer

      Equal to get_range()[0] * ... * get_range()[dimensions-1].
  */
  auto get_count() const {
    return implementation->implementation->get_count();
  }


  /** Returns the size of the buffer storage in bytes

      Equal to get_count()*sizeof(T).

      \todo rename to something else. In
      http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf
      it is named bytes() for example
  */
  size_t get_size() const {
    return implementation->implementation->get_size();
  }


  /** Returns the number of buffers that are shared/referenced

      For example
      \code
      cl::sycl::buffer<int> b { 1000 };
      // Here b.use_count() should return 1
      cl::sycl::buffer<int> c { b };
      // Here b.use_count() and b.use_count() should return 2
      \endcode

      \todo Add to the specification, useful for validation
  */
  auto use_count() const {
    // Rely on the shared_ptr<> use_count()
    return implementation.use_count();
  }


  /** Ask for read-only status of the buffer

      \todo Add to specification
  */
  bool is_read_only() const {
    return implementation->implementation->read_only;
  }


  /** Set destination of buffer data on destruction

      The finalData points to the host memory to which, the outcome of all
      the buffer processing is going to be copied to.

      This is the final pointer, which is going to be accessible after the
      destruction of the buffer and in the case where this is a valid
      pointer, the data are going to be copied to this host address.

      finalData is different from the original host address, if the buffer
      was created associated with one. This is mainly to be used when a
      shared_ptr is given in the constructor and the output data will
      reside in a different location from the initialization data.

      It is defined as a weak_ptr referring to a shared_ptr that is not
      associated with the cl::sycl::buffer, and so the cl::sycl::buffer
      will have no ownership of finalData.

      \todo Update the API to take finalData by value instead of by
            reference.  This way we can have an implicit conversion
            possible at the API call from a shared_ptr<>, avoiding an
            explicit weak_ptr<> creation

      \todo figure out how set_final_data() interact with the other
      way to write back some data or with some data sharing with the
      host that can not be undone
  */
  void set_final_data(weak_ptr_class<T> finalData) {
    implementation->implementation->set_final_data(std::move(finalData));
  }

};

/// @} End the data Doxygen group

}
}

/* Inject a custom specialization of std::hash to have the buffer
   usable into an unordered associative container

   \todo Add this to the spec
*/
namespace std {

template <typename T,
          std::size_t Dimensions,
          typename Allocator>
struct hash<cl::sycl::buffer<T, Dimensions, Allocator>> {

  auto operator()(const cl::sycl::buffer<T, Dimensions, Allocator>
 &b) const {
    // Forward the hashing to the implementation
    return b.hash();
  }

};

}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_BUFFER_HPP
```

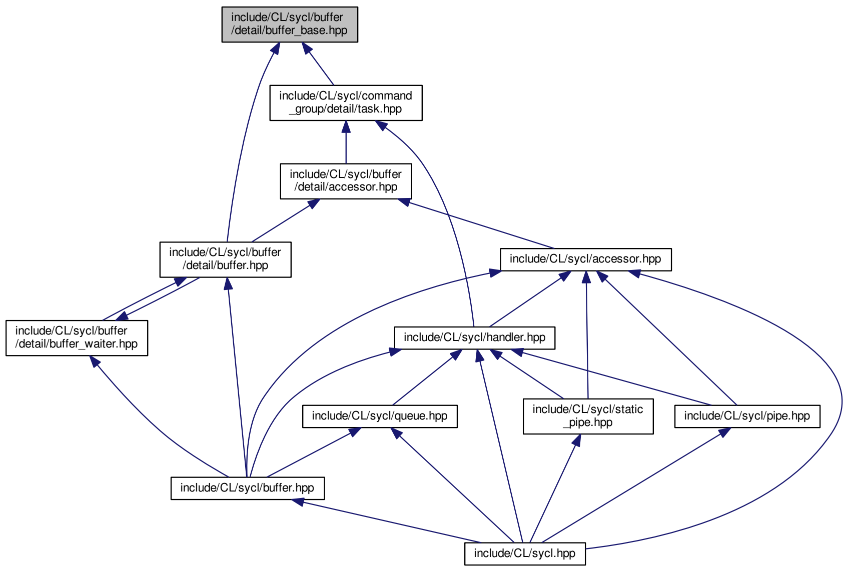
11.17 include/CL/sycl/buffer/detail/buffer_base.hpp File Reference

```
#include <atomic>
#include <condition_variable>
#include <future>
#include <memory>
#include <mutex>
#include <utility>
#include <boost/optional.hpp>
#include "CL/sycl/access.hpp"
```

Include dependency graph for buffer_base.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct cl::sycl::detail::buffer_base

  Factorize some template independent buffer aspects in a base class.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- static std::shared_ptr< detail::task > cl::sycl::detail::add_buffer_to_task (handler ∗command_group_handler, std::shared_ptr< detail::buffer_base > b, bool is_write_mode)

11.18 buffer_base.hpp

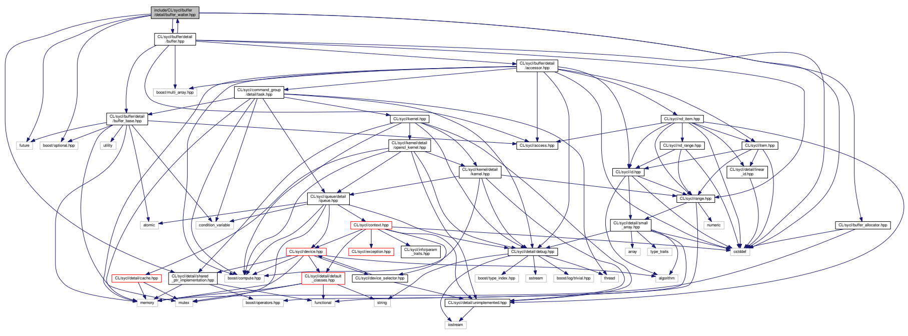
```
#ifndef TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_BASE_HPP
#define TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_BASE_HPP

/** \file The buffer_base behind the buffers, independent of the data
    type

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <atomic>
#include <condition_variable>
#include <future>
#include <memory>
#include <mutex>
#include <utility>

// \todo Use C++17 optional when it is mainstream
#include <boost/optional.hpp>

#include "CL/sycl/access.hpp"

namespace cl {
namespace sycl {

class handler;

namespace detail {

struct task;
struct buffer_base;
inline static std::shared_ptr<detail::task>
add_buffer_to_task(handler *command_group_handler,
                   std::shared_ptr<detail::buffer_base> b,
                   bool is_write_mode);

/** Factorize some template independent buffer aspects in a base class
 */
struct buffer_base : public std::enable_shared_from_this<buffer_base> {
  /// If the data are read-only, store the information for later optimization.
  /// \todo Replace this by a static read-only type for the buffer
  bool read_only;

  //// Keep track of the number of kernel accessors using this buffer
  std::atomic<size_t> number_of_users;

  /// Track the latest task to produce this buffer
  std::weak_ptr<detail::task> latest_producer;
  /// To protect the access to latest_producer
  std::mutex latest_producer_mutex;

  /// To signal when this buffer ready
  std::condition_variable ready;
  /// To protect the access to the condition variable
  std::mutex ready_mutex;

  /** If the SYCL user buffer destructor is blocking, use this to
      block until this buffer implementation is destroyed.

      Use a void promise since there is no value to send, only
      waiting */
  boost::optional<std::promise<void>> notify_buffer_destructor;


  /// Create a buffer base
  buffer_base(bool read_only) : read_only { read_only },
                                number_of_users { 0 } {}


  /// The destructor wait for not being used anymore
  ~buffer_base() {
    wait();
    // If there is the last SYCL user buffer waiting, notify it
    if (notify_buffer_destructor)
      notify_buffer_destructor->set_value();
  }


  /// Wait for this buffer to be ready, which is no longer in use
  void wait() {
    std::unique_lock<std::mutex> ul { ready_mutex };
    ready.wait(ul, [&] {
        // When there is no producer for this buffer, we are ready to use it
        return number_of_users == 0;
      });
  }


  /// Mark this buffer in use by a task
  void use() {
    // Increment the use count
    ++number_of_users;
  }


  /// A task has released the buffer
  void release() {
    if (--number_of_users == 0)
      // Notify the host consumers or the buffer destructor that it is ready
      ready.notify_all();
  }


  /// Return the latest producer for the buffer
  std::shared_ptr<detail::task> get_latest_producer() {
    std::lock_guard<std::mutex> lg { latest_producer_mutex };
    // Return the valid shared_ptr to the task, if any
    return latest_producer.lock();
  }


  /** Return the latest producer for the buffer and set another
      future producer
  */
  std::shared_ptr<detail::task>
  set_latest_producer(std::weak_ptr<detail::task> newer_latest_producer) {
    std::lock_guard<std::mutex> lg { latest_producer_mutex };
    using std::swap;

    swap(newer_latest_producer, latest_producer);
    // Return the valid shared_ptr to the previous producing task, if any
    return newer_latest_producer.lock();
  }


  /// Add a buffer to the task running the command group
  std::shared_ptr<detail::task>
  add_to_task(handler *command_group_handler, bool is_write_mode) {
    return add_buffer_to_task(command_group_handler,
                              shared_from_this(),
                              is_write_mode);
  }

};

```

```
}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_BASE_HPP
```

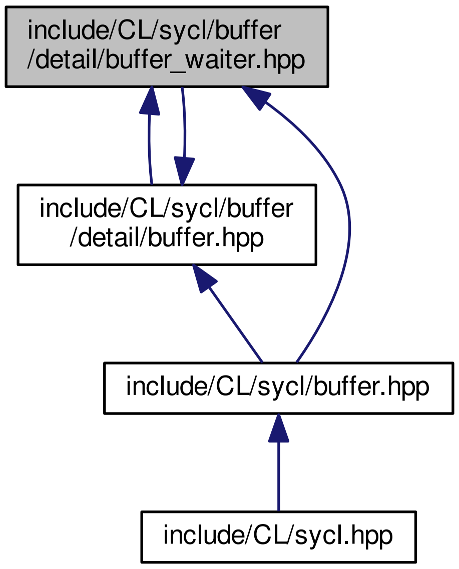
11.19 include/CL/sycl/buffer/detail/buffer_waiter.hpp File Reference

```
#include <cstddef>
#include <future>
#include <boost/optional.hpp>
#include "CL/sycl/buffer/detail/buffer.hpp"
#include "CL/sycl/buffer_allocator.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
```

Include dependency graph for buffer_waiter.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >

  A helper class to wait for the final buffer destruction if the conditions for blocking are met. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<typename T , std::size_t Dimensions = 1>
  auto cl::sycl::detail::waiter (detail::buffer< T, Dimensions > ∗b)

  Helper function to create a new buffer_waiter.

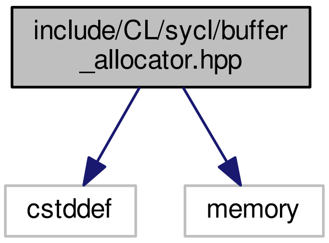
11.20 buffer_waiter.hpp

```
#ifndef TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_WAITER_HPP
#define TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_WAITER_HPP

/** \file A helper class to wait for the buffer<> detail

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <future>

// \todo Use C++17 optional when it is mainstream
#include <boost/optional.hpp>

#include "CL/sycl/buffer/detail/buffer.hpp"
#include "CL/sycl/buffer_allocator.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** A helper class to wait for the final buffer destruction if the
    conditions for blocking are met
*/
template <typename T,
          std::size_t Dimensions = 1,
          typename Allocator = buffer_allocator<T>>
class buffer_waiter :
    public detail::shared_ptr_implementation<buffer_waiter<T,
                                                           Dimensions,
                                                           Allocator>,
                                             detail::buffer<T, Dimensions>>,
    detail::debug<buffer_waiter<T, Dimensions, Allocator>> {

  // The type encapsulating the implementation
  using implementation_t =
    detail::shared_ptr_implementation<buffer_waiter<T, Dimensions, Allocator>
,
                                      detail::buffer<T, Dimensions>>;

public:

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

  /// Create a new buffer_waiter on top of a detail::buffer
  buffer_waiter(detail::buffer<T, Dimensions> *b) :
implementation_t { b } {}


  /** The buffer_waiter destructor waits for any data to be written
      back to the host, if any
  */
  ~buffer_waiter() {
    /* Get a future from the implementation if we have to wait for its
       destruction */
    auto f = implementation->get_destructor_future();
    if (f) {
      /* No longer carry for the implementation buffer which is free to
         live its life up to its destruction */
      implementation.reset();
      TRISYCL_DUMP_T("~buffer_waiter() is waiting");
      // Then wait for its end in some other thread
      f->wait();
      TRISYCL_DUMP_T("~buffer_waiter() is done");
    }
  }
};


/// Helper function to create a new buffer_waiter
template <typename T,
          std::size_t Dimensions = 1>
inline auto waiter(detail::buffer<T, Dimensions> *b) {
  return new buffer_waiter<T, Dimensions> { b };
}

/// @} End the data Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_WAITER_HPP
```

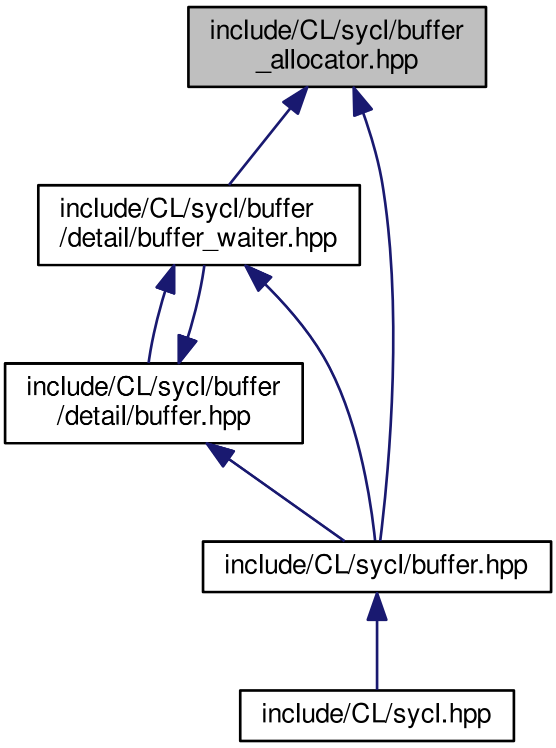
11.21 include/CL/sycl/buffer_allocator.hpp File Reference

```
#include <cstddef>
#include <memory>
```

Include dependency graph for buffer_allocator.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/buffer _allocator.hpp
include/CL/sycl/buffer /detail/buffer_waiter.hpp
include/CL/sycl/buffer /detail/buffer.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Typedefs

- template<typename T >

  using cl::sycl::buffer_allocator = std::allocator< T >

  The default buffer allocator used by the runtime, when no allocator is defined by the user.

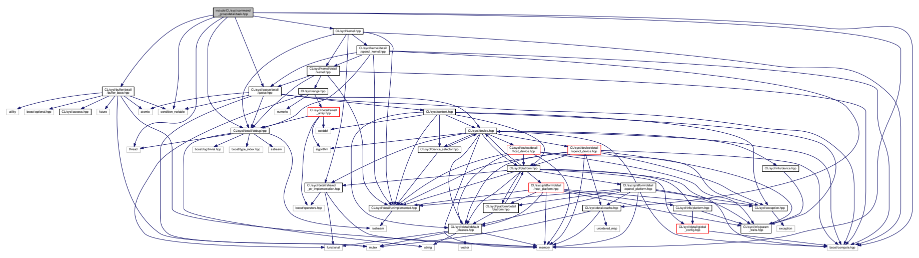
11.22 buffer_allocator.hpp

```
#ifndef TRISYCL_SYCL_BUFFER_ALLOCATOR_HPP
#define TRISYCL_SYCL_BUFFER_ALLOCATOR_HPP

/** \file The OpenCL SYCL buffer_allocator

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <memory>

namespace cl {
namespace sycl {

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** The default buffer allocator used by the runtime, when no allocator is
    defined by the user

    Reuse the C++ default allocator.
*/
template <typename T>
using buffer_allocator = std::allocator<T>;

/// @} End the data Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_BUFFER_ALLOCATOR_HPP
```

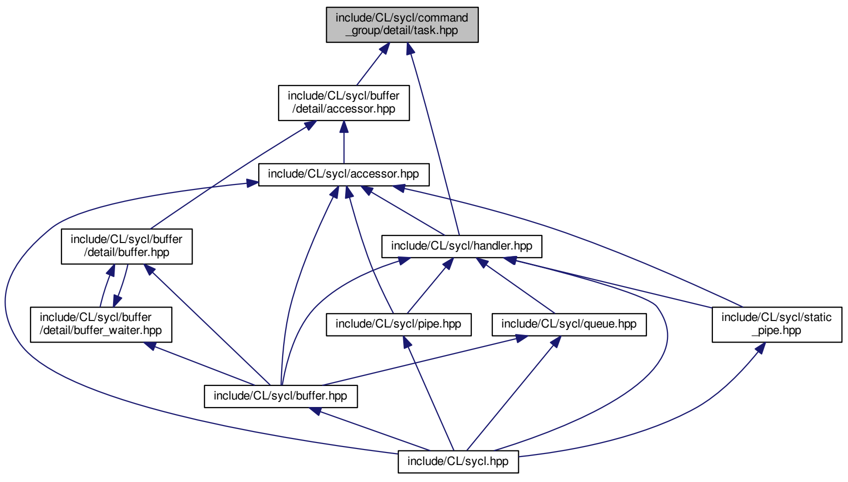
11.23 include/CL/sycl/command_group/detail/task.hpp File Reference

```
#include <condition_variable>
#include <memory>
#include <thread>
#include <boost/compute.hpp>
#include "CL/sycl/buffer/detail/buffer_base.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/kernel.hpp"
#include "CL/sycl/queue/detail/queue.hpp"
```

Include dependency graph for task.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- struct cl::sycl::detail::task

  The abstraction to represent SYCL tasks executing inside command_group.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.24 task.hpp

```
#ifndef TRISYCL_SYCL_TASK_HPP
#define TRISYCL_SYCL_TASK_HPP

/** \file The concept of task behind the scene

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <condition_variable>
#include <memory>
#include <thread>

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/buffer/detail/buffer_base.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/kernel.hpp"
#include "CL/sycl/queue/detail/queue.hpp"

namespace cl {
```

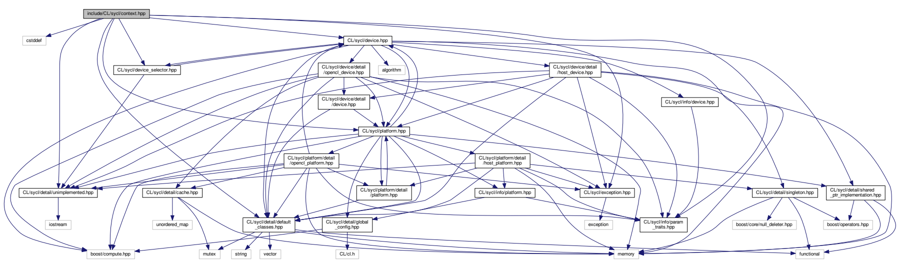
```
namespace sycl {
namespace detail {

/** The abstraction to represent SYCL tasks executing inside command_group

    "enable_shared_from_this" allows to access the shared_ptr behind the
    scene.
*/
struct task : public std::enable_shared_from_this<task>,
              public detail::debug<task> {

  /** List of the buffers used by this task

      \todo Use a set to check that some buffers are not used many
      times at least on writing
  */
  std::vector<std::shared_ptr<detail::buffer_base>> buffers_in_use;

  /// The tasks producing the buffers used by this task
  std::vector<std::shared_ptr<detail::task>> producer_tasks;

  /// Keep track of any prologue to be executed before the kernel
  std::vector<std::function<void(void)>> prologues;

  /// Keep track of any epilogue to be executed after the kernel
  std::vector<std::function<void(void)>> epilogues;

  /// Store if the execution ended, to be notified by task_ready
  bool execution_ended = false;

  /// To signal when this task is ready
  std::condition_variable ready;

  /// To protect the access to the condition variable
  std::mutex ready_mutex;

  /** Keep track of the queue used to submission to notify kernel completion
      or to run OpenCL kernels on */
  std::shared_ptr<detail::queue> owner_queue;

  std::shared_ptr<cl::sycl::detail::kernel> kernel;


  /// Create a task from a submitting queue
  task(const std::shared_ptr<detail::queue> &q)
    : owner_queue { q } {}


  /// Add a new task to the task graph and schedule for execution
  void schedule(std::function<void(void)> f) {
    /* To keep a copy of the task shared_ptr after the end of the
       command group, capture it by copy in the following lambda. This
       should be easier in C++17 with move semantics on capture
    */
    auto task = shared_from_this();
    auto execution = [=] {
      // Wait for the required tasks to be ready
      task->wait_for_producers();
      task->prelude();
      TRISYCL_DUMP_T("Execute the kernel");
      // Execute the kernel
      f();
      task->postlude();
      // Release the buffers that have been written by this task
      task->release_buffers();
      // Notify the waiting tasks that we are done
      task->notify_consumers();
      // Notify the queue we are done
      owner_queue->kernel_end();
      TRISYCL_DUMP_T("Task thread exit");
    };
    /* Notify the queue that there is a kernel submitted to the
       queue. Do not do it in the task contructor so that we can deal
       with command group without kernel and if we put it inside the
       thread, the queue may have finished before the thread is
       scheduled */
    owner_queue->kernel_start();
    /* \todo it may be implementable with packaged_task that would
       deal with exceptions in kernels
    */
#if TRISYCL_ASYNC
    /* If in asynchronous execution mode, execute the functor in a new
       thread */
    std::thread thread(execution);
    TRISYCL_DUMP_T("Task thread started");
    /** Detach the thread since it will synchronize by its own means

```

```
      \todo This is an issue if there is an exception in the kernel
    */
    thread.detach();
#else
    // Just a synchronous execution otherwise
    execution();
#endif
  }


  /// Wait for the required producer tasks to be ready
  void wait_for_producers() {
    TRISYCL_DUMP_T("Task " << this << " waits for the producer tasks");
    for (auto &t : producer_tasks)
      t->wait();
    // We can let the producers rest in peace
    producer_tasks.clear();
  }


  /// Release the buffers that have  been used by this task
  void release_buffers() {
    TRISYCL_DUMP_T("Task " << this << " releases the written buffers");
    for (auto b: buffers_in_use)
      b->release();
    buffers_in_use.clear();
  }


  /// Notify the waiting tasks that we are done
  void notify_consumers() {
    TRISYCL_DUMP_T("Notify all the task waiting for this task " << this);
    execution_ended = true;
    /* \todo Verify that the memory model with the notify does not
       require some fence or atomic */
    ready.notify_all();
  }


  /** Wait for this task to be ready

      This is to be called from another thread
  */
  void wait() {
    TRISYCL_DUMP_T("The task wait for task " << this << " to end");
    std::unique_lock<std::mutex> ul { ready_mutex };
    ready.wait(ul, [&] { return execution_ended; });
  }


  /** Register a buffer to this task

      This is how the dependency graph is incrementally built.
  */
  void add_buffer(std::shared_ptr<detail::buffer_base> &buf,
                  bool is_write_mode) {
    TRISYCL_DUMP_T("Add buffer " << buf << " in task " << this);
    /* Keep track of the use of the buffer to notify its release at
       the end of the execution */
    buffers_in_use.push_back(buf);
    // To be sure the buffer does not disappear before the kernel can run
    buf->use();

    std::shared_ptr<detail::task> latest_producer;
    if (is_write_mode) {
      /* Set this task as the latest producer of the buffer so that
         another kernel may wait on this task */
      latest_producer = buf->set_latest_producer(shared_from_this());
    }
    else
      latest_producer = buf->get_latest_producer();

    /* If the buffer is to be produced by a task, add the task in the
       producer list to wait on it before running the task core */
    if (latest_producer)
      producer_tasks.push_back(latest_producer);
  }


  /// Execute the prologues
  void prelude() {
    for (const auto &p : prologues)
      p();
    /* Free the functors that may own an accessor owning a buffer
       preventing the command group to complete */
    prologues.clear();
  }


  /// Execute the epilogues
  void postlude() {
    for (const auto &p : epilogues)
      p();
    /* Free the functors that may own an accessor owning a buffer
       preventing the command group to complete */
    epilogues.clear();
  }


  /// Add a function to the prelude to run before kernel execution
  void add_prelude(const std::function<void(void)> &f) {
    prologues.push_back(f);
  }


  /// Add a function to the postlude to run after kernel execution
  void add_postlude(const std::function<void(void)> &f) {
    epilogues.push_back(f);
  }


  /// Get the queue behind the task to run a kernel on
  auto get_queue() {
    return owner_queue;
  }


  /// Set the kernel running this task if any
  void set_kernel(const std::shared_ptr<cl::sycl::detail::kernel> &k) {
    kernel = k;
  }


  /** Get the kernel running if any

      \todo Specify this error in the spec
  */
  cl::sycl::detail::kernel &get_kernel() {
    if (!kernel)
      throw non_cl_error("Cannot use an OpenCL kernel in this context");
    return *kernel;
  }

};

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_TASK_HPP
```

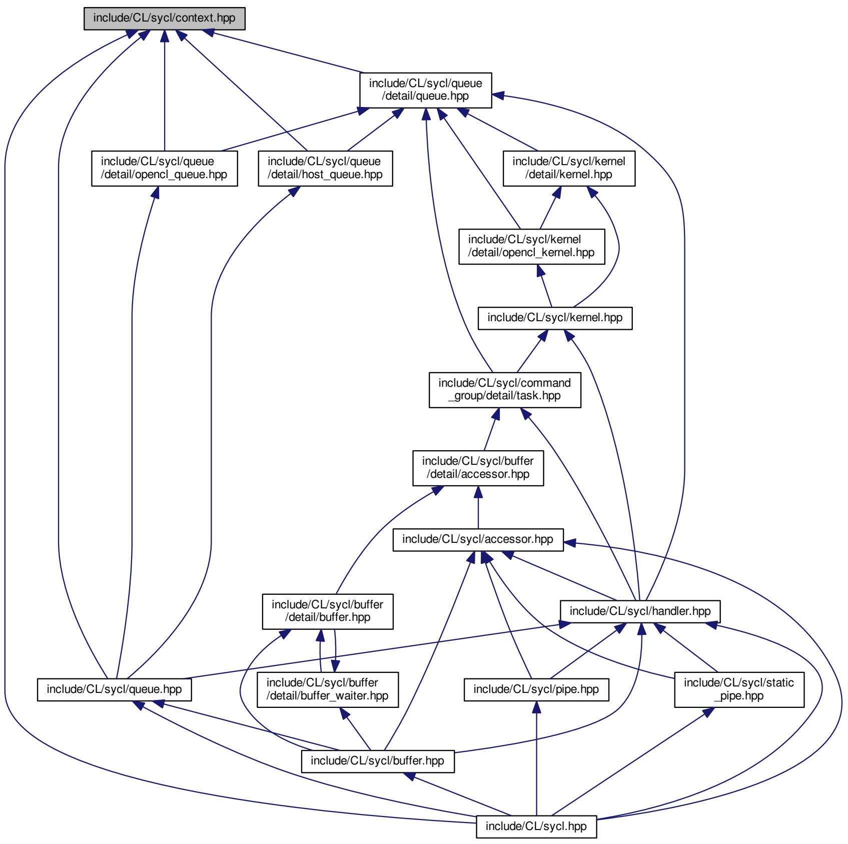
11.25 include/CL/sycl/context.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for context.hpp:


include/CL/sycl/context.hpp
cstddef
CL/sycl/device.hpp
CL/sycl/device_selector.hpp
CL/sycl/device/detail/opencl_device.hpp
algorithm
CL/sycl/device/detail/host_device.hpp
CL/sycl/device/detail/device.hpp
CL/sycl/info/device.hpp
CL/sycl/platform.hpp
CL/sycl/platform/detail/opencl_platform.hpp
CL/sycl/platform/detail/host_platform.hpp
CL/sycl/detail/unimplemented.hpp
CL/sycl/detail/cache.hpp
CL/sycl/platform/detail/platform.hpp
CL/sycl/info/platform.hpp
CL/sycl/exception.hpp
CL/sycl/detail/singleton.hpp
CL/sycl/detail/shared_ptr_implementation.hpp
iostream
unordered_map
CL/sycl/detail/default_classes.hpp
CL/sycl/detail/global_config.hpp
exception
CL/sycl/info/param_traits.hpp
boost/core/null_deleter.hpp
boost/operators.hpp
boost/compute.hpp
mutex
string
vector
CL/cl.h
memory
functional

This graph shows which files directly or indirectly include this file:


include/CL/sycl/context.hpp
include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::context

  SYCL context. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info

Typedefs

- using cl::sycl::info::gl_context_interop = bool

Enumerations

- enum cl::sycl::info::context : int { cl::sycl::info::context::reference_count, cl::sycl::info::context::num_devices, cl::sycl::info::context::gl_interop }

  Context information descriptors.

11.26 context.hpp

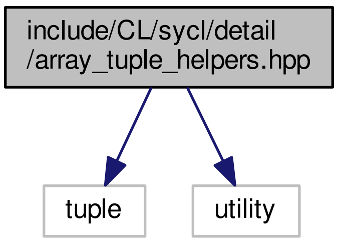
```
#ifndef TRISYCL_SYCL_CONTEXT_HPP
#define TRISYCL_SYCL_CONTEXT_HPP

/** \file The OpenCL SYCL context

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>

#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform.hpp"

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

namespace info {

using gl_context_interop = bool;

/** Context information descriptors

    \todo Should be unsigned int to be consistent with others?
*/
enum class context : int {
  reference_count,
  num_devices,
  gl_interop
};


/** Query the return type for get_info() on context stuff

    \todo To be implemented
*/
TRISYCL_INFO_PARAM_TRAITS_ANY_T(info::context, void)

}


/** SYCL context

    The context class encapsulates an OpenCL context, which is implicitly
    created and the lifetime of the context instance defines the lifetime
    of the underlying OpenCL context instance.

    On destruction clReleaseContext is called.

    The default context is the SYCL host context containing only the SYCL
    host device.

    \todo The implementation is quite minimal for now.
*/
class context {

public:

  /** Constructs a context object for SYCL host using an async_handler for
      handling asynchronous errors

      Note that the default case asyncHandler = nullptr is handled by the
      default constructor.
  */
  explicit context(async_handler asyncHandler) {
    detail::unimplemented();
  }


#ifdef TRISYCL_OPENCL
  /* Context constructor, where the underlying OpenCL context is given as
     a parameter

     The constructor executes a retain on the cl_context.

     Return synchronous errors via the SYCL exception class and
     asynchronous errors are handled via the async_handler, if provided.
  */
  context(cl_context clContext, async_handler asyncHandler = nullptr) {
    detail::unimplemented();
  }
#endif

  /** Constructs a context object using a device_selector object

      The context is constructed with a single device retrieved from the
      device_selector object provided.

      Return synchronous errors via the SYCL exception class and
      asynchronous errors are handled via the async_handler, if provided.
  */
  context(const device_selector &deviceSelector,
          info::gl_context_interop interopFlag,
          async_handler asyncHandler = nullptr) {
    detail::unimplemented();
  }


  /** Constructs a context object using a device object

      Return synchronous errors via the SYCL exception class and
      asynchronous errors are handled via the async_handler, if provided.
  */
  context(const device &dev,
          info::gl_context_interop interopFlag,
          async_handler asyncHandler = nullptr) {
    detail::unimplemented();
  }


  /** Constructs a context object using a platform object

      Return synchronous errors via the SYCL exception class and
      asynchronous errors are handled via the async_handler, if provided.
  */
  context(const platform &plt,
          info::gl_context_interop interopFlag,
          async_handler asyncHandler = nullptr) {
    detail::unimplemented();
  }


  /* Constructs a context object using a vector_class of device objects

     Return synchronous errors via the SYCL exception class and
     asynchronous errors are handled via the async_handler, if provided.

     \todo Update the specification to replace vector by collection
     concept.
  */
  context(const vector_class<device> &deviceList,
          info::gl_context_interop interopFlag,
          async_handler asyncHandler = nullptr) {
    detail::unimplemented();
  }

  /** Default constructor that chooses the context according the
      heuristics of the default selector

    Return synchronous errors via the SYCL exception class.

    Get the default constructors back.
  */
  context() = default;


#ifdef TRISYCL_OPENCL
  /* Returns the underlying cl_context object, after retaining the cl_context.

     Retains a reference to the returned cl_context object.

     Caller should release it when finished.
  */
  cl_context get() const {
    detail::unimplemented();
    return {};
  }
#endif


  /// Specifies whether the context is in SYCL Host Execution Mode.
  bool is_host() const {
    return true;
  }


  /** Returns the SYCL platform that the context is initialized for

      \todo To be implemented
  */
  platform get_platform();


  /** Returns the set of devices that are part of this context

      \todo To be implemented
  */
  vector_class<device> get_devices() const {
    detail::unimplemented();
    return {};
  }


  /** Queries OpenCL information for the under-lying cl context

      \todo To be implemented
  */
  template <info::context Param>
  typename info::param_traits<info::context, Param>::type
get_info() const {
    detail::unimplemented();
    return {};
  }

};

/// @} to end the execution Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_CONTEXT_HPP
```

11.27 include/CL/sycl/detail/array_tuple_helpers.hpp File Reference

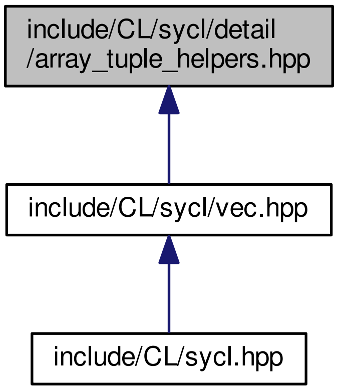
Some helpers to do array-tuple conversions.

```
#include <tuple>
#include <utility>
```

Include dependency graph for array_tuple_helpers.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct cl::sycl::detail::expand_to_vector< V, Tuple, expansion >

  Allows optional expansion of a 1-element tuple to a V::dimension tuple to replicate scalar values in vector initialization. More...
- struct cl::sycl::detail::expand_to_vector< V, Tuple, true >

  Specialization in the case we ask for expansion. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<typename V , typename Tuple , size_t... Is>
  std::array< typename V::element_type, V::dimension > cl::sycl::detail::tuple_to_array_iterate (Tuple t, std::index_sequence< Is... >)

  Helper to construct an array from initializer elements provided as a tuple.
- template<typename V , typename Tuple >
  auto cl::sycl::detail::tuple_to_array (Tuple t)

  Construct an array from initializer elements provided as a tuple.
- template<typename V , typename Tuple >
  auto cl::sycl::detail::expand (Tuple t)

  Create the array data of V from a tuple of initializer.

11.27.1 Detailed Description

Some helpers to do array-tuple conversions.

Used for example to implement cl::sycl::vec<> class.

Ronan at Keryell point FR

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file array_tuple_helpers.hpp.

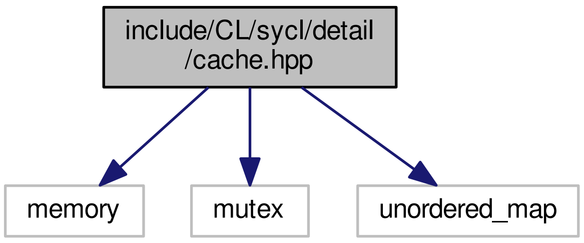
11.28 array_tuple_helpers.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_ARRAY_TUPLE_HELPERS_HPP
#define TRISYCL_SYCL_DETAIL_ARRAY_TUPLE_HELPERS_HPP

/** \file

    Some helpers to do array-tuple conversions

    Used for example to implement cl::sycl::vec<> class.

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <tuple>
#include <utility>

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup array_tuple_helpers Helpers to do array and tuple conversion

    @{
*/

/** Helper to construct an array from initializer elements provided as a
    tuple

    The trick is to get the std::index_sequence<> that represent 0,
    1,..., dimension-1 as a variadic template pack Is that we can
    iterate on, in this function.
*/
template <typename V, typename Tuple, size_t... Is>
std::array<typename V::element_type, V::dimension>
tuple_to_array_iterate(Tuple t, std::index_sequence<Is...>) {
  /* The effect is like a static for-loop with Is counting from 0 to
     dimension-1 and thus constructing a uniform initialization { }
     construction from each tuple element:
     { std::get<0>(t), std::get<1>(t), ..., std::get<dimension-1>(t) }

     The static cast is here to avoid the warning when there is a loss
     of precision, for example when initializing an int from a float.
  */
  return { { static_cast<typename V::element_type>(std::get<Is>(t))...} };
}


/** Construct an array from initializer elements provided as a tuple
 */
template <typename V, typename Tuple>
auto tuple_to_array(Tuple t) {
  /* Construct an index_sequence with 0, 1, ..., (size of the tuple-1)
     so that tuple_to_array_iterate can statically iterate on it */
  return tuple_to_array_iterate<V>(t,
                                   std::make_index_sequence<std::tuple_size<Tuple>::value>{});
}


/** Allows optional expansion of a 1-element tuple to a V::dimension
    tuple to replicate scalar values in vector initialization
*/
template <typename V, typename Tuple, bool expansion = false>
struct expand_to_vector {
  static_assert(V::dimension == std::tuple_size<Tuple>::value,
                "The number of elements in initialization should match the dimension of the vector");

  // By default, act as a pass-through and do not do any expansion
  static auto expand(Tuple t) { return t; }

};


/** Specialization in the case we ask for expansion */
template <typename V, typename Tuple>
struct expand_to_vector<V, Tuple, true> {
  static_assert(std::tuple_size<Tuple>::value == 1,
                "Since it is a vector initialization from a scalar there should be only one initializer
 value");


  /** Construct a tuple from a value

      \param value is used to initialize each tuple element

      \param size is the number of elements of the tuple to be generated

      The trick is to get the std::index_sequence<> that represent 0,
      1,..., dimension-1 as a variadic template pack Is that we can
      iterate on, in this function.
  */
  template <typename Value, size_t... Is>
  static auto fill_tuple(Value e, std::index_sequence<Is...>) {
    /* The effect is like a static for-loop with Is counting from 0 to
       dimension-1 and thus replicating the pattern to have
       make_tuple( (0, e), (1, e), ... (n - 1, e) )

       Since the "," operator is just here to throw away the Is value
       (which is needed for the pack expansion...), at the end this is
       equivalent to:
       make_tuple( e, e, ..., e )
    */
    return std::make_tuple(((void)Is, e)...);
  }


  /** We expand the 1-element tuple by replicating into a tuple with the
      size of the vector */
  static auto expand(Tuple t) {
    return fill_tuple(std::get<0>(t),
                      std::make_index_sequence<V::dimension>{});
  }

};


/** Create the array data of V from a tuple of initializer

    If there is only 1 initializer, this is a scalar initialization of a
    vector and the value is expanded to all the vector elements first.
*/
template <typename V, typename Tuple>
auto expand(Tuple t) {
  return tuple_to_array<V>(expand_to_vector<V,
                           decltype(t),
                            /* Only ask the expansion to all vector
                               element if there only a scalar
                               initializer */
                            std::tuple_size<Tuple>::value == 1>{}.expand(t));
}

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_ARRAY_TUPLE_HELPERS_HPP
```

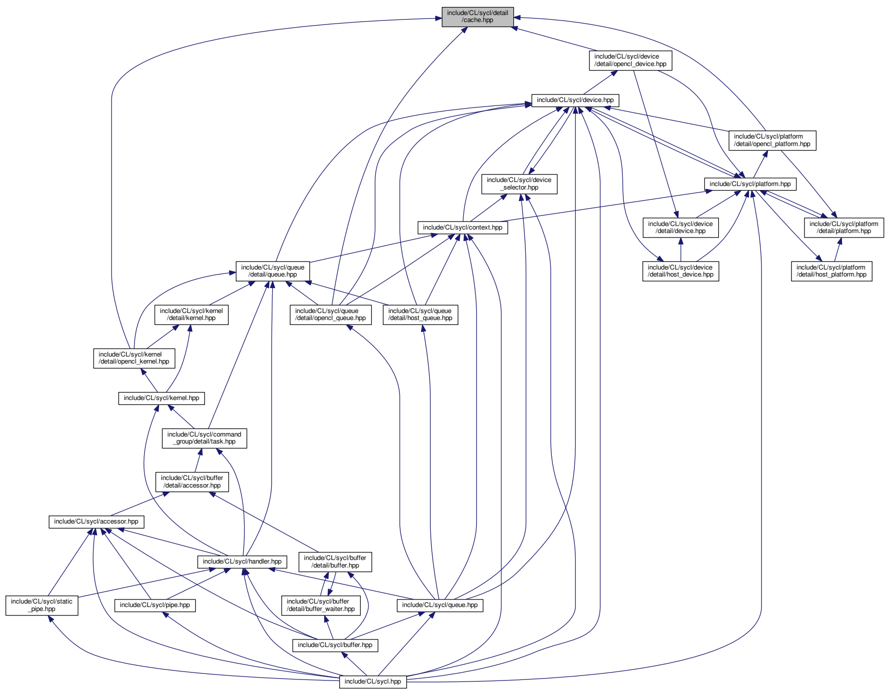
11.29 include/CL/sycl/detail/cache.hpp File Reference

```
#include <memory>
#include <mutex>
#include <unordered_map>
```

Include dependency graph for cache.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/detail/cache.hpp
include/CL/sycl/device/detail/opencl_device.hpp
include/CL/sycl/device.hpp
include/CL/sycl/platform/detail/opencl_platform.hpp
include/CL/sycl/device_selector.hpp
include/CL/sycl/platform.hpp
include/CL/sycl/context.hpp
include/CL/sycl/device/detail/device.hpp
include/CL/sycl/platform/detail/platform.hpp
include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/device/detail/host_device.hpp
include/CL/sycl/platform/detail/host_platform.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::detail::cache< Key, Value >

  A simple thread safe cache mechanism to cache std::shared_ptr of values indexed by keys.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.30 cache.hpp

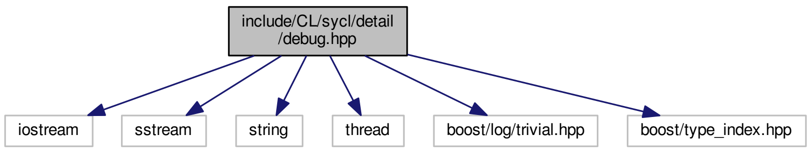
```
#ifndef TRISYCL_SYCL_DETAIL_CACHE_HPP
#define TRISYCL_SYCL_DETAIL_CACHE_HPP

/** \file A simple thread-safe cache

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <memory>
#include <mutex>
#include <unordered_map>

namespace cl {
namespace sycl {
```

```
namespace detail {


/** A simple thread safe cache mechanism to cache std::shared_ptr of
    values indexed by keys

    Since internally only std::weak_ptr are stored, this does not
    prevent object deletion but it is up to the programmer not to use
    this cache to retrieve deleted objects.
*/
template <typename Key, typename Value>
class cache {

public:

  /// The type of the keys used to indexed the cache
  using key_type = Key;

  /// The base type of the values stored in the cache
  using value_type = Value;

private:

  /// The caching storage
  std::unordered_map<key_type, std::weak_ptr<value_type>> c;

  /// To make the cache thread-safe
  std::mutex m;

public:

  /** Get a value stored in the cache if present or insert by calling
      a generator function

      \param[in] k is the key used to retrieve the value

      \param[in] create_element is the function to be called if the
      key is not found in the cache to generate a value which is
      inserted for the key. This function has to produce a value
      convertible to a shared_ptr

      \return a shared_ptr to the value retrieved or inserted
  */
  template <typename Functor>
  std::shared_ptr<value_type> get_or_register(const key_type &k,
                                              Functor &&create_element) {
    std::lock_guard<std::mutex> lg { m };

    auto i = c.find(k);
    if (i != c.end())
      // Return the found element
      return std::shared_ptr<value_type>{ i->second };

    // Otherwise create and insert a new element
    std::shared_ptr<value_type> e { create_element() };
    c.insert({ k, e });
    return e;
  }


  /** Remove an entry from the cache

      \param[in] k is the key associated to the value to remove from
      the cache
  */
  void remove(const key_type &k) {
    std::lock_guard<std::mutex> lg { m };
    c.erase(k);
  }

};

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_CACHE_HPP
```

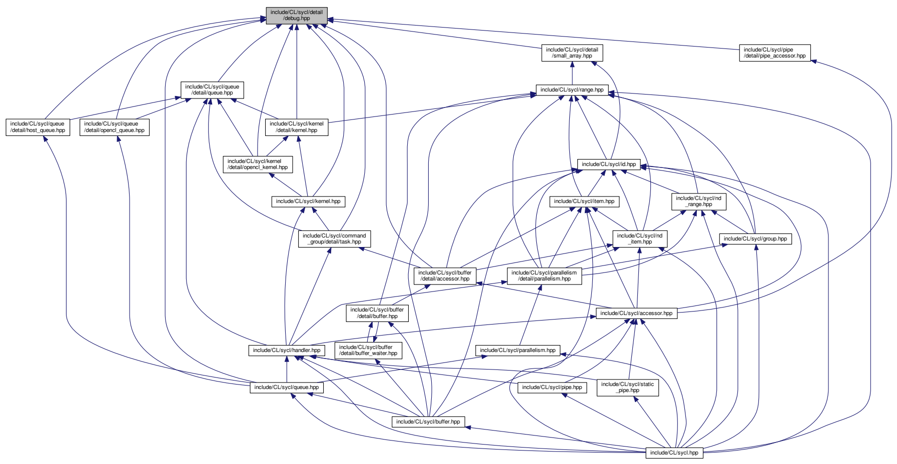
11.31 include/CL/sycl/detail/debug.hpp File Reference

```
#include <iostream>
#include <sstream>
#include <string>
#include <thread>
#include <boost/log/trivial.hpp>
#include <boost/type_index.hpp>
```

Include dependency graph for debug.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct cl::sycl::detail::debug< T >

  Class used to trace the construction, copy-construction, move-construction and destruction of classes that inherit from it. More...
- struct cl::sycl::detail::display_vector< T >

  Class used to display a vector-like type of classes that inherit from it. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Macros

- #define TRISYCL_INTERNAL_DUMP(expression)

  Dump a debug message in a formatted way.
- #define TRISYCL_DUMP(expression) TRISYCL_INTERNAL_DUMP(expression)
- #define TRISYCL_DUMP_T(expression)

  Same as TRISYCL_DUMP() but with thread id first.

Functions

- template<typename KernelName , typename Functor >
  auto cl::sycl::detail::trace_kernel (const Functor &f)

  Wrap a kernel functor in some tracing messages to have start/stop information when TRISYCL_TRACE_KERNEL macro is defined.

11.31.1 Macro Definition Documentation

11.31.1.1 #define TRISYCL_DUMP(*expression*) TRISYCL_INTERNAL_DUMP(expression)

Definition at line 43 of file debug.hpp.

11.31.1.2 #define TRISYCL_DUMP_T(*expression*)

Value:

```
TRISYCL_DUMP("Thread " << std::hex                                  \
               << std::this_thread::get_id() << ": " << expression)
```

Same as TRISYCL_DUMP() but with thread id first.

Definition at line 46 of file debug.hpp.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor(), cl::sycl::detail::task::add↩_buffer(), cl::sycl::detail::pipe_reservation< PipeAccessor >::commit(), cl::sycl::detail::pipe< value_type >↩::empty(), cl::sycl::detail::queue::kernel_end(), cl::sycl::detail::queue::kernel_start(), cl::sycl::detail::task::notify_↩consumers(), cl::sycl::detail::pipe_reservation< PipeAccessor >::operator[ ](), cl::sycl::detail::pipe< value_type >::read(), cl::sycl::detail::task::release_buffers(), cl::sycl::detail::pipe< value_type >::reserve_read(), cl::sycl↩::detail::pipe< value_type >::reserve_write(), cl::sycl::detail::task::schedule(), cl::sycl::detail::pipe< value_type >::size(), cl::sycl::detail::task::wait(), cl::sycl::detail::queue::wait_for_kernel_execution(), cl::sycl::detail::task::wait↩_for_producers(), cl::sycl::detail::pipe< value_type >::write(), and cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >::~buffer_waiter().

11.31.1.3 #define TRISYCL_INTERNAL_DUMP(*expression*)

Value:

```
do {        \
    std::ostringstream s;                           \
    s << expression;                                \
    BOOST_LOG_TRIVIAL(debug) << s.str();            \
  } while(0)
```

Dump a debug message in a formatted way.

Use an intermediate ostringstream because there are issues with BOOST_LOG_TRIVIAL to display C strings

Definition at line 35 of file debug.hpp.

Referenced by cl::sycl::detail::trace_kernel().

11.32 debug.hpp

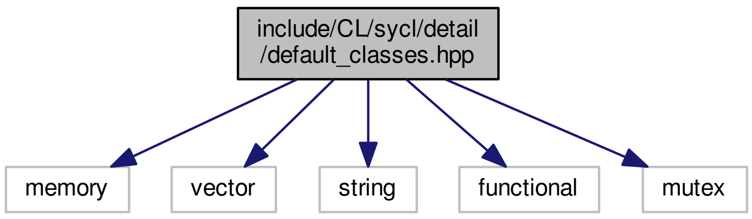
```
#ifndef TRISYCL_SYCL_DETAIL_DEBUG_HPP
#define TRISYCL_SYCL_DETAIL_DEBUG_HPP

/** \file Track constructor/destructor invocations and trace kernel execution

    Define the TRISYCL_DEBUG CPP flag to have an output.

    To use it in some class C, make C inherit from debug<C>.

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <iostream>

// The common debug and trace infrastructure
#if defined(TRISYCL_DEBUG) || defined(TRISYCL_TRACE_KERNEL)
#include <sstream>
#include <string>
#include <thread>

#include <boost/log/trivial.hpp>
#include <boost/type_index.hpp>

// To be able to construct string literals like "blah"s
using namespace std::string_literals;

/** Dump a debug message in a formatted way.

    Use an intermediate ostringstream because there are issues with
    BOOST_LOG_TRIVIAL to display C strings
*/
#define TRISYCL_INTERNAL_DUMP(expression) do {        \
    std::ostringstream s;                             \
    s << expression;                                  \
    BOOST_LOG_TRIVIAL(debug) << s.str();              \
  } while(0)
#endif

#ifdef TRISYCL_DEBUG
#define TRISYCL_DUMP(expression) TRISYCL_INTERNAL_DUMP(expression)

/// Same as TRISYCL_DUMP() but with thread id first
#define TRISYCL_DUMP_T(expression)                                        \
  TRISYCL_DUMP("Thread " << std::hex                                      \
                 << std::this_thread::get_id() << ": " << expression)
#else
#define TRISYCL_DUMP(expression) do { } while(0)
#define TRISYCL_DUMP_T(expression) do { } while(0)
#endif

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup debug_trace Debugging and tracing support
    @{
*/

/** Class used to trace the construction, copy-construction,
    move-construction and destruction of classes that inherit from it

    \param T is the real type name to be used in the debug output.
*/
template <typename T>
struct debug {
  // To trace the execution of the conSTRUCTORs and deSTRUCTORs
#ifdef TRISYCL_DEBUG_STRUCTORS
  /// Trace the construction with the compiler-dependent mangled named
  debug() {
    TRISYCL_DUMP("Constructor of "
                 << boost::typeindex::type_id<T>().pretty_name()
                 << " " << (void*) this);
  }


  /** Trace the copy construction with the compiler-dependent mangled
      named

      Only add this constructor if T has itself the same constructor,
      otherwise it may prevent the synthesis of default copy
      constructor and assignment.
  */
  template <typename U = T>
  debug(debug const &,
        /* Use intermediate U type to have the type dependent for
           enable_if to work

        \todo Use is_copy_constructible_v when moving to C++17 */
        std::enable_if_t<std::is_copy_constructible<U>::value> * = 0) {
    TRISYCL_DUMP("Copy of " << boost::typeindex::type_id<T>().pretty_name()
                 << " " << (void*) this);
  }


  /** Trace the move construction with the compiler-dependent mangled
      named

      Only add this constructor if T has itself the same constructor,
      otherwise it may prevent the synthesis of default move
      constructor and move assignment.
  */
  template <typename U = T>
  debug(debug &&,
        /* Use intermediate U type to have the type dependent for
           enable_if to work

        \todo Use is_move_constructible_v when moving to C++17 */
        std::enable_if_t<std::is_move_constructible<U>::value> * = 0) {
    TRISYCL_DUMP("Move of " << boost::typeindex::type_id<T>().pretty_name()
                 << " " << (void*) this);
  }


  /// Trace the destruction with the compiler-dependent mangled named
  ~debug() {
    TRISYCL_DUMP("~ Destructor of "
                 << boost::typeindex::type_id<T>().pretty_name()
                 << " " << (void*) this);
  }
#endif
};


/** Wrap a kernel functor in some tracing messages to have start/stop
    information when TRISYCL_TRACE_KERNEL macro is defined */
template <typename KernelName, typename Functor>
auto trace_kernel(const Functor &f) {
#ifdef TRISYCL_TRACE_KERNEL
  // Inject tracing message around the kernel
  return [=] {
    /* Since the class KernelName may just be declared and not really
       defined, just use it through a class pointer to have
       typeid().name() not complaining */
    TRISYCL_INTERNAL_DUMP(
      "Kernel started "
      << boost::typeindex::type_id<KernelName *>().pretty_name());
    f();
    TRISYCL_INTERNAL_DUMP(
      "Kernel stopped "
      << boost::typeindex::type_id<KernelName *>().pretty_name());
  };
#else
  // Identity by default
  return f;
#endif
}


/** Class used to display a vector-like type of classes that inherit from
    it

    \param T is the real type name to be used in the debug output.

    Calling the display() method dump the values on std::cout
*/
template <typename T>
struct display_vector {

  /// To debug and test
  void display() const {
#ifdef TRISYCL_DEBUG
    std::cout << boost::typeindex::type_id<T>().pretty_name() << ":";
#endif
    // Get a pointer to the real object
    for (auto e : *static_cast<const T *>(this))
      std::cout << " " << e;
    std::cout << std::endl;
  }
```

```

};

/// @} End the debug_trace Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_DEBUG_HPP
```

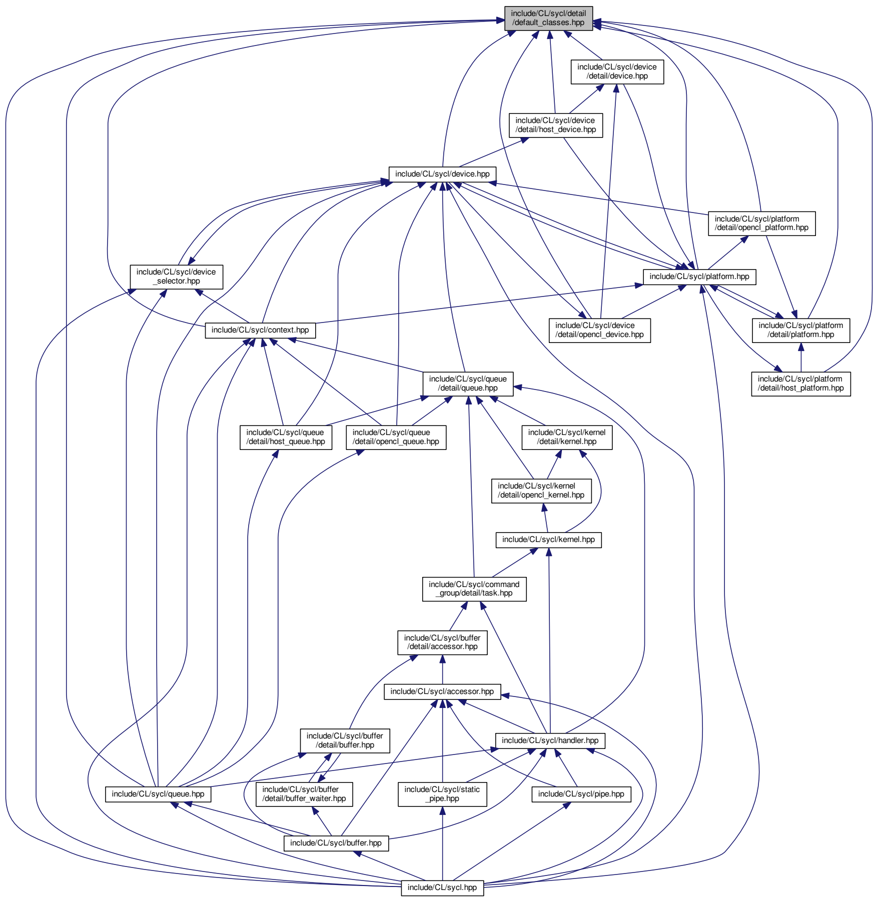
11.33 include/CL/sycl/detail/default_classes.hpp File Reference

```
#include <memory>
#include <vector>
#include <string>
#include <functional>
#include <mutex>
```

Include dependency graph for default_classes.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/detail/default_classes.hpp
include/CL/sycl/device/detail/device.hpp
include/CL/sycl/device/detail/host_device.hpp
include/CL/sycl/device.hpp
include/CL/sycl/platform/detail/opencl_platform.hpp
include/CL/sycl/device_selector.hpp
include/CL/sycl/platform.hpp
include/CL/sycl/context.hpp
include/CL/sycl/device/detail/opencl_device.hpp
include/CL/sycl/platform/detail/platform.hpp
include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/platform/detail/host_platform.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Typedefs

- template<class T , class Alloc = std::allocator<T>>
  using cl::sycl::vector_class = std::vector< T, Alloc >
- using cl::sycl::string_class = std::string
- template<class R , class... ArgTypes>
  using cl::sycl::function_class = std::function< R(ArgTypes...)>
- using cl::sycl::mutex_class = std::mutex
- template<class T , class D = std::default_delete<T>>
  using cl::sycl::unique_ptr_class = std::unique_ptr< T[ ], D >
- template<class T >
  using cl::sycl::shared_ptr_class = std::shared_ptr< T >
- template<class T >
  using cl::sycl::weak_ptr_class = std::weak_ptr< T >

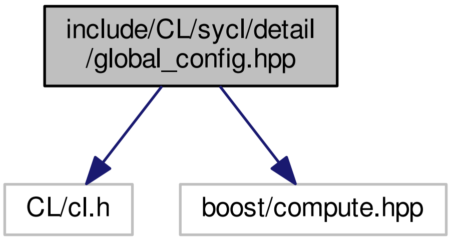
11.34 default_classes.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_DEFAULT_CLASSES_HPP
#define TRISYCL_SYCL_DETAIL_DEFAULT_CLASSES_HPP

/** \file The OpenCL SYCL default classes to use from the STL according to
    section 3.2 of SYCL 1.2 specification

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

/** \addtogroup defaults Manage default configuration and types
    @{
*/

#ifndef CL_SYCL_NO_STD_VECTOR
/** The vector type to be used as SYCL vector
 */
#include <memory>
#include <vector>
namespace cl {
namespace sycl {

template <class T, class Alloc = std::allocator<T>>
using vector_class = std::vector<T, Alloc>;

}
}
#endif


#ifndef CL_SYCL_NO_STD_STRING
/** The string type to be used as SYCL string
 */
#include <string>
namespace cl {
namespace sycl {

using string_class = std::string;

}
}
#endif


#ifndef CL_SYCL_NO_STD_FUNCTION
/** The functional type to be used as SYCL function
 */
#include <functional>
namespace cl {
namespace sycl {

template <class R, class... ArgTypes>
using function_class = std::function<R(ArgTypes...)>;

}
}
#endif


#ifndef CL_SYCL_NO_STD_MUTEX
/** The mutex type to be used as SYCL mutex
 */
#include <mutex>
namespace cl {
namespace sycl {

using mutex_class = std::mutex;

}
}
#endif


#ifndef CL_SYCL_NO_STD_UNIQUE_PTR
/** The unique pointer type to be used as SYCL unique pointer
 */
#include <memory>
namespace cl {
namespace sycl {

template <class T, class D = std::default_delete<T>>
using unique_ptr_class = std::unique_ptr<T[], D>;

}
}
#endif


#ifndef CL_SYCL_NO_STD_SHARED_PTR
/** The shared pointer type to be used as SYCL shared pointer
 */
#include <memory>
namespace cl {
namespace sycl {

template <class T>
using shared_ptr_class = std::shared_ptr<T>;

}
}
#endif


#ifndef CL_SYCL_NO_STD_WEAK_PTR
/** The weak pointer type to be used as SYCL weak pointer
 */
#include <memory>
namespace cl {
namespace sycl {

template <class T>
using weak_ptr_class = std::weak_ptr<T>;

}
}
#endif

/// @} End the defaults Doxygen group

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_DEFAULT_CLASSES_HPP
```

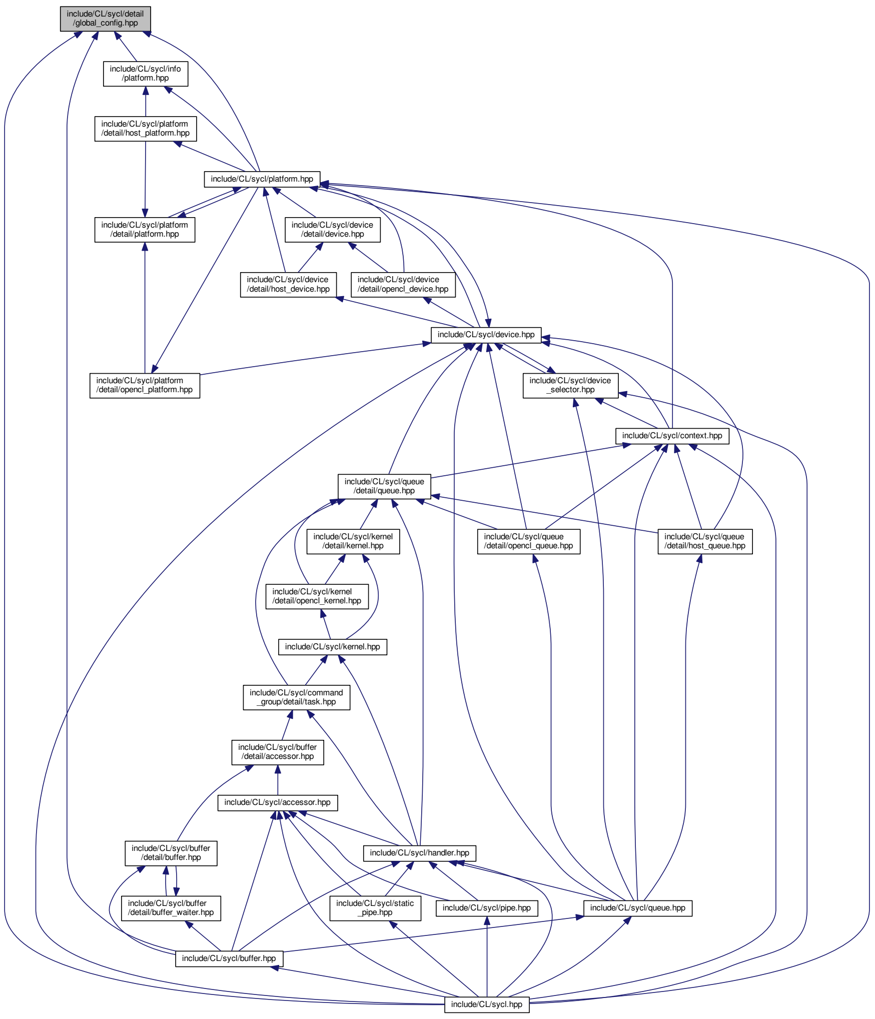
11.35 include/CL/sycl/detail/global_config.hpp File Reference

```
#include <CL/cl.h>
#include <boost/compute.hpp>
```

Include dependency graph for global_config.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define CL_SYCL_LANGUAGE_VERSION 220

  This implement SYCL 2.2.
- #define CL_TRISYCL_LANGUAGE_VERSION 220

  This implement triSYCL 2.2.
- #define __SYCL_SINGLE_SOURCE__

  This source is compiled by a single source compiler.
- #define TRISYCL_SKIP_OPENCL(x) x

  Define TRISYCL_OPENCL to add OpenCL.
- #define TRISYCL_ASYNC 0

  Allow the asynchronous implementation of tasks.

11.36 global_config.hpp

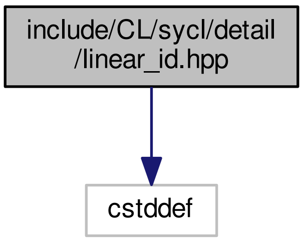
```
00001 #ifndef TRISYCL_SYCL_DETAIL_GLOBAL_CONFIG_HPP
```

```
#define TRISYCL_SYCL_DETAIL_GLOBAL_CONFIG_HPP

/** \file The OpenCL SYCL details on the global triSYCL configuration

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

/** \addtogroup defaults Manage default configuration and types
    @{
*/

// The following symbols can be set to implement a different version
#ifndef CL_SYCL_LANGUAGE_VERSION
/// This implement SYCL 2.2
#define CL_SYCL_LANGUAGE_VERSION 220
#endif

#ifndef CL_TRISYCL_LANGUAGE_VERSION
/// This implement triSYCL 2.2
#define CL_TRISYCL_LANGUAGE_VERSION 220
#endif

/// This source is compiled by a single source compiler
#define __SYCL_SINGLE_SOURCE__


/** Define TRISYCL_OPENCL to add OpenCL

    triSYCL can indeed work without OpenCL if only host support is needed.
*/
#ifdef TRISYCL_OPENCL

// SYCL interoperation API with OpenCL requires some OpenCL C types:
#if defined(__APPLE__)
#include <OpenCL/cl.h>
#else
#include <CL/cl.h>
#endif
// But the triSYCL OpenCL implementation is actually based on Boost.Compute
#include <boost/compute.hpp>
/// A macro to keep some stuff in OpenCL mode
#define TRISYCL_SKIP_OPENCL(x) x
#else
/// A macro to skip stuff when not supporting OpenCL
#define TRISYCL_SKIP_OPENCL(x)
#endif

/** Allow the asynchronous implementation of tasks */
#ifndef TRISYCL_ASYNC
/** Use asynchronous tasks by default.

    Is set to 0, the functors are executed synchronously.
*/
#define TRISYCL_ASYNC 0
#endif

/// @} End the defaults Doxygen group

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_GLOBAL_CONFIG_HPP
```

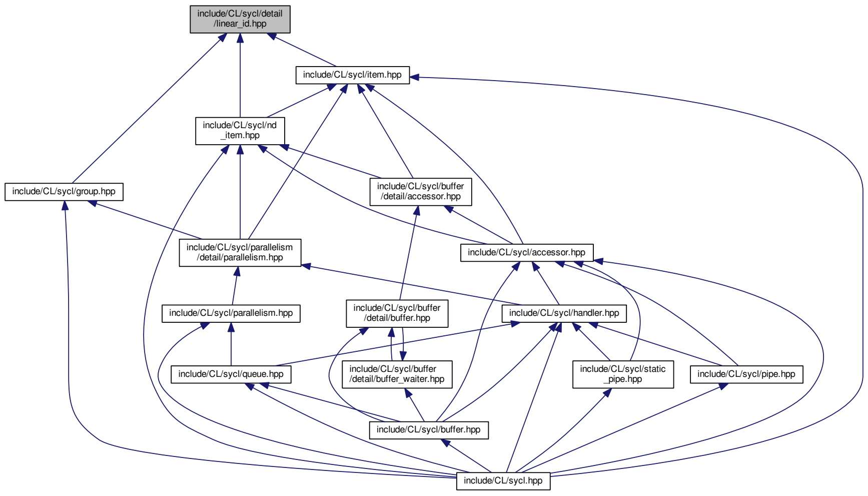
11.37 include/CL/sycl/detail/linear_id.hpp File Reference

```
#include <cstddef>
```

Include dependency graph for linear_id.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<typename Range , typename Id >

  size_t constexpr cl::sycl::detail::linear_id (Range range, Id id, Id offset={})

  Compute a linearized array access used in the OpenCL 2 world.

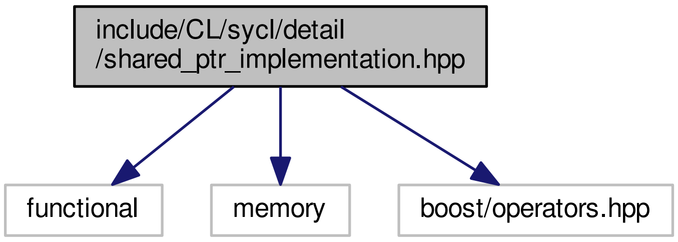
11.38 linear_id.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_LINEAR_ID_HPP
#define TRISYCL_SYCL_DETAIL_LINEAR_ID_HPP

/** \file Compute linearized array access

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup helpers Some helpers for the implementation
    @{
*/

/** Compute a linearized array access used in the OpenCL 2 world

    Typically for the get_global_linear_id() and get_local_linear_id()
    functions.
*/
template <typename Range, typename Id>
size_t constexpr inline linear_id(Range range, Id id, Id offset = {}) {
  auto dims = std::distance(std::begin(range), std::end(range));

  size_t linear_id = 0;
  /* A good compiler should unroll this and do partial evaluation to
     remove the first multiplication by 0 of this Horner evaluation and
     remove the 0 offset evaluation */
    for (int i = dims - 1; i >= 0; --i)
      linear_id = linear_id*range[i] + id[i] - offset[i];

    return linear_id;
  }


/// @} End the helpers Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_LINEAR_ID_HPP
```

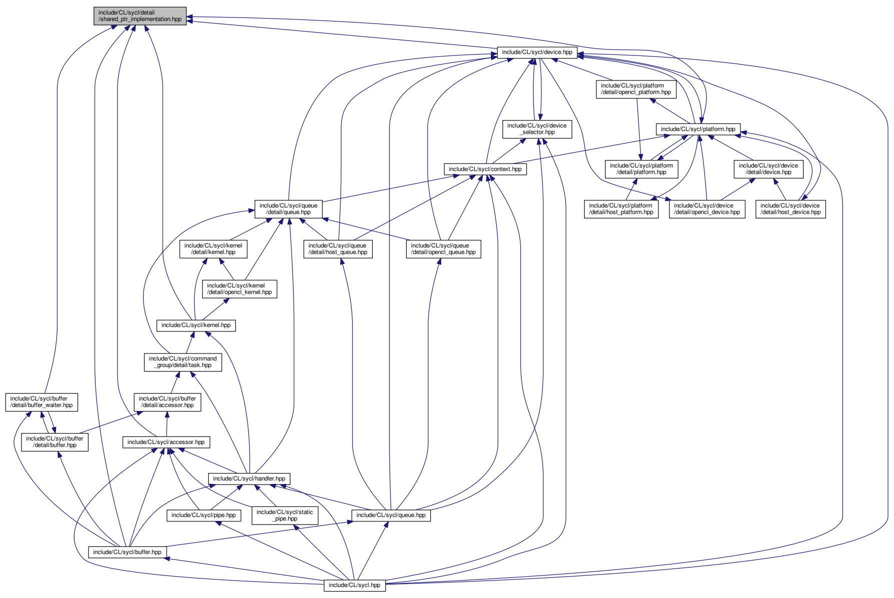
11.39 include/CL/sycl/detail/shared_ptr_implementation.hpp File Reference

```
#include <functional>
#include <memory>
#include <boost/operators.hpp>
```

Include dependency graph for shared_ptr_implementation.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >

    Provide an implementation as shared_ptr with total ordering and hashing to be used with algorithms and in (un)ordered containers.

Namespaces

- cl

    The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

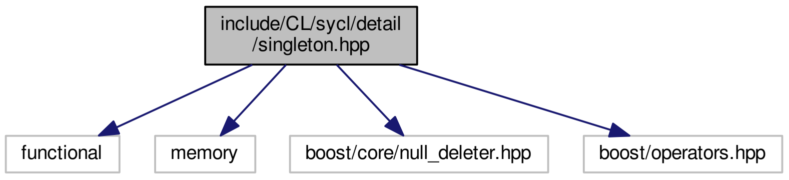
11.40 shared_ptr_implementation.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_SHARED_PTR_IMPLEMENTATION_HPP
#define TRISYCL_SYCL_DETAIL_SHARED_PTR_IMPLEMENTATION_HPP

/** \file Mix-in to add an implementation as shared_ptr with total
    ordering and hashing so that the class can be used with algorithms
    and in (un)ordered containers

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <functional>
#include <memory>

#include <boost/operators.hpp>

namespace cl {
namespace sycl {
namespace detail {

/** Provide an implementation as shared_ptr with total ordering and
    hashing to be used with algorithms and in (un)ordered containers

    To be used, a Parent class wanting an Implementation needs to
    inherit from.

    The implementation ends up in a member really named
    "implementation".

    \code
    public detail::shared_ptr_implementation<Parent, Implementation>
    \endcode

    and also inject in std namespace a specialization for
    \code hash<Parent> \endcode
*/
template <typename Parent, typename Implementation>
struct shared_ptr_implementation : public boost::totally_ordered<Parent> {

  /// The implementation forward everything to this... implementation
  std::shared_ptr<Implementation> implementation;

public:

  /// The implementation directly as a shared pointer
  shared_ptr_implementation(std::shared_ptr<Implementation> i)
    : implementation { i } {}


  /// The implementation takes the ownership from a raw pointer
  shared_ptr_implementation(Implementation *i) : implementation { i } {}


  /// Keep all other constructors to have usual shared_ptr behaviour
  shared_ptr_implementation() = default;


  /** Equality operator

      This is generalized by boost::equality_comparable from
      boost::totally_ordered to implement the equality comparable
      concept
  */
  bool operator ==(const Parent &other) const {
    return implementation == other.implementation;
  }


  /** Inferior operator

      This is generalized by boost::less_than_comparable from
      boost::totally_ordered to implement the equality comparable
      concept

      \todo Add this to the spec
  */
  bool operator <(const Parent &other) const {
    return implementation < other.implementation;
  }


  /// Forward the hashing for unordered containers to the implementation
  auto hash() const {
    return std::hash<decltype(implementation)>{}(implementation);
  }

};

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_SHARED_PTR_IMPLEMENTATION_HPP
```

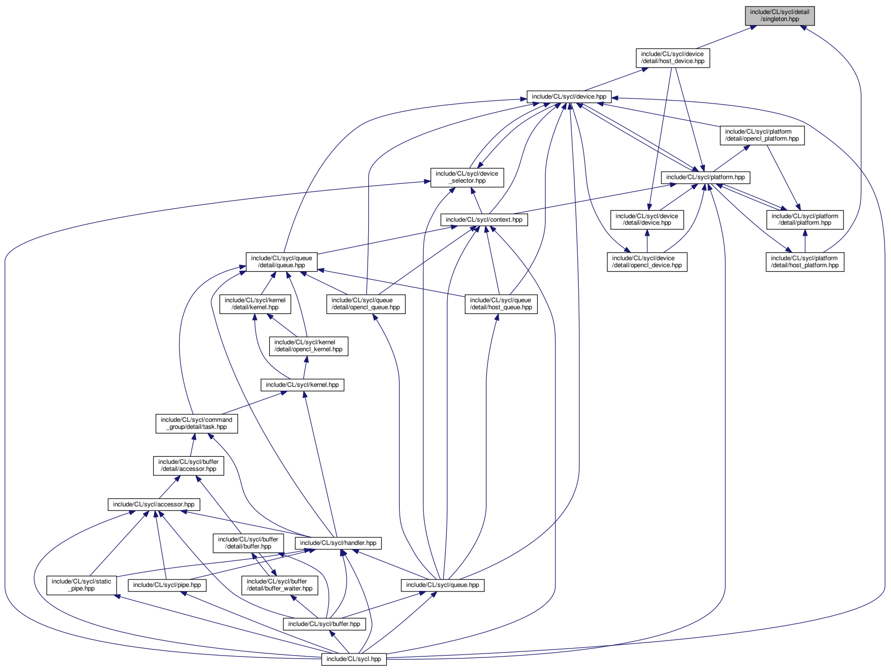
11.41 include/CL/sycl/detail/singleton.hpp File Reference

```
#include <functional>
#include <memory>
#include <boost/core/null_deleter.hpp>
#include <boost/operators.hpp>
```

Include dependency graph for singleton.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/detail/singleton.hpp
include/CL/sycl/device/detail/host_device.hpp
include/CL/sycl/device.hpp
include/CL/sycl/platform/detail/opencl_platform.hpp
include/CL/sycl/device_selector.hpp
include/CL/sycl/platform.hpp
include/CL/sycl/context.hpp
include/CL/sycl/device/detail/device.hpp
include/CL/sycl/platform/detail/platform.hpp
include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/device/detail/opencl_device.hpp
include/CL/sycl/platform/detail/host_platform.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- struct cl::sycl::detail::singleton< T >

    Provide a singleton factory.

Namespaces

- cl

    The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.42 singleton.hpp

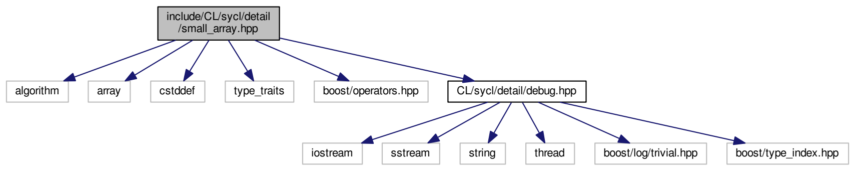
```
#ifndef TRISYCL_SYCL_DETAIL_SINGLETON_HPP
#define TRISYCL_SYCL_DETAIL_SINGLETON_HPP

/** \file Mix-in to add a singleton implementation with an instance() method

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <functional>
#include <memory>

#include <boost/core/null_deleter.hpp>
#include <boost/operators.hpp>


```

```
namespace cl {
namespace sycl {
namespace detail {

/// Provide a singleton factory
template <typename T>
struct singleton {

  /// Get a singleton instance of T
  static std::shared_ptr<T> instance() {
    // C++11 guaranties the static construction is thread-safe
    static T single;
    /** Use a null_deleter since the singleton should not be deleted,
        as allocated in the static area */
    static std::shared_ptr<T> sps { &single,
                                    boost::null_deleter {} };

    return sps;
  }

};

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_SINGLETON_HPP
```

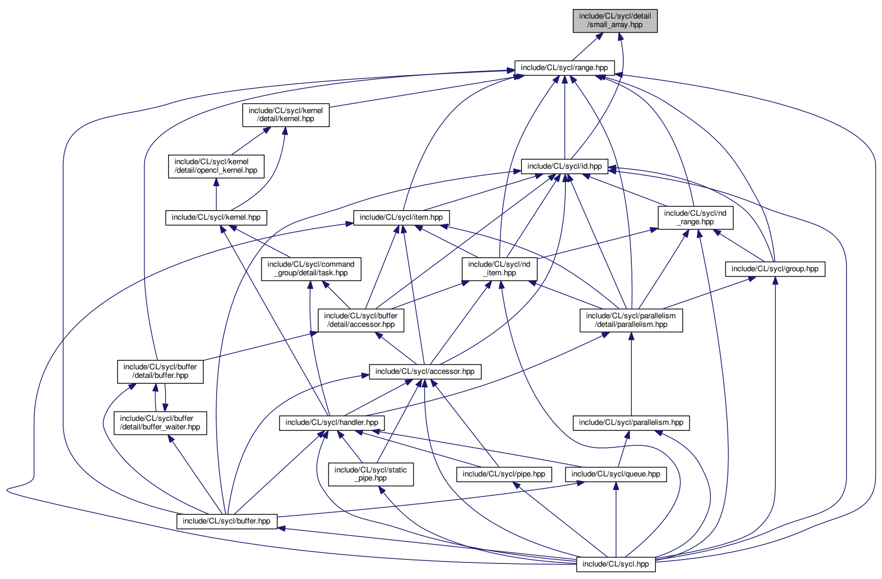
11.43 include/CL/sycl/detail/small_array.hpp File Reference

```
#include <algorithm>
#include <array>
#include <cstddef>
#include <type_traits>
#include <boost/operators.hpp>
#include "CL/sycl/detail/debug.hpp"
```

Include dependency graph for small_array.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >

  Define a multi-dimensional index, used for example to locate a work item or a buffer element. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >

  A small array of 1, 2 or 3 elements with the implicit constructors. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >

  Use some specializations so that some function overloads can be determined according to some implicit constructors and to have an implicit conversion from/to BasicType (such as an int typically) if dims = 1. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Macros

- #define TRISYCL_BOOST_OPERATOR_VECTOR_OP(op)

  Helper macro to declare a vector operation with the given side-effect operator.

11.44 small_array.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_SMALL_ARRAY_HPP
#define TRISYCL_SYCL_DETAIL_SMALL_ARRAY_HPP

/** \file This is a small array class to build range<>, id<>, etc.

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <algorithm>
#include <array>
#include <cstddef>
#include <type_traits>

#include <boost/operators.hpp>

#include "CL/sycl/detail/debug.hpp"


namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup helpers Some helpers for the implementation
    @{
*/


/** Helper macro to declare a vector operation with the given side-effect
    operator */
#define TRISYCL_BOOST_OPERATOR_VECTOR_OP(op)              \
  FinalType operator op(const FinalType &rhs) {           \
    for (std::size_t i = 0; i != Dims; ++i)               \
      (*this)[i] op rhs[i];                               \
    return *this;                                         \
  }


/** Define a multi-dimensional index, used for example to locate a work
    item or a buffer element

    Unfortunately, even if std::array is an aggregate class allowing
    native list initialization, it is no longer an aggregate if we derive
    from an aggregate. Thus we have to redeclare the constructors.

    \param BasicType is the type element, such as int

    \param Dims is the dimension number, typically between 1 and 3

    \param FinalType is the final type, such as range<> or id<>, so that
    boost::operator can return the right type

    \param EnableArgsConstructor adds a constructors from Dims variadic
    elements when true. It is false by default.

    std::array<> provides the collection concept, with .size(), == and !=
    too.
*/
template <typename BasicType,
          typename FinalType,
          std::size_t Dims,
          bool EnableArgsConstructor = false>
struct small_array : std::array<BasicType, Dims>,
  // To have all the usual arithmetic operations on this type
  boost::euclidean_ring_operators<FinalType>,
  // Bitwise operations
  boost::bitwise<FinalType>,
  // Shift operations
  boost::shiftable<FinalType>,
  // Already provided by array<> lexicographically:
  // boost::equality_comparable<FinalType>,
  // boost::less_than_comparable<FinalType>,
  // Add a display() method
  detail::display_vector<FinalType> {

  /// \todo add this Boost::multi_array or STL concept to the
  /// specification?
  static const auto dimensionality = Dims;

  /* Note that constexpr size() from the underlying std::array provides
     the same functionality */
  static const size_t dimension = Dims;
```

```
using element_type = BasicType;


/** A constructor from another array

    Make it explicit to avoid spurious range<> constructions from int *
    for example
*/
template <typename SourceType>
small_array(const SourceType src[Dims]) {
  // (*this)[0] is the first element of the underlying array
  std::copy_n(src, Dims, &(*this)[0]);
}


/// A constructor from another small_array of the same size
template <typename SourceBasicType,
          typename SourceFinalType,
          bool SourceEnableArgsConstructor>
small_array(const small_array<SourceBasicType,
            SourceFinalType,
            Dims,
            SourceEnableArgsConstructor> &src) {
  std::copy_n(&src[0], Dims, &(*this)[0]);
}


/** Initialize the array from a list of elements

    Strangely, even when using the array constructors, the
    initialization of the aggregate is not available. So recreate an
    equivalent here.

    Since there are inherited types that defines some constructors with
    some conflicts, make it optional here, according to
    EnableArgsConstructor template parameter.
 */
template <typename... Types,
          // Just to make enable_if depend of the template and work
          bool Depend = true,
          typename = typename std::enable_if_t<EnableArgsConstructor
                                               && Depend>>
small_array(const Types &... args)
  : std::array<BasicType, Dims> {
  // Allow a loss of precision in initialization with the static_cast
  { static_cast<BasicType>(args)... }
}
{
  static_assert(sizeof...(args) == Dims,
                "The number of initializing elements should match "
                "the dimension");
}


/// Construct a small_array from a std::array
template <typename SourceBasicType>
small_array(const std::array<SourceBasicType, Dims> &src)
: std::array<BasicType, Dims>(src) {}


/// Keep other constructors from the underlying std::array
using std::array<BasicType, Dims>::array;

/// Keep the synthesized constructors
small_array() = default;

/// Return the element of the array
auto get(std::size_t index) const {
  return (*this)[index];
}

/* Implement minimal methods boost::euclidean_ring_operators needs to
   generate everything */
/// Add + like operations on the id<> and others
TRISYCL_BOOST_OPERATOR_VECTOR_OP(+=)

/// Add - like operations on the id<> and others
TRISYCL_BOOST_OPERATOR_VECTOR_OP(-=)

/// Add * like operations on the id<> and others
TRISYCL_BOOST_OPERATOR_VECTOR_OP(*=)

/// Add / like operations on the id<> and others
TRISYCL_BOOST_OPERATOR_VECTOR_OP(/=)

/// Add % like operations on the id<> and others
TRISYCL_BOOST_OPERATOR_VECTOR_OP(%=)
```

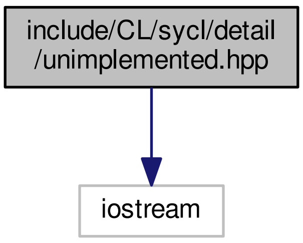
```

  /// Add << like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(<<=)

  /// Add >> like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(>>=)

  /// Add & like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(&=)

  /// Add ^ like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(^=)

  /// Add | like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(|=)


  /** Since the boost::operator work on the small_array, add an implicit
      conversion to produce the expected type */
  operator FinalType () {
    return *static_cast<FinalType *>(this);
  }

};


/** A small array of 1, 2 or 3 elements with the implicit constructors */
template <typename BasicType, typename FinalType, std::size_t Dims>
struct small_array_123 : small_array<BasicType, FinalType, Dims> {
  static_assert(1 <= Dims && Dims <= 3,
                "Dimensions are between 1 and 3");
};


/** Use some specializations so that some function overloads can be
    determined according to some implicit constructors and to have an
    implicit conversion from/to BasicType (such as an int typically) if
    dims = 1
*/
template <typename BasicType, typename FinalType>
struct small_array_123<BasicType, FinalType, 1>
  : public small_array<BasicType, FinalType, 1> {
  /// A 1-D constructor to have implicit conversion from from 1 integer
  /// and automatic inference of the dimensionality
  small_array_123(BasicType x) {
    (*this)[0] = x;
  }


  /// Keep other constructors
  small_array_123() = default;

  using small_array<BasicType, FinalType, 1>::small_array;

  /** Conversion so that an for example an id<1> can basically be used
      like an integer */
  operator BasicType() const {
    return (*this)[0];
  }
};


template <typename BasicType, typename FinalType>
struct small_array_123<BasicType, FinalType, 2>
  : public small_array<BasicType, FinalType, 2> {
  /// A 2-D constructor to have implicit conversion from from 2 integers
  /// and automatic inference of the dimensionality
  small_array_123(BasicType x, BasicType y) {
    (*this)[0] = x;
    (*this)[1] = y;
  }


  /** Broadcasting constructor initializing all the elements with the
      same value

      \todo Add to the specification of the range, id...
  */
  explicit small_array_123(BasicType e) : small_array_123 { e, e } { }


  /// Keep other constructors
  small_array_123() = default;

  using small_array<BasicType, FinalType, 2>::small_array;
};

```

```

template <typename BasicType, typename FinalType>
struct small_array_123<BasicType, FinalType, 3>
  : public small_array<BasicType, FinalType, 3> {
  /// A 3-D constructor to have implicit conversion from from 3 integers
  /// and automatic inference of the dimensionality
  small_array_123(BasicType x, BasicType y, BasicType z) {
    (*this)[0] = x;
    (*this)[1] = y;
    (*this)[2] = z;
  }


  /** Broadcasting constructor initializing all the elements with the
      same value

      \todo Add to the specification of the range, id...
  */
  explicit small_array_123(BasicType e) : small_array_123 { e, e, e } { }


  /// Keep other constructors
  small_array_123() = default;

  using small_array<BasicType, FinalType, 3>::small_array;
};

/// @} End the helpers Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_SMALL_ARRAY_HPP
```

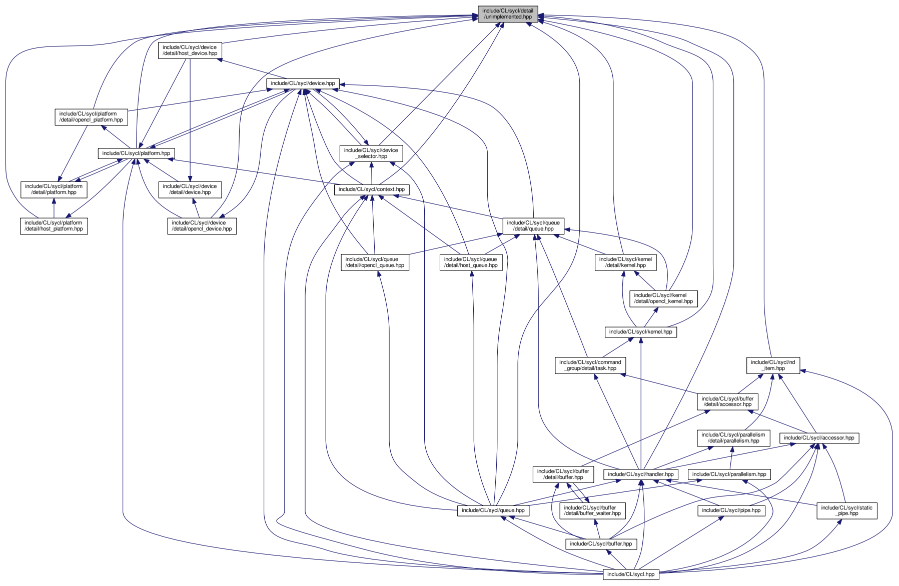
11.45 include/CL/sycl/detail/unimplemented.hpp File Reference

`#include <iostream>`

Include dependency graph for unimplemented.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/detail/unimplemented.hpp
include/CL/sycl/device/detail/host_device.hpp
include/CL/sycl/device.hpp
include/CL/sycl/platform/detail/opencl_platform.hpp
include/CL/sycl/platform.hpp
include/CL/sycl/device_selector.hpp
include/CL/sycl/platform/detail/platform.hpp
include/CL/sycl/device/detail/device.hpp
include/CL/sycl/context.hpp
include/CL/sycl/platform/detail/host_platform.hpp
include/CL/sycl/device/detail/opencl_device.hpp
include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/nd_item.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/parallelism/detail/parallelism.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/parallelism.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- void cl::sycl::detail::unimplemented ()

  Display an "unimplemented" message.

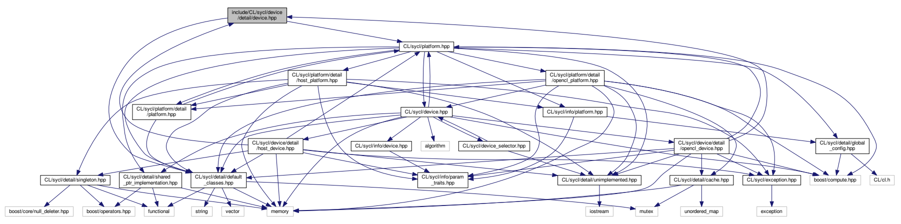
11.46 unimplemented.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_UNIMPLEMENTED_HPP
#define TRISYCL_SYCL_DETAIL_UNIMPLEMENTED_HPP

/** \file Deal with unimplemented features
    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <iostream>

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup helpers Some helpers for the implementation
    @{
*/

/** Display an "unimplemented" message

    Can be changed to call assert(0) or whatever.
*/
inline void unimplemented() {
  std::cerr << "Error: using a non implemented feature!!!" << std::endl
            << "Please contribute to the open source implementation. :-)"
            << std::endl;
}

/// @} End the helpers Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_UNIMPLEMENTED_HPP
```

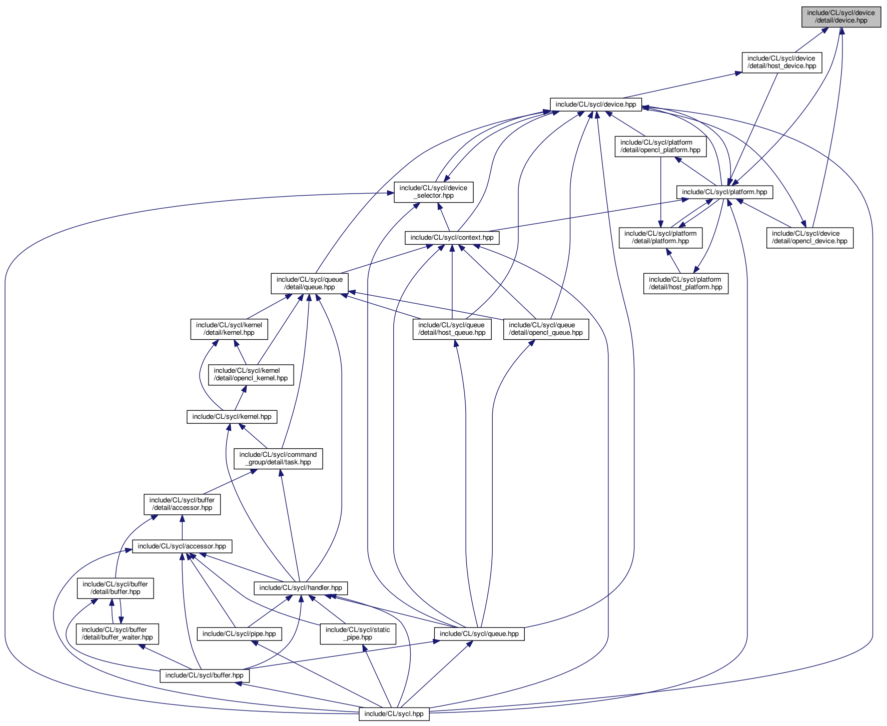
11.47 include/CL/sycl/device/detail/device.hpp File Reference

```
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for device.hpp:


include/CL/sycl/device/detail/device.hpp
CL/sycl/platform.hpp
CL/sycl/platform/detail/host_platform.hpp
CL/sycl/platform/detail/opencl_platform.hpp
CL/sycl/platform/detail/platform.hpp
CL/sycl/device.hpp
CL/sycl/info/platform.hpp
CL/sycl/device/detail/host_device.hpp
CL/sycl/info/device.hpp
algorithm
CL/sycl/device_selector.hpp
CL/sycl/device/detail/opencl_device.hpp
CL/sycl/detail/global_config.hpp
CL/sycl/detail/singleton.hpp
CL/sycl/detail/shared_ptr_implementation.hpp
CL/sycl/detail/default_classes.hpp
CL/sycl/info/param_traits.hpp
CL/sycl/detail/unimplemented.hpp
CL/sycl/detail/cache.hpp
CL/sycl/exception.hpp
boost/compute.hpp
CL/cl.h
boost/core/null_deleter.hpp
boost/operators.hpp
functional
string
vector
memory
iostream
mutex
unordered_map
exception

This graph shows which files directly or indirectly include this file:


include/CL/sycl/device/detail/device.hpp
include/CL/sycl/device/detail/host_device.hpp
include/CL/sycl/device.hpp
include/CL/sycl/platform/detail/opencl_platform.hpp
include/CL/sycl/device_selector.hpp
include/CL/sycl/platform.hpp
include/CL/sycl/context.hpp
include/CL/sycl/platform/detail/platform.hpp
include/CL/sycl/device/detail/opencl_device.hpp
include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/platform/detail/host_platform.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::detail::device

  An abstract class representing various models of SYCL devices. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.48 device.hpp

```
#ifndef TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_HPP
#define TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_HPP

/** \file The OpenCL SYCL abstract device

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/platform.hpp"

```

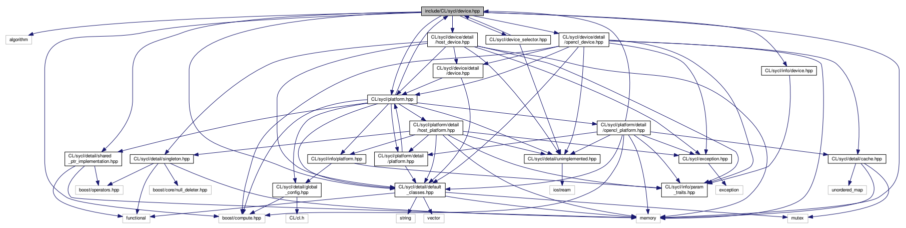
```
namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/// An abstract class representing various models of SYCL devices
class device {

public:

#ifdef TRISYCL_OPENCL
  /// Return the cl_device_id of the underlying OpenCL platform
  virtual cl_device_id get() const = 0;
#endif


  /// Return true if the device is a SYCL host device
  virtual bool is_host() const = 0;


  /// Return true if the device is an OpenCL CPU device
  virtual bool is_cpu() const = 0;


  /// Return true if the device is an OpenCL GPU device
  virtual bool is_gpu() const = 0;


  /// Return true if the device is an OpenCL accelerator device
  virtual bool is_accelerator() const = 0;


  /// Return the platform of device
  virtual cl::sycl::platform get_platform() const = 0;


  /// Query the device for OpenCL info::device info
  /** \todo virtual cannot be templated
  template <typename T>
  virtual T get_info(info::device param) const = 0;
  */


  /// Specify whether a specific extension is supported on the device.
  virtual bool has_extension(const string_class &extension) const = 0;


  // Virtual to call the real destructor
  virtual ~device() {}

};

/// @} to end the execution Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_HPP
```

11.49 include/CL/sycl/device.hpp File Reference

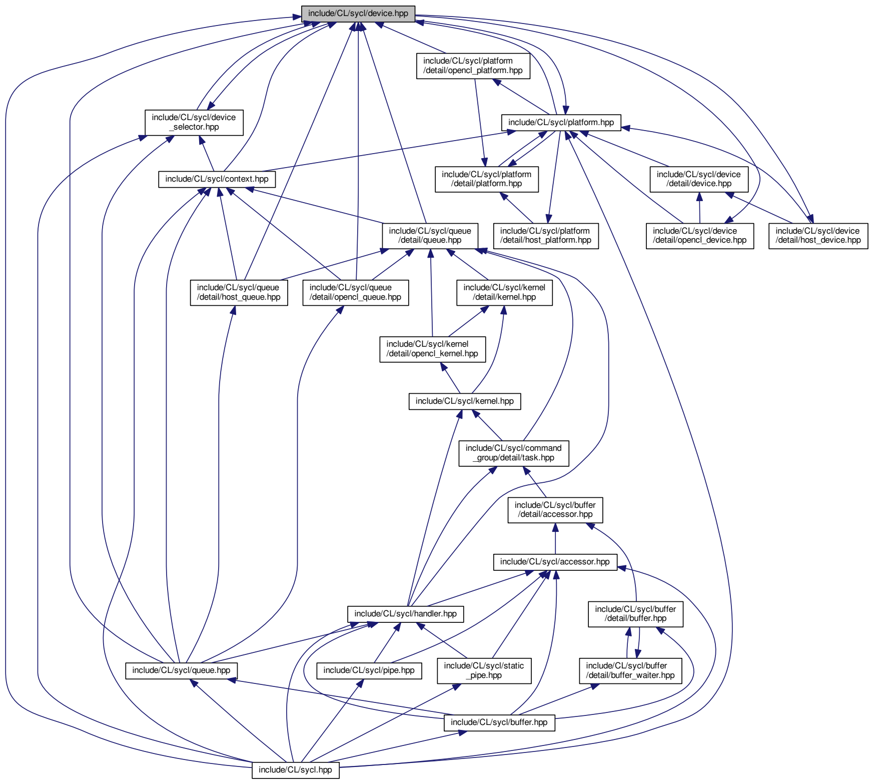
```
#include <algorithm>
```

```
#include <memory>
#include <boost/compute.hpp>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/device/detail/host_device.hpp"
#include "CL/sycl/device/detail/opencl_device.hpp"
#include "CL/sycl/info/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for device.hpp:


include/CL/sycl/device.hpp
algorithm
CL/sycl/device/detail/host_device.hpp
CL/sycl/device_selector.hpp
CL/sycl/device/detail/opencl_device.hpp
CL/sycl/info/device.hpp
CL/sycl/device/detail/device.hpp
CL/sycl/platform.hpp
CL/sycl/platform/detail/host_platform.hpp
CL/sycl/platform/detail/opencl_platform.hpp
CL/sycl/detail/shared_ptr_implementation.hpp
CL/sycl/detail/singleton.hpp
CL/sycl/info/platform.hpp
CL/sycl/platform/detail/platform.hpp
CL/sycl/detail/unimplemented.hpp
CL/sycl/exception.hpp
CL/sycl/detail/cache.hpp
boost/operators.hpp
boost/core/null_deleter.hpp
CL/sycl/detail/global_config.hpp
CL/sycl/detail/default_classes.hpp
iostream
CL/sycl/info/param_traits.hpp
exception
unordered_map
functional
boost/compute.hpp
CL/cl.h
string
vector
memory
mutex

This graph shows which files directly or indirectly include this file:


include/CL/sycl/device.hpp
include/CL/sycl/platform/detail/opencl_platform.hpp
include/CL/sycl/device_selector.hpp
include/CL/sycl/platform.hpp
include/CL/sycl/context.hpp
include/CL/sycl/platform/detail/platform.hpp
include/CL/sycl/device/detail/device.hpp
include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/platform/detail/host_platform.hpp
include/CL/sycl/device/detail/opencl_device.hpp
include/CL/sycl/device/detail/host_device.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::device

  SYCL device. More...
- struct std::hash< cl::sycl::device >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- std

11.50 device.hpp

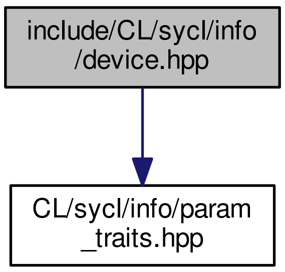
```
#ifndef TRISYCL_SYCL_DEVICE_HPP
#define TRISYCL_SYCL_DEVICE_HPP

/** \file The OpenCL SYCL device

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <algorithm>
#include <memory>

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/device/detail/host_device.hpp"
#ifdef TRISYCL_OPENCL
#include "CL/sycl/device/detail/opencl_device.hpp"
#endif
#include "CL/sycl/info/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/platform.hpp"

namespace cl {
namespace sycl {

class device_selector;
class platform;

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/// SYCL device
class device
  /* Use the underlying device implementation that can be shared in the
     SYCL model */
  : public detail::shared_ptr_implementation<device, detail::device> {

  // The type encapsulating the implementation
  using implementation_t =
    detail::shared_ptr_implementation<device, detail::device>
;

public:

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

  /// The default constructor uses the SYCL host device
  device() : implementation_t {
detail::host_device::instance() } {}


#ifdef TRISYCL_OPENCL
  /** Construct a device class instance using cl_device_id of the
      OpenCL device

      Return synchronous errors via the SYCL exception class.

      Retain a reference to the OpenCL device and if this device was
      an OpenCL subdevice the device should be released by the caller
      when it is no longer needed.
  */
  device(cl_device_id device_id)
    : device { boost::compute::device { device_id } } {}


  /** Construct a device class instance using a boost::compute::device

      This is a triSYCL extension for boost::compute interoperation.

      Return synchronous errors via the SYCL exception class.
  */
  device(const boost::compute::device &d)
    : implementation_t { detail::opencl_device::instance(d)
} {}
#endif


  /** Construct a device class instance using the device selector
      provided

      Return errors via C++ exception class.

      \todo Make it non-explicit in the specification?
  */
  explicit device(const device_selector &ds) {
    auto devices = device::get_devices();
    if (devices.empty())
      // \todo Put a SYCL exception
      throw std::domain_error("No device at all! Internal error...");

    /* Find the device with the best score according to the given
       device_selector */
    auto max = std::max_element(devices.cbegin(), devices.cend(),
                                [&] (const device &d1, const device &d2) {
                                  return ds(d1) < ds(d2);
                                });
    if (ds(*max) < 0)
      // \todo Put a SYCL exception
      throw std::domain_error("No device selected because no positive "
                              "device_selector score found");

    // Create the current device as a shared copy of the selected one
    implementation = max->implementation;
  }


#ifdef TRISYCL_OPENCL
  /** Return the cl_device_id of the underlying OpenCL platform

      Return synchronous errors via the SYCL exception class.

      Retain a reference to the returned cl_device_id object. Caller
      should release it when finished.

      In the case where this is the SYCL host device it will throw an
      exception.
  */
  cl_device_id get() const {
    return implementation->get();
  }
#endif


  /// Return true if the device is the SYCL host device
  bool is_host() const {
    return implementation->is_host();
  }


  /// Return true if the device is an OpenCL CPU device
  bool is_cpu() const {
    return implementation->is_cpu();
  }


  /// Return true if the device is an OpenCL GPU device
  bool is_gpu() const {
    return implementation->is_gpu();
  }


  /// Return true if the device is an OpenCL accelerator device
  bool is_accelerator() const {
    return implementation->is_accelerator();
  }


  /** Return the device_type of a device

      \todo Present in Boost.Compute, to be added to the specification
  */
  info::device_type type() const {
    if (is_host())
      return info::device_type::host;
    else if (is_cpu())
      return info::device_type::cpu;
    else if (is_gpu())
      return info::device_type::gpu;
    else if (is_accelerator())
      return info::device_type::accelerator;
    else
      // \todo Put a SYCL exception
      throw std::domain_error("Unknown cl::sycl::info::device_type");
  }


  /** Return the platform of device

      Return synchronous errors via the SYCL exception class.
  */
  platform get_platform() const {
    return implementation->get_platform();
  }


  /** Return a list of all available devices

      Return synchronous errors via SYCL exception classes.
  */
  static vector_class<device>
  get_devices(info::device_type device_type =
info::device_type::all)
    __attribute__((weak));


  /** Query the device for OpenCL info::device info

      Return synchronous errors via the SYCL exception class.

      \todo
  */
  template <typename T>
  T get_info(info::device param) const {
    //return implementation->get_info<Param>(param);
  }


  /** Query the device for OpenCL info::device info

      Return synchronous errors via the SYCL exception class.

      \todo
  */
  template <info::device Param>
  auto get_info() const {
    // Forward to the version where the info parameter is not a template
    //return get_info<typename info::param_traits_t<info::device, Param>>(Param);
  }


  /// Test if a specific extension is supported on the device
  bool has_extension(const string_class &extension) const {
    return implementation->has_extension(extension);
  }


#ifdef XYZTRISYCL_OPENCL
  /** Partition the device into sub devices based upon the properties
      provided

      Return synchronous errors via SYCL exception classes.

      \todo
  */
  vector_class<device>
  create_sub_devices(info::device_partition_type partition_type,
                     info::device_partition_property partition_property,
                     info::device_affinity_domain affinity_domain) const {
    return implementation->create_sub_devices(partition_type,
                                              partition_property,
                                              affinity_domain);
  }
#endif

```

```
};

/// @} to end the Doxygen group

}
}


/* Inject a custom specialization of std::hash to have the buffer
   usable into an unordered associative container

   \todo Add this to the spec
*/
namespace std {

template <> struct hash<cl::sycl::device> {

  auto operator()(const cl::sycl::device &d) const {
    // Forward the hashing to the implementation
    return d.hash();
  }

};

}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_HPP
```

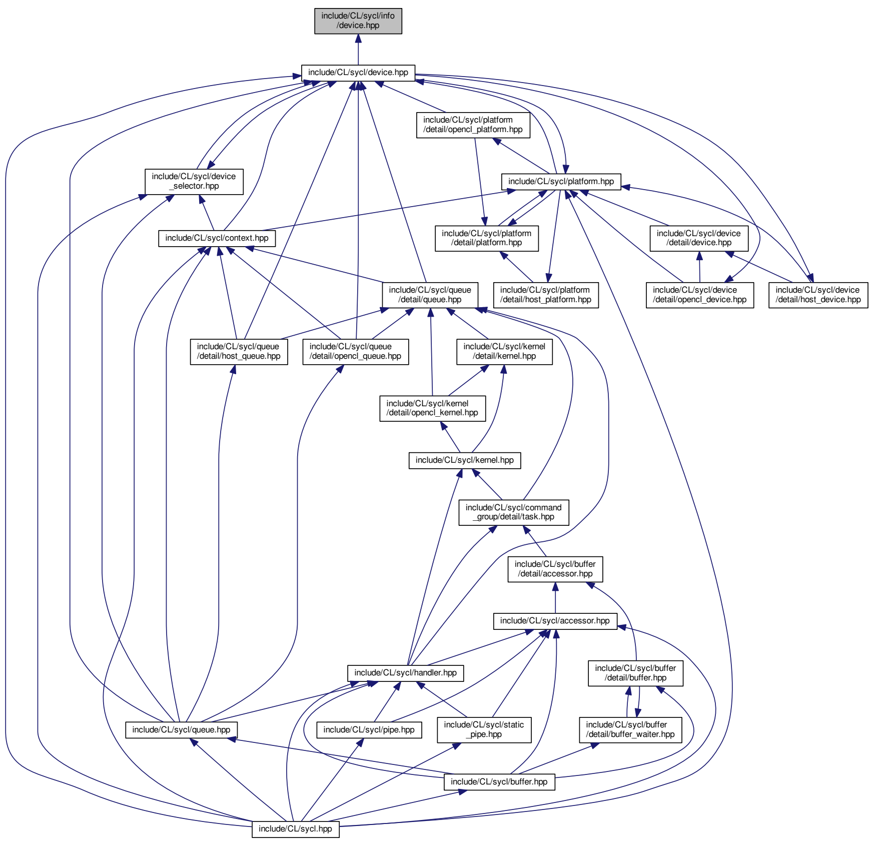
11.51 include/CL/sycl/info/device.hpp File Reference

`#include "CL/sycl/info/param_traits.hpp"`

Include dependency graph for device.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/info/device.hpp
include/CL/sycl/device.hpp
include/CL/sycl/platform/detail/opencl_platform.hpp
include/CL/sycl/device_selector.hpp
include/CL/sycl/platform.hpp
include/CL/sycl/context.hpp
include/CL/sycl/platform/detail/platform.hpp
include/CL/sycl/device/detail/device.hpp
include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/platform/detail/host_platform.hpp
include/CL/sycl/device/detail/opencl_device.hpp
include/CL/sycl/device/detail/host_device.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info

Typedefs

- using cl::sycl::info::device_fp_config = unsigned int
- using cl::sycl::info::device_exec_capabilities = unsigned int
- using cl::sycl::info::device_queue_properties = unsigned int

Enumerations

- enum cl::sycl::info::device_type : unsigned int {
  cl::sycl::info::device_type::cpu, cl::sycl::info::device_type::gpu, cl::sycl::info::device_type::accelerator, cl↩
  ::sycl::info::device_type::custom,
  cl::sycl::info::device_type::defaults, cl::sycl::info::device_type::host, cl::sycl::info::device_type::opencl, cl↩
  ::sycl::info::device_type::all }

  Type of devices.

- enum cl::sycl::info::device : int {
  cl::sycl::info::device::device_type, cl::sycl::info::device::vendor_id, cl::sycl::info::device::max_compute_units, cl::sycl::info::device::max_work_item_dimensions,
  cl::sycl::info::device::max_work_item_sizes, cl::sycl::info::device::max_work_group_size, cl::sycl::info::device::preferred_vector_width_char, cl::sycl::info::device::preferred_vector_width_short,
  cl::sycl::info::device::preferred_vector_width_int, cl::sycl::info::device::preferred_vector_width_long_long, cl::sycl::info::device::preferred_vector_width_float, cl::sycl::info::device::preferred_vector_width_double,
  cl::sycl::info::device::preferred_vector_width_half, cl::sycl::info::device::native_vector_witdth_char, cl::sycl::info::device::native_vector_witdth_short, cl::sycl::info::device::native_vector_witdth_int,
  cl::sycl::info::device::native_vector_witdth_long_long, cl::sycl::info::device::native_vector_witdth_float, cl::sycl::info::device::native_vector_witdth_double, cl::sycl::info::device::native_vector_witdth_half,
  cl::sycl::info::device::max_clock_frequency, cl::sycl::info::device::address_bits, cl::sycl::info::device::max_mem_alloc_size, cl::sycl::info::device::image_support,
  cl::sycl::info::device::max_read_image_args, cl::sycl::info::device::max_write_image_args, cl::sycl::info::device::image2d_max_height, cl::sycl::info::device::image2d_max_width,
  cl::sycl::info::device::image3d_max_height, cl::sycl::info::device::image3d_max_widht, cl::sycl::info::device::image3d_mas_depth, cl::sycl::info::device::image_max_buffer_size,
  cl::sycl::info::device::image_max_array_size, cl::sycl::info::device::max_samplers, cl::sycl::info::device::max_parameter_size, cl::sycl::info::device::mem_base_addr_align,
  cl::sycl::info::device::single_fp_config, cl::sycl::info::device::double_fp_config, cl::sycl::info::device::global_mem_cache_type, cl::sycl::info::device::global_mem_cache_line_size,
  cl::sycl::info::device::global_mem_cache_size, cl::sycl::info::device::global_mem_size, cl::sycl::info::device::max_constant_buffer_size, cl::sycl::info::device::max_constant_args,
  cl::sycl::info::device::local_mem_type, cl::sycl::info::device::local_mem_size, cl::sycl::info::device::error_correction_support, cl::sycl::info::device::host_unified_memory,
  cl::sycl::info::device::profiling_timer_resolution, cl::sycl::info::device::endian_little, cl::sycl::info::device::is_available, cl::sycl::info::device::is_compiler_available,
  cl::sycl::info::device::is_linker_available, cl::sycl::info::device::execution_capabilities, cl::sycl::info::device::queue_properties, cl::sycl::info::device::built_in_kernels,
  cl::sycl::info::device::platform, cl::sycl::info::device::name, cl::sycl::info::device::vendor, cl::sycl::info::device::driver_version,
  cl::sycl::info::device::profile, cl::sycl::info::device::device_version, cl::sycl::info::device::opencl_version, cl::sycl::info::device::extensions,
  cl::sycl::info::device::printf_buffer_size, cl::sycl::info::device::preferred_interop_user_sync, cl::sycl::info::device::parent_device, cl::sycl::info::device::partition_max_sub_devices,
  cl::sycl::info::device::partition_properties, cl::sycl::info::device::partition_affinity_domain, cl::sycl::info::device::partition_type, cl::sycl::info::device::reference_count }

  Device information descriptors.
- enum cl::sycl::info::device_partition_property : int {
  cl::sycl::info::device_partition_property::unsupported, cl::sycl::info::device_partition_property::partition_equally, cl::sycl::info::device_partition_property::partition_by_counts, cl::sycl::info::device_partition_property::partition_by_affinity_domain,
  cl::sycl::info::device_partition_property::partition_affinity_domain_next_partitionable }
- enum cl::sycl::info::device_affinity_domain : int {
  cl::sycl::info::device_affinity_domain::unsupported, cl::sycl::info::device_affinity_domain::numa, cl::sycl::info::device_affinity_domain::L4_cache, cl::sycl::info::device_affinity_domain::L3_cache,
  cl::sycl::info::device_affinity_domain::L2_cache, cl::sycl::info::device_affinity_domain::next_partitionable }
- enum cl::sycl::info::device_partition_type : int {
  cl::sycl::info::device_partition_type::no_partition, cl::sycl::info::device_partition_type::numa, cl::sycl::info::device_partition_type::L4_cache, cl::sycl::info::device_partition_type::L3_cache,
  cl::sycl::info::device_partition_type::L2_cache, cl::sycl::info::device_partition_type::L1_cache }
- enum cl::sycl::info::local_mem_type : int { cl::sycl::info::local_mem_type::none, cl::sycl::info::local_mem_type::local, cl::sycl::info::local_mem_type::global }
- enum cl::sycl::info::fp_config : int {
  cl::sycl::info::fp_config::denorm, cl::sycl::info::fp_config::inf_nan, cl::sycl::info::fp_config::round_to_nearest, cl::sycl::info::fp_config::round_to_zero,
  cl::sycl::info::fp_config::round_to_inf, cl::sycl::info::fp_config::fma, cl::sycl::info::fp_config::correctly_rounded_divide_sqrt, cl::sycl::info::fp_config::soft_float }

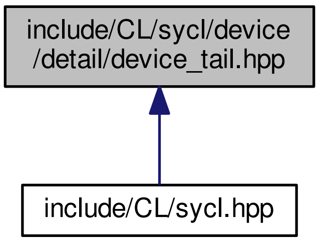
- enum cl::sycl::info::global_mem_cache_type : int { cl::sycl::info::global_mem_cache_type::none, cl::sycl↩ ::info::global_mem_cache_type::read_only, cl::sycl::info::global_mem_cache_type::write_only }
- enum cl::sycl::info::device_execution_capabilities : unsigned int { cl::sycl::info::device_execution_↩ capabilities::exec_kernel, cl::sycl::info::device_execution_capabilities::exec_native_kernel }

11.52 device.hpp

```
#ifndef TRISYCL_SYCL_INFO_DEVICE_HPP
#define TRISYCL_SYCL_INFO_DEVICE_HPP

/** \file The OpenCL SYCL device information parameters

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/info/param_traits.hpp"

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

namespace info {

/** Type of devices

    To be used either to define a device type or to select more
    broadly a kind of device

    \todo To be moved in the specification from platform to device

    \todo Add opencl to the specification

    \todo there is no accelerator_selector and custom_accelerator
*/
enum class device_type : unsigned int {
  cpu,
  gpu,
  accelerator,
  custom,
  defaults,
  host,
  opencl,
  all
};


/** Device information descriptors

    From specs/latex/headers/deviceInfo.h in the specification

    \todo Should be unsigned int?
*/
enum class device : int {
  device_type,
  vendor_id,
  max_compute_units,
  max_work_item_dimensions,
  max_work_item_sizes,
  max_work_group_size,
  preferred_vector_width_char,
  preferred_vector_width_short,
  preferred_vector_width_int,
  preferred_vector_width_long_long,
  preferred_vector_width_float,
  preferred_vector_width_double,
  preferred_vector_width_half,
  native_vector_witdth_char,
  native_vector_witdth_short,
  native_vector_witdth_int,
  native_vector_witdth_long_long,
  native_vector_witdth_float,
  native_vector_witdth_double,
  native_vector_witdth_half,
  max_clock_frequency,
  address_bits,
  max_mem_alloc_size,
  image_support,
  max_read_image_args,
  max_write_image_args,
  image2d_max_height,
  image2d_max_width,
  image3d_max_height,
  image3d_max_widht,
  image3d_mas_depth,
  image_max_buffer_size,
  image_max_array_size,
  max_samplers,
  max_parameter_size,
  mem_base_addr_align,
  single_fp_config,
  double_fp_config,
  global_mem_cache_type,
  global_mem_cache_line_size,
  global_mem_cache_size,
  global_mem_size,
  max_constant_buffer_size,
  max_constant_args,
  local_mem_type,
  local_mem_size,
  error_correction_support,
  host_unified_memory,
  profiling_timer_resolution,
  endian_little,
  is_available,
  is_compiler_available,
  is_linker_available,
  execution_capabilities,
  queue_properties,
  built_in_kernels,
  platform,
  name,
  vendor,
  driver_version,
  profile,
  device_version,
  opencl_version,
  extensions,
  printf_buffer_size,
  preferred_interop_user_sync,
  parent_device,
  partition_max_sub_devices,
  partition_properties,
  partition_affinity_domain,
  partition_type,
  reference_count
};

enum class device_partition_property : int {
  unsupported,
  partition_equally,
  partition_by_counts,
  partition_by_affinity_domain,
  partition_affinity_domain_next_partitionable
};

enum class device_affinity_domain : int {
  unsupported,
  numa,
  L4_cache,
  L3_cache,
  L2_cache,
  next_partitionable
};

enum class device_partition_type : int {
  no_partition,
  numa,
  L4_cache,
  L3_cache,
  L2_cache,
  L1_cache
};

enum class local_mem_type : int {
  none,
  local,
  global
};

enum class fp_config : int {
  denorm,
  inf_nan,
  round_to_nearest,
  round_to_zero,
  round_to_inf,
  fma,
  correctly_rounded_divide_sqrt,
  soft_float
};

enum class global_mem_cache_type : int {
  none,
  read_only,
  write_only
};

enum class device_execution_capabilities : unsigned int {
  exec_kernel,
  exec_native_kernel
};


using device_fp_config = unsigned int;
using device_exec_capabilities = unsigned int;
using device_queue_properties = unsigned int;


/** Query the return type for get_info() on context stuff

    \todo To be implemented, return always void.
*/
TRISYCL_INFO_PARAM_TRAITS_ANY_T(info::device, void)

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_INFO_DEVICE_HPP
```

11.53 include/CL/sycl/device/detail/device_tail.hpp File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- cl

    The vector type to be used as SYCL vector.
- cl::sycl

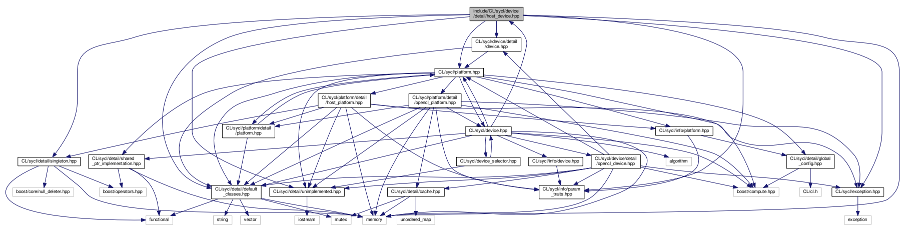
11.54 device_tail.hpp

```
#ifndef TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_TAIL_HPP
#define TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_TAIL_HPP

/** \file The ending part of of OpenCL SYCL device

    This is here to break a dependence between device and device_selector

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/** Return a list of all available devices

    Return synchronous errors via SYCL exception classes.
*/
vector_class<device>
device::get_devices(info::device_type device_type) {
  // Start with the default device
  vector_class<device> devices = { {} };

#ifdef TRISYCL_OPENCL
  // Then add all the OpenCL devices
  for (const auto &d : boost::compute::system::devices())
    devices.emplace_back(d);
#endif

  // The selected devices
  vector_class<device> sd;
  device_type_selector s { device_type };

  // Return the devices with the good criterion according to the selector
  std::copy_if(devices.begin(), devices.end(), std::back_inserter(sd),
               [&](const device &e ) { return s(e) >= 0; });
  return sd;
}

/// @} to end the Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_TAIL_HPP
```

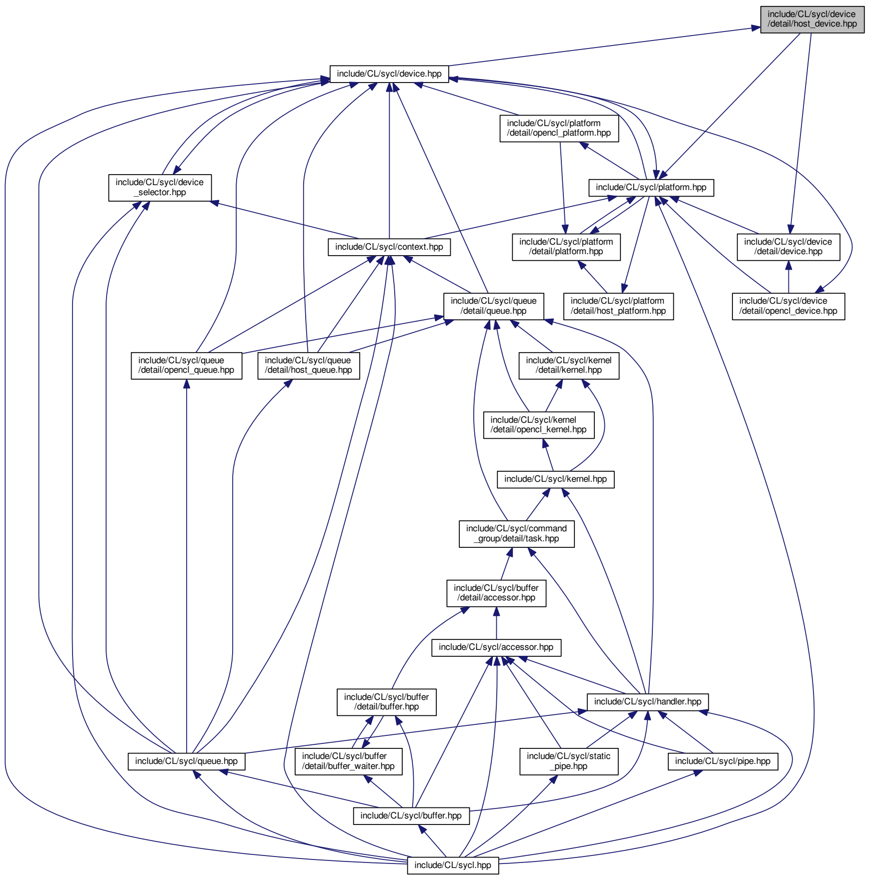
11.55 include/CL/sycl/device/detail/host_device.hpp File Reference

```
#include <memory>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/singleton.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device/detail/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for host_device.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class cl::sycl::detail::host_device

  SYCL host device.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.56 host_device.hpp

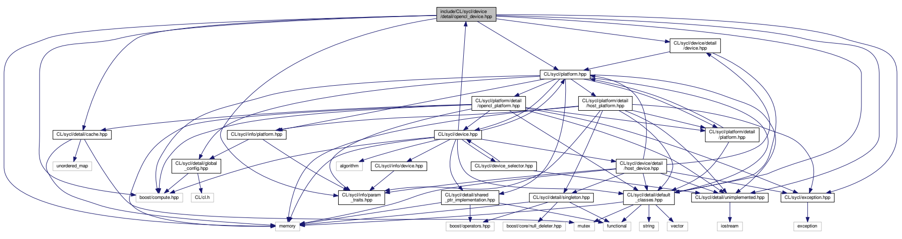
```
#ifndef TRISYCL_SYCL_DEVICE_DETAIL_HOST_DEVICE_HPP
#define TRISYCL_SYCL_DEVICE_DETAIL_HOST_DEVICE_HPP

/** \file The OpenCL SYCL host device implementation

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <memory>

#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/detail/singleton.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device/detail/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** SYCL host device

    \todo The implementation is quite minimal for now. :-)
*/
class host_device : public detail::device,
                    public detail::singleton<host_device> {

public:

#ifdef TRISYCL_OPENCL
  /** Return the cl_device_id of the underlying OpenCL platform

      This throws an error since there is no OpenCL device associated
      to the host device.
  */
  cl_device_id get() const override {
    throw non_cl_error("The host device has no OpenCL device");
  }
#endif


  /// Return true since the device is a SYCL host device
  bool is_host() const override {
    return true;
  }


  /// Return false since the host device is not an OpenCL CPU device
  bool is_cpu() const override {
    return false;
  }


  /// Return false since the host device is not an OpenCL GPU device
  bool is_gpu() const override {
    return false;
  }


  /// Return false since the host device is not an OpenCL accelerator device
  bool is_accelerator() const override {
    return false;
  }


  /** Return the platform of device

      Return synchronous errors via the SYCL exception class.

      \todo To be implemented
  */
  cl::sycl::platform get_platform() const override {
    detail::unimplemented();
    return {};
  }

#if 0
  /** Query the device for OpenCL info::device info
```

```

      Return synchronous errors via the SYCL exception class.

      \todo To be implemented
  */
  template <info::device Param>
  typename info::param_traits<info::device, Param>::type
  get_info() const override {
    detail::unimplemented();
    return {};
  }
#endif

  /** Specify whether a specific extension is supported on the device

      \todo To be implemented
  */
  bool has_extension(const string_class &extension) const override {
    detail::unimplemented();
    return {};
  }


};

/// @} to end the execution Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_DETAIL_HOST_DEVICE_HPP
```

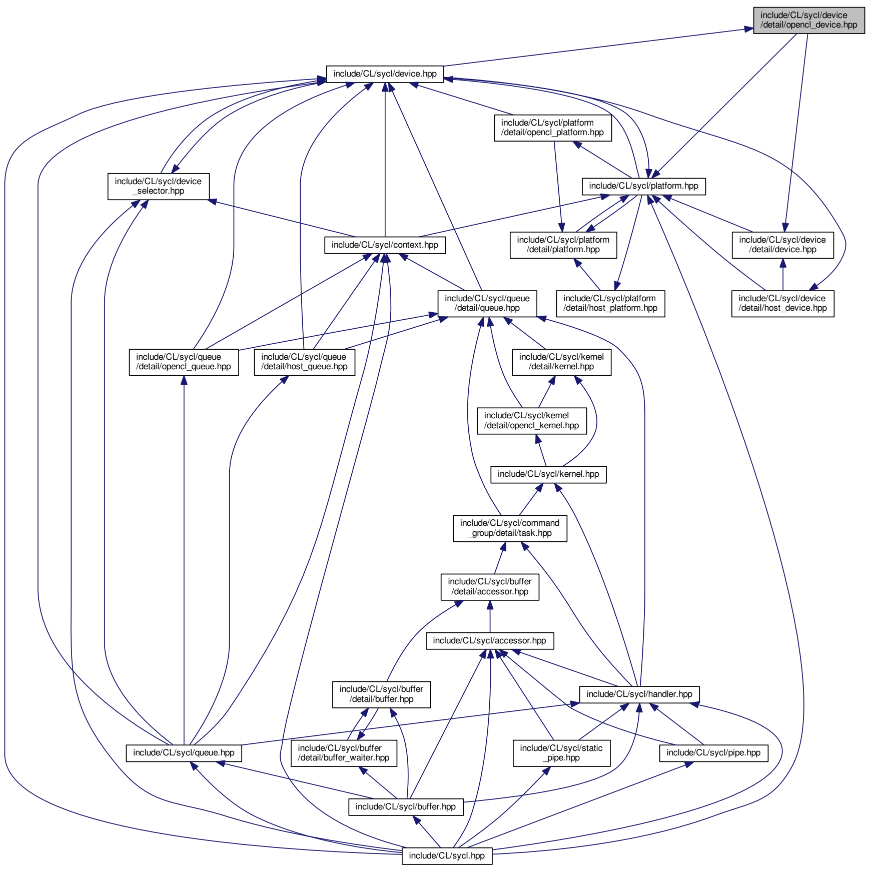
11.57 include/CL/sycl/device/detail/opencl_device.hpp File Reference

```
#include <memory>
#include <boost/compute.hpp>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device/detail/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for opencl_device.hpp:


include/CL/sycl/device/detail/opencl_device.hpp
CL/sycl/device/detail/device.hpp
CL/sycl/platform.hpp
CL/sycl/platform/detail/opencl_platform.hpp
CL/sycl/platform/detail/host_platform.hpp
CL/sycl/detail/cache.hpp
CL/sycl/info/platform.hpp
CL/sycl/device.hpp
CL/sycl/platform/detail/platform.hpp
unordered_map
CL/sycl/detail/global_config.hpp
algorithm
CL/sycl/info/device.hpp
CL/sycl/device_selector.hpp
CL/sycl/device/detail/host_device.hpp
boost/compute.hpp
CL/cl.h
CL/sycl/info/param_traits.hpp
CL/sycl/detail/shared_ptr_implementation.hpp
CL/sycl/detail/singleton.hpp
CL/sycl/detail/default_classes.hpp
CL/sycl/detail/unimplemented.hpp
CL/sycl/exception.hpp
memory
boost/operators.hpp
boost/core/null_deleter.hpp
mutex
functional
string
vector
iostream
exception

This graph shows which files directly or indirectly include this file:



Classes

- class cl::sycl::detail::opencl_device

  SYCL OpenCL device.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Variables

- detail::cache< cl_device_id, detail::opencl_device > opencl_device::cache cl::sycl::detail::__attribute__↩ ((weak))

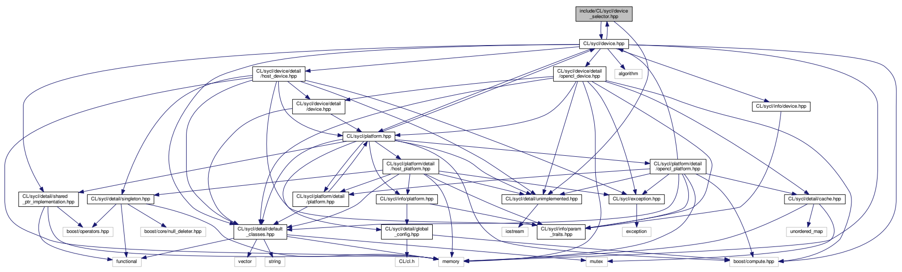
11.58 opencl_device.hpp

```
#ifndef TRISYCL_SYCL_DEVICE_DETAIL_OPENCL_DEVICE_HPP
#define TRISYCL_SYCL_DEVICE_DETAIL_OPENCL_DEVICE_HPP

/** \file The SYCL OpenCL device implementation

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <memory>

#include <boost/compute.hpp>

#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device/detail/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform.hpp"

namespace cl {
namespace sycl {
namespace detail {

/// SYCL OpenCL device
class opencl_device : public detail::device {

  /// Use the Boost Compute abstraction of the OpenCL device
  boost::compute::device d;

  /** A cache to always return the same alive device for a given
      OpenCL device

      C++11 guaranties the static construction is thread-safe
  */
  static detail::cache<cl_device_id, detail::opencl_device>
 cache;

public:

  /// Return the cl_device_id of the underlying OpenCL device
  cl_device_id get() const override {
    return d.id();
  }


  /// Return false since an OpenCL device is not the SYCL host device
  bool is_host() const override {
    return false;
  }


  /// Test if the OpenCL is a CPU device
  bool is_cpu() const override {
    return d.type() == boost::compute::device::cpu;
  }


  /// Test if the OpenCL is a GPU device
  bool is_gpu() const override {
    return d.type() == boost::compute::device::gpu;
  }


  /// Test if the OpenCL is an accelerator device
  bool is_accelerator() const override {
    return d.type() == boost::compute::device::accelerator;
  }


  /** Return the platform of device

      Return synchronous errors via the SYCL exception class.

      \todo To be implemented
  */
  cl::sycl::platform get_platform() const override {
    detail::unimplemented();
    return {};
  }

#if 0
  /** Query the device for OpenCL info::device info

      Return synchronous errors via the SYCL exception class.

      \todo To be implemented
  */
  template <info::device Param>
  typename info::param_traits<info::device, Param>::type
  get_info() const override {
    detail::unimplemented();
    return {};
  }
#endif

  /** Specify whether a specific extension is supported on the device.

      \todo To be implemented
  */
  bool has_extension(const string_class &extension) const override {
    detail::unimplemented();
    return {};
  }


  ///// Get a singleton instance of the opencl_device
  static std::shared_ptr<opencl_device>
  instance(const boost::compute::device &d) {
    return cache.get_or_register(d.id(),
                                 [&] { return new opencl_device { d }; });
  }

private:

  /// Only the instance factory can built it
  opencl_device(const boost::compute::device &d) : d { d } {}

public:

  /// Unregister from the cache on destruction
  ~opencl_device() override {
    cache.remove(d.id());
  }

};

/* Allocate the cache here but since this is a pure-header library,
   use a weak symbol so that only one remains when SYCL headers are
   used in different compilation units of a program
*/
detail::cache<cl_device_id, detail::opencl_device>
opencl_device::cache
  __attribute__((weak));

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_DETAIL_OPENCL_DEVICE_HPP
```

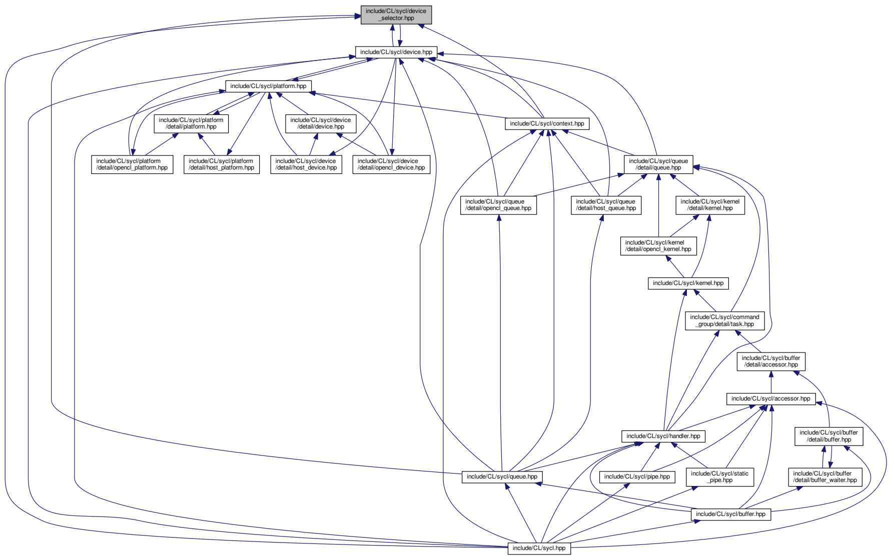
11.59 include/CL/sycl/device_selector.hpp File Reference

```
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
```

Include dependency graph for device_selector.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class cl::sycl::device_selector

  The SYCL heuristics to select a device. More...

Namespaces

- cl

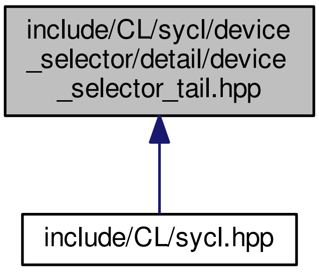
  The vector type to be used as SYCL vector.
- cl::sycl

11.60 device_selector.hpp

```
#ifndef TRISYCL_SYCL_DEVICE_SELECTOR_HPP
#define TRISYCL_SYCL_DEVICE_SELECTOR_HPP

/** \file The OpenCL SYCL device_selector

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/** The SYCL heuristics to select a device

    The device with the highest score is selected
*/
class device_selector {

public:

  /** Returns a selected device using the functor operator defined in
      sub-classes operator()(const device &dev)

      \todo Remove this from specification
  */
  void /* device */ select_device() const {
    //    return {};
  }


  /**  This pure virtual operator allows the customization of device
       selection.

       It defines the behavior of the device_selector functor called by
       the SYCL runtime on device selection. It returns a "score" for each
       device in the system and the highest rated device will be used
       by the SYCL runtime.
  */
  virtual int operator()(const device &dev) const = 0;


  /// Virtual destructor so the final destructor can be called if any
  virtual ~device_selector() {}

};

/// @} to end the execution Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_SELECTOR_HPP
```

11.61 include/CL/sycl/device_selector/detail/device_selector_tail.hpp File Reference

This graph shows which files directly or indirectly include this file:


include/CL/sycl/device_selector/detail/device_selector_tail.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::device_type_selector

  A device selector by device_type. More...
- class cl::sycl::device_typename_selector< DeviceType >

  Select a device by template device_type parameter. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Typedefs

- using cl::sycl::default_selector = device_typename_selector< info::device_type::defaults >

  Devices selected by heuristics of the system.
- using cl::sycl::gpu_selector = device_typename_selector< info::device_type::gpu >

  Select devices according to device type info::device::device_type::gpu from all the available OpenCL devices.
- using cl::sycl::cpu_selector = device_typename_selector< info::device_type::cpu >

  Select devices according to device type info::device::device_type::cpu from all the available devices and heuristics.
- using cl::sycl::host_selector = device_typename_selector< info::device_type::host >

  Selects the SYCL host CPU device that does not require an OpenCL runtime.

11.62 device_selector_tail.hpp

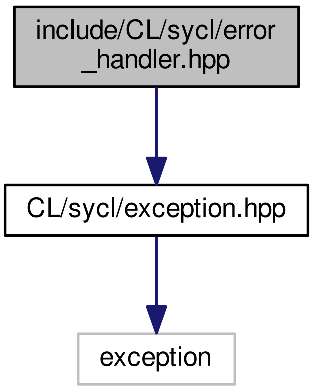
```
#ifndef TRISYCL_SYCL_DEVICE_SELECTOR_DETAIL_DEVICE_SELECTOR_TAIL_HPP
#define TRISYCL_SYCL_DEVICE_SELECTOR_DETAIL_DEVICE_SELECTOR_TAIL_HPP

/** \file The ending part of of the OpenCL SYCL device_selector

    This is here to break a dependence between device and device_selector

    \todo Implement lacking SYCL 2.2 selectors

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/


/** A device selector by device_type

    \todo To be added to the specification
*/
class device_type_selector : public device_selector {

private:

  /// The device_type to select
  info::device_type device_type;

  /** Cache the default device to select with the default device
      selector.

      This is the host device at construction time and remains as is
      if there is no openCL device */
  device default_device;

public:

  device_type_selector(info::device_type device_type)
    : device_type { device_type } {
    // The default device selection heuristic
    if (device_type == info::device_type::defaults) {
      auto devices = device::get_devices(
info::device_type::opencl);
      /* If there is an OpenCL device, pick the first one as the
         default device, other wise it is the host device */
      if (!devices.empty())
        default_device = devices[0];
    }
}

  // To select only the requested device_type
  int operator()(const device &dev) const override {
    if (device_type == info::device_type::all)
      // All devices fit all
      return 1;

    if (device_type == info::device_type::defaults)
      // Only select the default device
      return dev == default_device ? 1 : -1;

    if (device_type == info::device_type::opencl)
      // For now, any non host device is an OpenCL device
      return dev.is_host() ? -1 : 1;

    return dev.type() == device_type ? 1 : -1;
  }

};


/** Select a device by template device_type parameter

    \todo To be added to the specification
*/
template <info::device_type DeviceType>
class device_typename_selector : public
device_type_selector {

```

```
public:

  device_typename_selector() : device_type_selector {
DeviceType } {}

};


/** Devices selected by heuristics of the system

    If no OpenCL device is found then it defaults to the SYCL host device.
*/
using default_selector =
device_typename_selector<info::device_type::defaults>;


  /** Select devices according to device type info::device::device_type::gpu
    from all the available OpenCL devices.

    If no OpenCL GPU device is found the selector fails.

    Select the best GPU, if any.
*/
using gpu_selector =
device_typename_selector<info::device_type::gpu>;


/** Select devices according to device type info::device::device_type::cpu
    from all the available devices and heuristics

    If no OpenCL CPU device is found the selector fails.
*/
using cpu_selector =
device_typename_selector<info::device_type::cpu>;


/** Selects the SYCL host CPU device that does not require an OpenCL
    runtime
*/
using host_selector =
device_typename_selector<info::device_type::host>;

/// @} to end the execution Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_SELECTOR_DETAIL_DEVICE_SELECTOR_TAIL_HPP
```

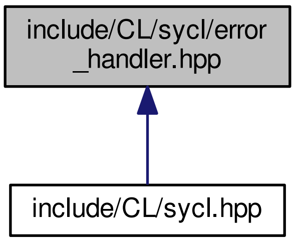
11.63 include/CL/sycl/error_handler.hpp File Reference

```
#include "CL/sycl/exception.hpp"
```

Include dependency graph for error_handler.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- 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

Namespaces

- cl

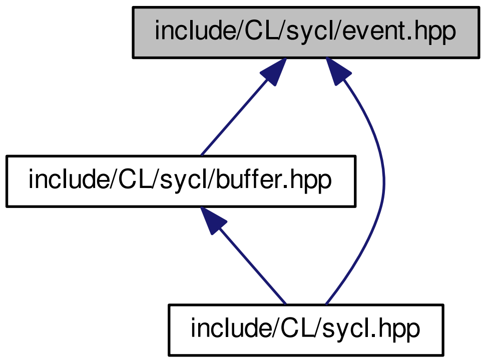
    The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::trisycl

11.64 error_handler.hpp

```
#ifndef TRISYCL_SYCL_ERROR_HANDLER_HPP
#define TRISYCL_SYCL_ERROR_HANDLER_HPP

/** \file The OpenCL SYCL error_handler

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/exception.hpp"

namespace cl {
namespace sycl {

/** \addtogroup error_handling Error handling
    @{
*/

/// \todo Refactor when updating to latest specification
namespace trisycl {
  // Create a default error handler to be used when nothing is specified
  struct default_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
*/
struct error_handler {
  /** The method to define to be called in the case of an error

      \todo Add "virtual void" to the specification
  */
  virtual void report_error(exception &error) = 0;

  /** Add a default_handler to be used by default

      \todo add this concept to the specification?
  */
  static trisycl::default_error_handler
default_handler;
};


namespace trisycl {

  struct default_error_handler : error_handler {

    void report_error(exception &) override {
    }
  };
}

  // \todo finish initialization
  //error_handler::default_handler = nullptr;


/// @} End the error_handling Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ERROR_HANDLER_HPP
```

11.65 include/CL/sycl/event.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class cl::sycl::event

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

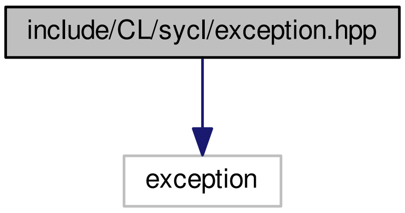
11.66 event.hpp

```
#ifndef TRISYCL_SYCL_EVENT_HPP
#define TRISYCL_SYCL_EVENT_HPP

/** \file The event class

    Ronan at keryell dot FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/
namespace cl {
namespace sycl {

class event {

public:

  event() = default;


/** \todo To be implemented */
#if 0
  explicit event(cl_event clEvent);

  event(const event & rhs);

  cl_event get();

  vector_class<event> get_wait_list();

  void wait();

  static void wait(const vector_class<event> &eventList);

  void wait_and_throw();

  static void wait_and_throw(const vector_class<event> &eventList);

  template <info::event param>
  typename param_traits<info::event, param>::type get_info() const;

  template <info::event_profiling param>
  typename param_traits<info::event_profiling,
                         param>::type get_profiling_info() const;
#endif
};

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_EVENT_HPP
```

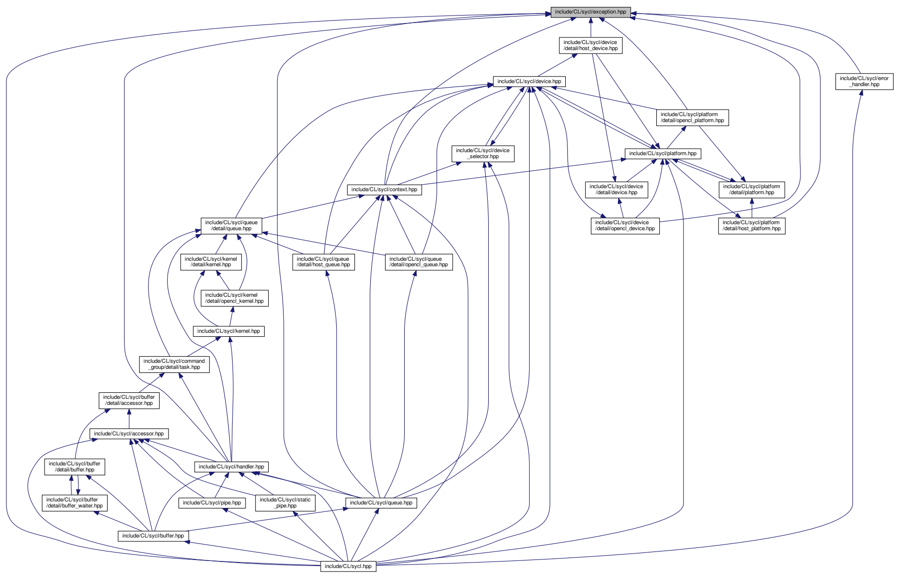
11.67 include/CL/sycl/exception.hpp File Reference

```
#include <exception>
```

Include dependency graph for exception.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/exception.hpp
include/CL/sycl/device/detail/host_device.hpp
include/CL/sycl/error_handler.hpp
include/CL/sycl/device.hpp
include/CL/sycl/platform/detail/opencl_platform.hpp
include/CL/sycl/device_selector.hpp
include/CL/sycl/platform.hpp
include/CL/sycl/context.hpp
include/CL/sycl/device/detail/device.hpp
include/CL/sycl/platform/detail/platform.hpp
include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/device/detail/opencl_device.hpp
include/CL/sycl/platform/detail/host_platform.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- struct cl::sycl::exception_list

  Exception list to store several exceptions. More...
- class cl::sycl::exception

  Encapsulate a SYCL error information. More...
- class cl::sycl::cl_exception

  Returns the OpenCL error code encapsulated in the exception. More...
- struct cl::sycl::async_exception

  An error stored in an exception_list for asynchronous errors. More...
- class cl::sycl::runtime_error
- class cl::sycl::kernel_error

  Error that occurred before or while enqueuing the SYCL kernel. More...
- class cl::sycl::accessor_error

  Error regarding the cl::sycl::accessor objects defined. More...
- class cl::sycl::nd_range_error

  Error regarding the cl::sycl::nd_range specified for the SYCL kernel. More...
- class cl::sycl::event_error

  Error regarding associated cl::sycl::event objects. More...
- class cl::sycl::invalid_parameter_error

  Error regarding parameters to the SYCL kernel, it may apply to any captured parameters to the kernel lambda. More...
- class cl::sycl::device_error

  The SYCL device will trigger this exception on error. More...
- class cl::sycl::compile_program_error

  Error while compiling the SYCL kernel to a SYCL device. More...
- class cl::sycl::link_program_error

  Error while linking the SYCL kernel to a SYCL device. More...
- class cl::sycl::invalid_object_error

  Error regarding any memory objects being used inside the kernel. More...
- class cl::sycl::memory_allocation_error

Error on memory allocation on the SYCL device for a SYCL kernel. More...

- class cl::sycl::pipe_error

  A failing pipe error will trigger this exception on error. More...
- class cl::sycl::platform_error

  The SYCL platform will trigger this exception on error. More...
- class cl::sycl::profiling_error

  The SYCL runtime will trigger this error if there is an error when profiling info is enabled. More...
- class cl::sycl::feature_not_supported

  Exception thrown when an optional feature or extension is used in a kernel but its not available on the device the SYCL kernel is being enqueued on. More...
- class cl::sycl::non_cl_error

  Exception for an OpenCL operation requested in a non OpenCL area. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Typedefs

- using cl::sycl::exception_ptr = std::exception_ptr

  A shared pointer to an exception as in C++ specification.
- using cl::sycl::async_handler = function_class< void, exception_list >

11.68 exception.hpp

```
#ifndef TRISYCL_SYCL_EXCEPTION_HPP
#define TRISYCL_SYCL_EXCEPTION_HPP

/** \file The OpenCL SYCL exception

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <exception>

namespace cl {
namespace sycl {

/** \addtogroup error_handling Error handling
    @{
*/


/** A shared pointer to an exception as in C++ specification

    \todo Do we need this instead of reusing directly the one from C++11?
*/
using exception_ptr = std::exception_ptr;


/** Exception list to store several exceptions

    \todo Do we need to define it in SYCL or can we rely on plain C++17 one?
*/
struct exception_list : std::vector<exception_ptr> {
  using std::vector<exception_ptr>::vector;
};

```

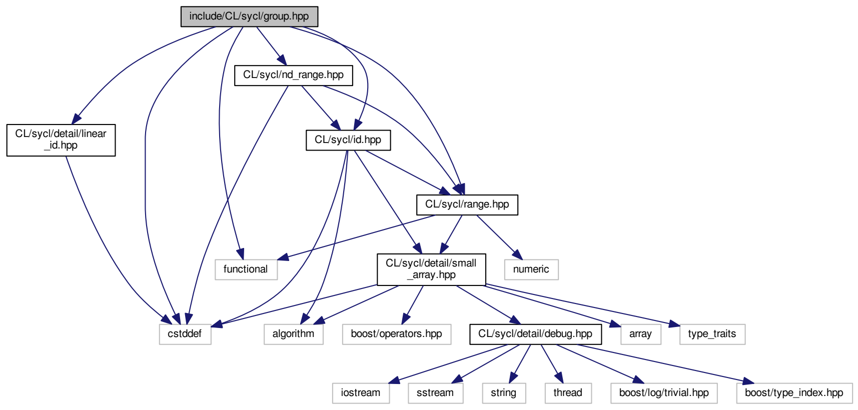
```
using async_handler = function_class<void, exception_list>
;


/// Encapsulate a SYCL error information
class exception {

  /// The error message to return
  string_class message;

public:

  /// Construct an exception with a message for internal use
  exception(const string_class &message) : message { message } {}

  /// Returns a descriptive string for the error, if available
  string_class what() const {
    return message;
  }


  /** Returns the context that caused the error

      Returns nullptr if not a buffer error.

      \todo Cannot return nullptr. Use optional? Use a specific exception type?
  */
  //context get_context()

};


/// Returns the OpenCL error code encapsulated in the exception
class cl_exception : public exception {

#ifdef TRISYCL_OPENCL
  /// The OpenCL error code to return

  cl_int cl_code;

public:

  /**  Construct an exception with a message and OpenCL error code for
       internal use */
  cl_exception(const string_class &message, cl_int cl_code)
    : exception { message }, cl_code { cl_code } {}

  // thrown as a result of an OpenCL API error code
  cl_int get_cl_code() const {
    return cl_code;
  }
#endif

};


/// An error stored in an exception_list for asynchronous errors
struct async_exception : exception {
  using exception::exception;
};


class runtime_error : public exception {
  using exception::exception;
};


/// Error that occurred before or while enqueuing the SYCL kernel
class kernel_error : public runtime_error {
  using runtime_error::runtime_error;
};


/// Error regarding the cl::sycl::accessor objects defined
class accessor_error : public runtime_error {
  using runtime_error::runtime_error;
};


/// Error regarding the cl::sycl::nd_range specified for the SYCL kernel
class nd_range_error : public runtime_error {
  using runtime_error::runtime_error;
};


/// Error regarding associated cl::sycl::event objects
class event_error : public runtime_error {
  using runtime_error::runtime_error;
};


/** Error regarding parameters to the SYCL kernel, it may apply to any
    captured parameters to the kernel lambda
*/
class invalid_parameter_error : public runtime_error {
  using runtime_error::runtime_error;
};


/// The SYCL device will trigger this exception on error
class device_error : public exception {
  using exception::exception;
};


/// Error while compiling the SYCL kernel to a SYCL device
class compile_program_error : public device_error {
  using device_error::device_error;
};


/// Error while linking the SYCL kernel to a SYCL device
class link_program_error : public device_error {
  using device_error::device_error;
};


/// Error regarding any memory objects being used inside the kernel
class invalid_object_error : public device_error {
  using device_error::device_error;
};


/// Error on memory allocation on the SYCL device for a SYCL kernel
class memory_allocation_error : public device_error {
  using device_error::device_error;
};


/// A failing pipe error will trigger this exception on error
class pipe_error : public runtime_error {
  using runtime_error::runtime_error;
};


/// The SYCL platform will trigger this exception on error
class platform_error : public device_error {
  using device_error::device_error;
};


/** The SYCL runtime will trigger this error if there is an error when
    profiling info is enabled
*/
class profiling_error : public device_error {
  using device_error::device_error;
};


/** Exception thrown when an optional feature or extension is used in
    a kernel but its not available on the device the SYCL kernel is
    being enqueued on
*/
class feature_not_supported : public device_error {
  using device_error::device_error;
};


/** Exception for an OpenCL operation requested in a non OpenCL area

    \todo Add to the specification

    \todo Clean implementation

    \todo Exceptions are named error in C++
*/
class non_cl_error : public runtime_error {
  using runtime_error::runtime_error;
};


/// @} End the error_handling Doxygen group

}
```

```
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_EXCEPTION_HPP
```

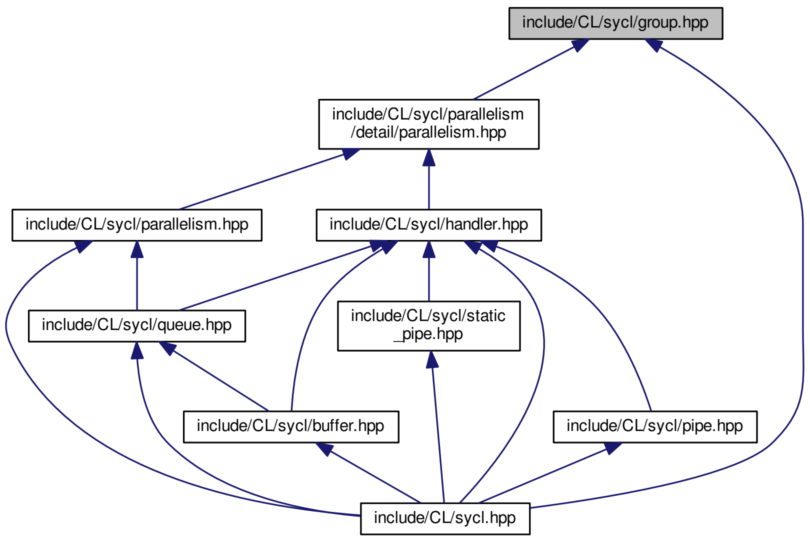
11.69 include/CL/sycl/group.hpp File Reference

```
#include <cstddef>
#include <functional>
#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for group.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/group.hpp
include/CL/sycl/parallelism /detail/parallelism.hpp
include/CL/sycl/parallelism.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/static _pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl.hpp

Classes

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

    A group index used in a parallel_for_workitem to specify a work_group. More...
- struct cl::sycl::group< dims >

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

Namespaces

- cl

    The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_workitem (const group< Dimensions > &g, ParallelForFunctor f)

    Implement the loop on the work-items inside a work-group.

11.70 group.hpp

```
#ifndef TRISYCL_SYCL_GROUP_HPP
#define TRISYCL_SYCL_GROUP_HPP

/** \file The OpenCL SYCL nd_item<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <functional>

#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"

namespace cl {
namespace sycl {

template <std::size_t dims = 1>
struct group;

namespace detail {

template <std::size_t Dimensions = 1, typename ParallelForFunctor>
void parallel_for_workitem(const group<Dimensions> &g,
                           ParallelForFunctor f);

}

/** \addtogroup parallelism Expressing parallelism through kernels
    @{
*/

/** A group index used in a parallel_for_workitem to specify a work_group
 */
template <std::size_t dims>
struct group {
  /// \todo add this Boost::multi_array or STL concept to the
  /// specification?
  static constexpr auto dimensionality = dims;

private:

  /// The coordinate of the group item
  id<dims> group_id;

  /// Keep a reference on the nd_range to serve potential query on it
  nd_range<dims> ndr;

public:

  /** Create a group from an nd_range<> with a 0 id<>

      \todo This should be private since it is only used by the triSYCL
      implementation
  */
  group(const nd_range<dims> &ndr) : ndr { ndr } {}


  /** Create a group from an id and a nd_range<>

      \todo This should be private somehow, but it is used by the
      validation infrastructure
  */
  group(const id<dims> &i, const nd_range<dims> &ndr ) :
    group_id { i }, ndr { ndr } {}


  /** To be able to copy and assign group, use default constructors too

      \todo Make most of them protected, reserved to implementation
  */
  group() = default;


  /** Return an id representing the index of the group within the nd_range
      for every dimension
  */
  id<dims> get() const { return group_id; }

```

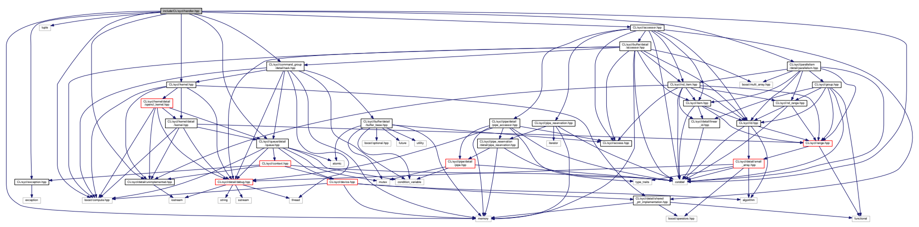
```

/// Return the index of the group in the given dimension
size_t get(int dimension) const { return get()[dimension]; }


/** Return the index of the group in the given dimension within the
    nd_range<>

    \todo In this implementation it is not const because the group<> is
    written in the parallel_for iterators. To fix according to the
    specification
*/
auto &operator[](int dimension) {
  return group_id[dimension];
}


/** Return a range<> representing the dimensions of the current
    group

    This local range may have been provided by the programmer, or chosen
    by the runtime.

    \todo Fix this comment and the specification
*/
range<dims> get_group_range() const {
  return get_nd_range().get_group();
}


/// Return element dimension from the con stituent group range
size_t get_group_range(int dimension) const {
  return get_group_range()[dimension];
}


/// Get the local range for this work_group
range<dims> get_global_range() const { return get_nd_range().get_global(); }


/// Return element dimension from the constituent global range
size_t get_global_range(int dimension) const {
  return get_global_range()[dimension];
}


/** Get the local range for this work_group

    \todo Add to the specification
*/
range<dims> get_local_range() const { return get_nd_range().get_local(); }


/** Return element dimension from the constituent local range

    \todo Add to the specification
*/
size_t get_local_range(int dimension) const {
  return get_local_range()[dimension];
}


/**  Get the offset of the NDRange

     \todo Add to the specification
*/
id<dims> get_offset() const { return get_nd_range().get_offset(); }


/**  Get the offset of the NDRange

     \todo Add to the specification
*/
size_t get_offset(int dimension) const { return get_offset()[dimension]; }


/// \todo Also provide this access to the current nd_range
nd_range<dims> get_nd_range() const { return ndr; }


/** Get a linearized version of the group ID

 */
size_t get_linear() const {
  return detail::linear_id(get_group_range(), get());
}

```

```

  /** Loop on the work-items inside a work-group

      \todo Add this method in the specification
  */
  void parallel_for_work_item(std::function<void(nd_item<dimensionality>)> f)
    const {
    detail::parallel_for_workitem(*this, f);
  }


  /** Loop on the work-items inside a work-group

      \todo Add this method in the specification
  */
  void parallel_for_work_item(std::function<void(item<dimensionality>)> f)
    const {
    auto item_adapter = [=] (nd_item<dimensionality> ndi) {
      item<dimensionality> i = ndi.get_item();
      f(i);
    };
    detail::parallel_for_workitem(*this, item_adapter);
  }

};

/// @} End the parallelism Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_GROUP_HPP
```

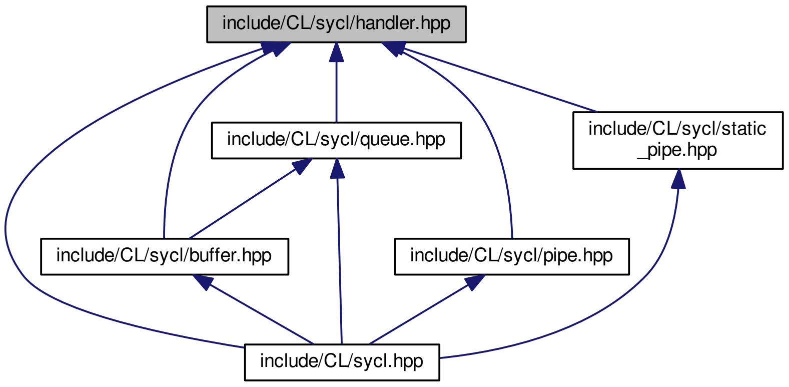
11.71 include/CL/sycl/handler.hpp File Reference

```
#include <cstddef>
#include <memory>
#include <tuple>
#include <boost/compute.hpp>
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/kernel.hpp"
#include "CL/sycl/parallelism/detail/parallelism.hpp"
#include "CL/sycl/queue/detail/queue.hpp"
```

Include dependency graph for handler.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/handler.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/static
_pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::handler

    Command group handler class. More...

Namespaces

- cl

    The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Macros

- #define TRISYCL_parallel_for_functor_GLOBAL(N)

    SYCL parallel_for launches a data parallel computation with parallelism specified at launch time by a range<>
- #define TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(N)
- #define TRISYCL_ParallelForKernel_RANGE(N)

    Kernel invocation method of a kernel defined as a kernel object, for the specified range and given an id or item for indexing in the indexing space defined by range, described in detail in 5.4.

Functions

- static std::shared_ptr< detail::task > cl::sycl::detail::add_buffer_to_task (handler ∗command_group_handler, std::shared_ptr< detail::buffer_base > b, bool is_write_mode)

    Register a buffer as used by a task.

11.71.1 Macro Definition Documentation

11.71.1.1 #define TRISYCL_parallel_for_functor_GLOBAL(*N*)

Value:

```
template <typename KernelName = std::nullptr_t,                        \
            typename ParallelForFunctor>                                  \
  void parallel_for(range<N> global_size,                                 \
                    ParallelForFunctor f) {                               \
    task->schedule(detail::trace_kernel<KernelName>([=] {                 \
          detail::parallel_for(global_size, f);                           \
        }));                                                              \
  }
```

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

Kernel invocation method of a kernel defined as a lambda or functor, for the specified range and given an id or item for indexing in the indexing space defined by range.

If it is a lambda function or the if the functor type is globally visible there is no need for the developer to provide a kernel name type (typename KernelName) for it, as described in detail in 3.5.3

Parameters

global_size	is the full size of the range<>
N	dimensionality of the iteration space
f	is the kernel functor to execute
KernelName	is a class type that defines the name to be used for the underlying kernel

Unfortunately, to have implicit conversion to work on the range, the function can not be templated, so instantiate it for all the dimensions

Definition at line 202 of file handler.hpp.

11.71.1.2 #define TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(*N*)

Value:

```
template <typename KernelName = std::nullptr_t,          \
            typename ParallelForFunctor>                    \
  void parallel_for(range<N> global_size,                   \
                    id<N> offset,                           \
                    ParallelForFunctor f) {                 \
    task->schedule(detail::trace_kernel<KernelName>([=] {   \
          detail::parallel_for_global_offset(global_size,   \
                                             offset,        \
                                             f);            \
        }));                                                \
  }
```

11.71.1.3 #define TRISYCL_ParallelForKernel_RANGE(*N*)

Value:

```
void parallel_for(range<N> num_work_items,                              \
                    kernel sycl_kernel) {                                 \
    /* For now just use the usual host task system to schedule           \
       manually the OpenCL kernels instead of using OpenCL event-based   \
       scheduling                                                         \
                                                                          \
       \todo Move the tracing inside the kernel implementation           \
                                                                          \
       \todo Simplify this 2 step ugly interface                         \
    */                                                                    \
    task->set_kernel(sycl_kernel.implementation);                        \
    /* Use an intermediate variable to capture task by copy because      \
       otherwise "this" is captured by reference and havoc with task     \
       just accessing the dead "this". Nasty bug to find... */           \
    task->schedule(detail::trace_kernel<kernel>([=, t = task] {          \
          sycl_kernel.implementation->parallel_for(t, t->get_queue(),    \
                                                    num_work_items); })); \
  }
```

Kernel invocation method of a kernel defined as a kernel object, for the specified range and given an id or item for indexing in the indexing space defined by range, described in detail in 5.4.

Todo Add in the spec a version taking a kernel and a functor, to have host fall-back

Definition at line 340 of file handler.hpp.

11.72 handler.hpp

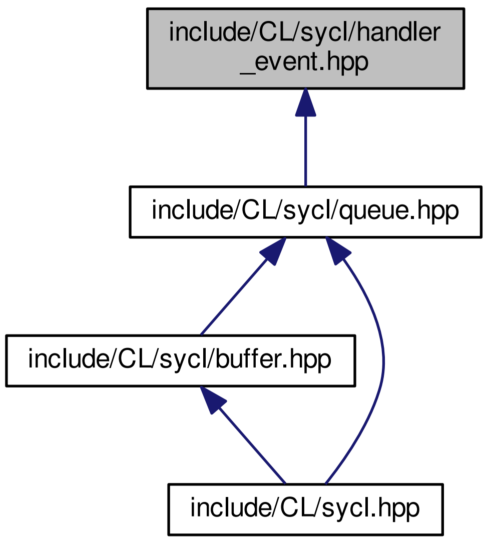
```
#ifndef TRISYCL_SYCL_HANDLER_HPP
#define TRISYCL_SYCL_HANDLER_HPP

/** \file The OpenCL SYCL command group handler

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <memory>
#include <tuple>

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/accessor.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/kernel.hpp"
#include "CL/sycl/parallelism/detail/parallelism.hpp"
#include "CL/sycl/queue/detail/queue.hpp"

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/** Command group handler class

    A command group handler object can only be constructed by the SYCL runtime.

    All of the accessors defined in the command group scope take as a
    parameter an instance of the command group handler and all the kernel
    invocation functions are methods of this class.
*/
class handler {

public:

  /** Attach the task and accessors to it.
   */
  std::shared_ptr<detail::task> task;


  /* Create a command group handler from the queue detail

     The queue detail is used to track kernel completion.

     Note that this is an implementation dependent constructor. Normal
     users cannot construct handler from scratch.

     \todo Make this constructor private
  */
  handler(const std::shared_ptr<detail::queue> &q) {
    // Create a new task for this command_group
    task = std::make_shared<detail::task>(q);
  }


#ifdef TRISYCL_OPENCL
  /** Set kernel arg for an OpenCL kernel which is used through the
      SYCL/OpenCL interop interface

      The index value specifies which parameter of the OpenCL kernel is
      being set and the accessor object, which OpenCL buffer or image is
      going to be given as kernel argument.

      \todo Update the specification to use a ref && to the accessor instead?

      \todo It is not that clean to have set_arg() associated to a
      command handler. Rethink the specification?

      \todo It seems more logical to have these methods on kernel instead
  */
  template <typename DataType,
            std::size_t Dimensions,
            access::mode Mode,
            access::target Target = access::target::global_buffer
>
  void set_arg(int arg_index,
               accessor<DataType, Dimensions, Mode, Target>
acc_obj) {
    /* Before running the kernel, make sure the cl_mem behind this
       accessor is up-to-date on the device if needed and pass it to
       the kernel.

       Explicitly capture task by copy instead of having this captured
       by reference and task by reference by side effect */
    task->add_prelude([=, task = task] {
        acc_obj.implementation->copy_in_cl_buffer();
        task->get_kernel().get_boost_compute()
          .set_arg(arg_index, acc_obj.implementation->get_cl_buffer());
      });
    /* After running the kernel, make sure the cl_mem behind this
       accessor is up-to-date on the host if needed */
    task->add_postlude([=] {
        acc_obj.implementation->copy_back_cl_buffer();
      });
  }


  /** Set kernel args for an OpenCL kernel which is used through the
      SYCL/OpenCL interoperability interface

      The index value specifies which parameter of the OpenCL kernel is
      being set and the accessor object, which OpenCL buffer or image is
      going to be given as kernel argument.

      \todo It is not that clean to have set_arg() associated to a
      command handler. Rethink the specification?

      \todo To be implemented
  */
  template <typename T>
  void set_arg(int arg_index, T scalar_value) {
    detail::unimplemented();
  }


```

```
private:

  /// Helper to individually call set_arg() for each argument
  template <std::size_t... Is, typename... Ts>
  void dispatch_set_arg(std::index_sequence<Is...>, Ts&&... args) {
    // Use an intermediate tuple to ease individual argument access
    auto &&t = std::make_tuple(std::forward<Ts>(args)...);
    // Dispatch individual set_arg() for each argument
    auto just_to_evaluate = {
      0 /*< At least 1 element to deal with empty set_args() */,
      ( set_arg(Is, std::forward<Ts>(std::get<Is>(t))), 0)...
    };
    // Remove the warning about unused variable
    static_cast<void>(just_to_evaluate);
  }

public:

  /** Set all kernel args for an OpenCL kernel which is used through the
      SYCL/OpenCL interop interface

      \todo Update the specification to add this function according to
      https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15978 proposal
  */
  template <typename... Ts>
  void set_args(Ts&&... args) {
    /* Construct a set of increasing argument index to be able to call
       the real set_arg */
    dispatch_set_arg(std::make_index_sequence<sizeof...(Ts)>{},
                     std::forward<Ts>(args)...);
  }
#endif


  /** Kernel invocation method of a kernel defined as a lambda or
      functor. If it is a lambda function or the functor type is globally
      visible there is no need for the developer to provide a kernel name type
      (typename KernelName) for it, as described in 3.5.3

      SYCL single_task launches a computation without parallelism at
      launch time.

      \param F specify the kernel to be launched as a single_task

      \param KernelName is a class type that defines the name to be used for
      the underlying kernel
  */
  template <typename KernelName = std::nullptr_t>
  void single_task(std::function<void(void)> F) {
    task->schedule(detail::trace_kernel<KernelName>(F));
  }


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

      Kernel invocation method of a kernel defined as a lambda or functor,
      for the specified range and given an id or item for indexing in the
      indexing space defined by range.

      If it is a lambda function or the if the functor type is globally
      visible there is no need for the developer to provide a kernel name
      type (typename KernelName) for it, as described in detail in 3.5.3

      \param global_size is the full size of the range<>

      \param N dimensionality of the iteration space

      \param f is the kernel functor to execute

      \param KernelName is a class type that defines the name to be used
      for the underlying kernel

      Unfortunately, to have implicit conversion to work on the range, the
      function can not be templated, so instantiate it for all the
      dimensions
  */
#define TRISYCL_parallel_for_functor_GLOBAL(N)                          \
  template <typename KernelName = std::nullptr_t,                       \
            typename ParallelForFunctor>                                \
  void parallel_for(range<N> global_size,                               \
                    ParallelForFunctor f) {                             \
    task->schedule(detail::trace_kernel<KernelName>([=] {               \
          detail::parallel_for(global_size, f);                         \
        }));                                                            \
  }

  TRISYCL_parallel_for_functor_GLOBAL(1)
  TRISYCL_parallel_for_functor_GLOBAL(2)
  TRISYCL_parallel_for_functor_GLOBAL(3)


  /** Kernel invocation method of a kernel defined as a lambda or functor,
      for the specified range and offset and given an id or item for
      indexing in the indexing space defined by range

      If it is a lambda function or the if the functor type is globally
      visible there is no need for the developer to provide a kernel name
      type (typename KernelName) for it, as described in detail in 3.5.3

      \param global_size is the global size of the range<>

      \param offset is the offset to be add to the id<> during iteration

      \param f is the kernel functor to execute

      \param ParallelForFunctor is the kernel functor type

      \param KernelName is a class type that defines the name to be used for
      the underlying kernel

      Unfortunately, to have implicit conversion to work on the range, the
      function can not be templated, so instantiate it for all the
      dimensions
  */
#define TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(N)       \
  template <typename KernelName = std::nullptr_t,         \
            typename ParallelForFunctor>                  \
  void parallel_for(range<N> global_size,                 \
                    id<N> offset,                         \
                    ParallelForFunctor f) {               \
    task->schedule(detail::trace_kernel<KernelName>([=] { \
        detail::parallel_for_global_offset(global_size, \
                                           offset,       \
                                           f);           \
      }));                                                \
  }

  TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(1)
  TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(2)
  TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(3)


  /** Kernel invocation method of a kernel defined as a lambda or functor,
      for the specified nd_range and given an nd_item for indexing in the
      indexing space defined by the nd_range

      If it is a lambda function or the if the functor type is globally
      visible there is no need for the developer to provide a kernel name
      type (typename KernelName) for it, as described in detail in 3.5.3

      \param r defines the iteration space with the work-group layout and
      offset

      \param Dimensions dimensionality of the iteration space

      \param f is the kernel functor to execute

      \param ParallelForFunctor is the kernel functor type

      \param KernelName is a class type that defines the name to be used for
      the underlying kernel
  */
  template <typename KernelName,
            std::size_t Dimensions,
            typename ParallelForFunctor>
  void parallel_for(nd_range<Dimensions> r, ParallelForFunctor f) {
    task->schedule(detail::trace_kernel<KernelName>([=] {
          detail::parallel_for(r, f);
      }));
  }


  /** Hierarchical kernel invocation method of a kernel defined as a
      lambda encoding the body of each work-group to launch

      May contain multiple kernel built-in parallel_for_work_item
      functions representing the execution on each work-item.

      Launch num_work_groups work-groups of runtime-defined
      size. Described in detail in 3.5.3.

      \param r defines the iteration space with the work-group layout and
      offset

      \param Dimensions dimensionality of the iteration space

      \param f is the kernel functor to execute

      \param ParallelForFunctor is the kernel functor type

      \param KernelName is a class type that defines the name to be used for
      the underlying kernel
  */
  template <typename KernelName = std::nullptr_t,
            std::size_t Dimensions = 1,
            typename ParallelForFunctor>
  void parallel_for_work_group(nd_range<Dimensions> r,
                               ParallelForFunctor f) {
    task->schedule(detail::trace_kernel<KernelName>([=] {
          detail::parallel_for_workgroup(r, f); }));
  }


  /** Kernel invocation method of a kernel defined as pointer to a kernel
      object, described in detail in 3.5.3

      \todo Add in the spec a version taking a kernel and a functor,
      to have host fall-back

      \todo To be implemented
  */
  void single_task(kernel syclKernel) {
    detail::unimplemented();
  }


  /** Kernel invocation method of a kernel defined as a kernel object,
      for the specified range and given an id or item for indexing in
      the indexing space defined by range, described in detail in
      5.4.

      \todo Add in the spec a version taking a kernel and a functor,
      to have host fall-back
  */
#define TRISYCL_ParallelForKernel_RANGE(N)                           \
  void parallel_for(range<N> num_work_items,                         \
                    kernel sycl_kernel) {                            \
    /* For now just use the usual host task system to schedule       \
       manually the OpenCL kernels instead of using OpenCL event-based \
       scheduling                                                    \
                                                                     \
       \todo Move the tracing inside the kernel implementation       \
                                                                     \
       \todo Simplify this 2 step ugly interface                     \
    */                                                               \
    task->set_kernel(sycl_kernel.implementation);                    \
    /* Use an intermediate variable to capture task by copy because  \
       otherwise "this" is captured by reference and havoc with task \
       just accessing the dead "this". Nasty bug to find... */       \
    task->schedule(detail::trace_kernel<kernel>([=, t = task] {      \
          sycl_kernel.implementation->parallel_for(t, t->get_queue(), \
                                                   num_work_items); })); \
  }

  /* Do not use a template parameter since otherwise the parallel_for
     functor is selected instead of this one

     \todo Clean this
  */
  TRISYCL_ParallelForKernel_RANGE(1)
  TRISYCL_ParallelForKernel_RANGE(2)
  TRISYCL_ParallelForKernel_RANGE(3)
#undef TRISYCL_ParallelForKernel_RANGE

  /** Kernel invocation method of a kernel defined as pointer to a kernel
      object, for the specified nd_range and given an nd_item for indexing
      in the indexing space defined by the nd_range, described in detail
      in 3.5.3

      \todo Add in the spec a version taking a kernel and a functor,
      to have host fall-back

      \todo To be implemented
  */
  template <std::size_t Dimensions = 1>
  void parallel_for(nd_range<Dimensions>, kernel syclKernel) {
    detail::unimplemented();
  }

};
```

```

namespace detail {

/** Register a buffer as used by a task

    This is a proxy function to avoid complicated type recursion.
*/
static std::shared_ptr<detail::task>
add_buffer_to_task(handler *command_group_handler,
                   std::shared_ptr<detail::buffer_base> b,
                   bool is_write_mode) {
  command_group_handler->task->add_buffer(b, is_write_mode);
  return command_group_handler->task;
}

}

/// @} End the execution Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_HANDLER_HPP
```

11.73 include/CL/sycl/handler_event.hpp File Reference

This graph shows which files directly or indirectly include this file:


include/CL/sycl/handler _event.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class handler_event

    Handler event.

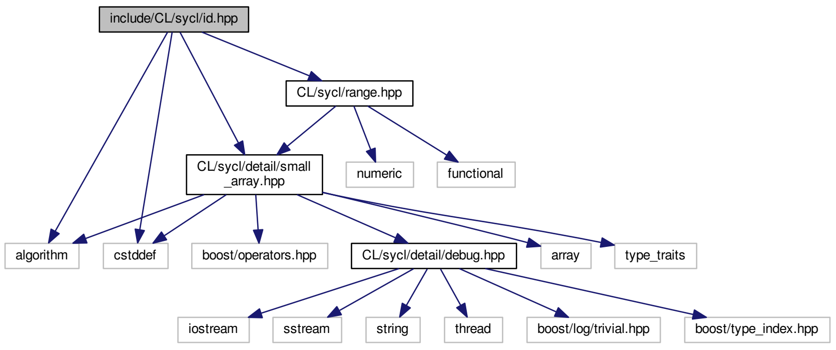
11.74 handler_event.hpp

```
#ifndef TRISYCL_SYCL_HANDLER_EVENT_HPP
#define TRISYCL_SYCL_HANDLER_EVENT_HPP

/** \file The handler event

    Implement parallel constructions to launch kernels

    Ronan at keryell dot FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

/** \todo To be implemented */
/** Handler event

    \todo To be implemented
*/
class handler_event {
/*
 public:
  event get_kernel() const;
  event get_complete() const;
  event get_end() const;
*/
};


/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_HANDLER_EVENT_HPP
```

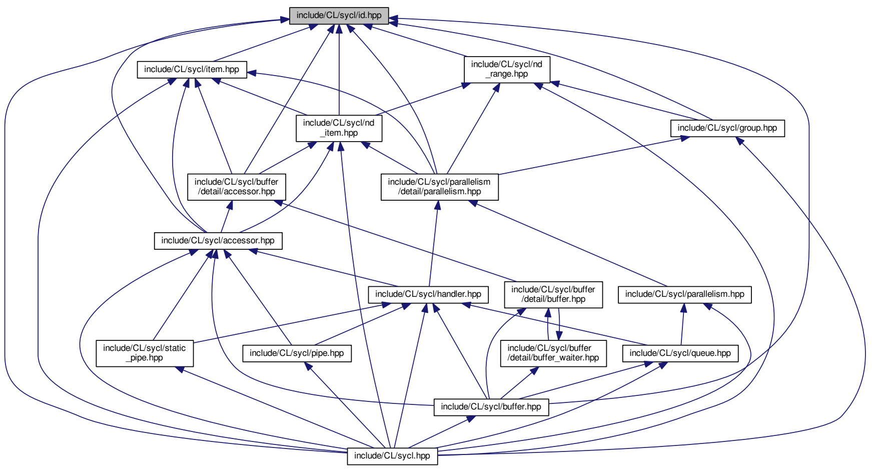
11.75 include/CL/sycl/id.hpp File Reference

```
#include <algorithm>
#include <cstddef>
#include "CL/sycl/detail/small_array.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for id.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/id.hpp
include/CL/sycl/item.hpp
include/CL/sycl/nd_range.hpp
include/CL/sycl/nd_item.hpp
include/CL/sycl/group.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/parallelism/detail/parallelism.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/parallelism.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::item< dims >

  A SYCL item stores information on a work-item with some more context such as the definition range and offset. More...

- class cl::sycl::id< dims >

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

Namespaces

- cl

  The vector type to be used as SYCL vector.

- cl::sycl

Functions

- auto cl::sycl::make_id (id< 1 > i)

  Implement a make_id to construct an id<> of the right dimension with implicit conversion from an initializer list for example.

- auto cl::sycl::make_id (id< 2 > i)
- auto cl::sycl::make_id (id< 3 > i)
- template<typename... BasicType>
  auto cl::sycl::make_id (BasicType...Args)

  Construct an id<> from a function call with arguments, like make_id(1, 2, 3)

11.76 id.hpp

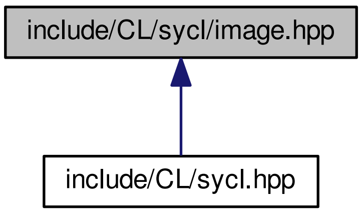
```
#ifndef TRISYCL_SYCL_ID_HPP
#define TRISYCL_SYCL_ID_HPP

/** \file The OpenCL SYCL id<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <algorithm>
#include <cstddef>

#include "CL/sycl/detail/small_array.hpp"
#include "CL/sycl/range.hpp"

namespace cl {
namespace sycl {

template <std::size_t dims> class item;

/** \addtogroup parallelism Expressing parallelism through kernels
    @{
*/

/** Define a multi-dimensional index, used for example to locate a work
    item
*/
template <std::size_t dims = 1>
class id : public detail::small_array_123<std::size_t, id<dims>, dims> {

public:

  // Inherit from all the constructors
  using detail::small_array_123<std::size_t,
                                id<dims>,
                                dims>::small_array_123;


  /// Construct an id from the dimensions of a range
  id(const range<dims> &range_size)
    /** Use the fact we have a constructor of a small_array from a another
        kind of small_array
     */
    : detail::small_array_123<std::size_t, id<dims>, dims> { range_size } {}


  /// Construct an id from an item global_id
  id(const item<dims> &rhs)
    : detail::small_array_123<std::size_t, id<dims>, dims>
      { rhs.get() }
  {}

  /// Keep other constructors
  id() = default;

};


/** Implement a make_id to construct an id<> of the right dimension with
    implicit conversion from an initializer list for example.

    Cannot use a template on the number of dimensions because the implicit
    conversion would not be tried. */
inline auto make_id(id<1> i) { return i; }
inline auto make_id(id<2> i) { return i; }
inline auto make_id(id<3> i) { return i; }


/** Construct an id<> from a function call with arguments, like
    make_id(1, 2, 3) */
template<typename... BasicType>
auto make_id(BasicType... Args) {
  // Call constructor directly to allow narrowing
  return id<sizeof...(Args)>(Args...);
}

/// @} End the parallelism Doxygen group

}
}

/*
```

```
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ID_HPP
```

11.77 include/CL/sycl/image.hpp File Reference

OpenCL SYCL image class.

This graph shows which files directly or indirectly include this file:



Classes

- struct cl::sycl::image< dimensions >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.77.1 Detailed Description

OpenCL SYCL image class.

Ronan at Keryell point FR

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

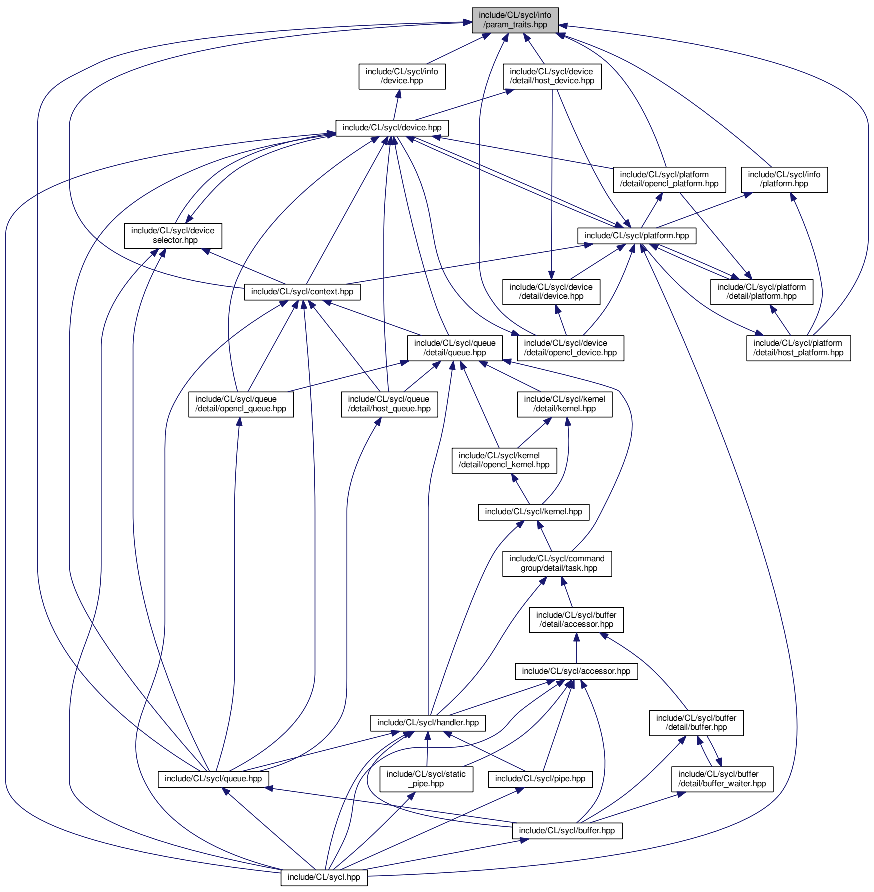
Definition in file image.hpp.

11.78 image.hpp

```
#ifndef TRISYCL_SYCL_IMAGE_HPP
#define TRISYCL_SYCL_IMAGE_HPP

/** \file

    OpenCL SYCL image class

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

namespace cl {
namespace sycl {

/** \addtogroup data

    @{
*/

/// \todo implement image
template <std::size_t dimensions> struct image;


/// @} End the data Doxygen group


}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_IMAGE_HPP
```

11.79 include/CL/sycl/info/param_traits.hpp File Reference

This graph shows which files directly or indirectly include this file:


include/CL/sycl/info/param_traits.hpp
include/CL/sycl/info/device.hpp
include/CL/sycl/device/detail/host_device.hpp
include/CL/sycl/device.hpp
include/CL/sycl/platform/detail/opencl_platform.hpp
include/CL/sycl/info/platform.hpp
include/CL/sycl/device_selector.hpp
include/CL/sycl/platform.hpp
include/CL/sycl/context.hpp
include/CL/sycl/device/detail/device.hpp
include/CL/sycl/platform/detail/platform.hpp
include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/device/detail/opencl_device.hpp
include/CL/sycl/platform/detail/host_platform.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- struct cl::sycl::info::param_traits< T, Param >

    Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value)

Namespaces

- cl

    The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info

Macros

- #define TRISYCL_INFO_PARAM_TRAITS_ANY_T(T, RETURN_TYPE)

    To declare a param_traits returning RETURN_TYPE for function of any T.
- #define TRISYCL_INFO_PARAM_TRAITS(VALUE, RETURN_TYPE)

    To declare a param_traits returning RETURN_TYPE for function taking a VALUE of type T.

11.79.1 Macro Definition Documentation

11.79.1.1 #define TRISYCL_INFO_PARAM_TRAITS(*VALUE*, *RETURN_TYPE*)

Value:

```
template <>                                                        \
  struct param_traits<decltype(VALUE), VALUE> {                      \
    using type = RETURN_TYPE;                                        \
  };
```

To declare a param_traits returning RETURN_TYPE for function taking a VALUE of type T.

Definition at line 36 of file param_traits.hpp.

11.79.1.2 #define TRISYCL_INFO_PARAM_TRAITS_ANY_T(*T*, *RETURN_TYPE*)

Value:

```
template <T Param>                                                 \
  struct param_traits<T, Param> {                                    \
    using type = RETURN_TYPE;                                        \
  };
```

To declare a param_traits returning RETURN_TYPE for function of any T.

Definition at line 26 of file param_traits.hpp.

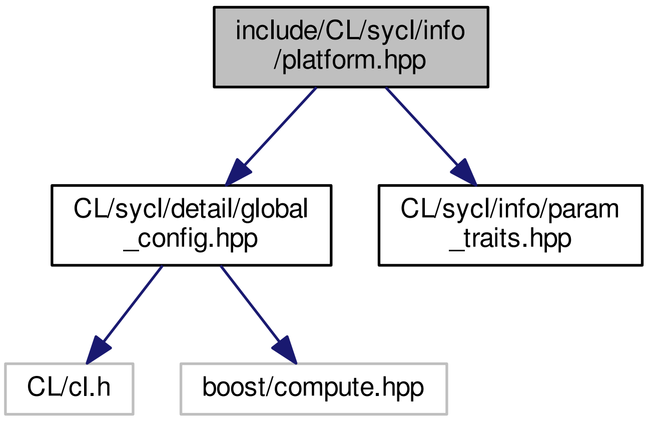
11.80 param_traits.hpp

```
00001 #ifndef TRISYCL_SYCL_INFO_PARAM_TRAITS_HPP
00002 #define TRISYCL_SYCL_INFO_PARAM_TRAITS_HPP
00003
00004 /** \file The OpenCL SYCL param_traits
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 namespace cl {
00013 namespace sycl {
00014 namespace info {
00015
00016 /** Implement a meta-function from (T, value) to T' to express the return type
00017     value of an OpenCL function of kind (T, value)
00018 */
00019 template <typename T, T Param>
00020 struct param_traits {
00021   // By default no return type
00022 };
00023
00024
00025 /// To declare a param_traits returning RETURN_TYPE for function of any T
00026 #define TRISYCL_INFO_PARAM_TRAITS_ANY_T(T, RETURN_TYPE)      \
00027   template <T Param>                                         \
00028   struct param_traits<T, Param> {                            \
00029     using type = RETURN_TYPE;                                \
00030   };
00031
00032
00033 /** To declare a param_traits returning RETURN_TYPE for function taking a
00034     VALUE of type T
*/
#define TRISYCL_INFO_PARAM_TRAITS(VALUE, RETURN_TYPE)        \
  template <>                                                \
  struct param_traits<decltype(VALUE), VALUE> {              \
    using type = RETURN_TYPE;                                \
  };

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_INFO_PARAM_TRAITS_HPP
```

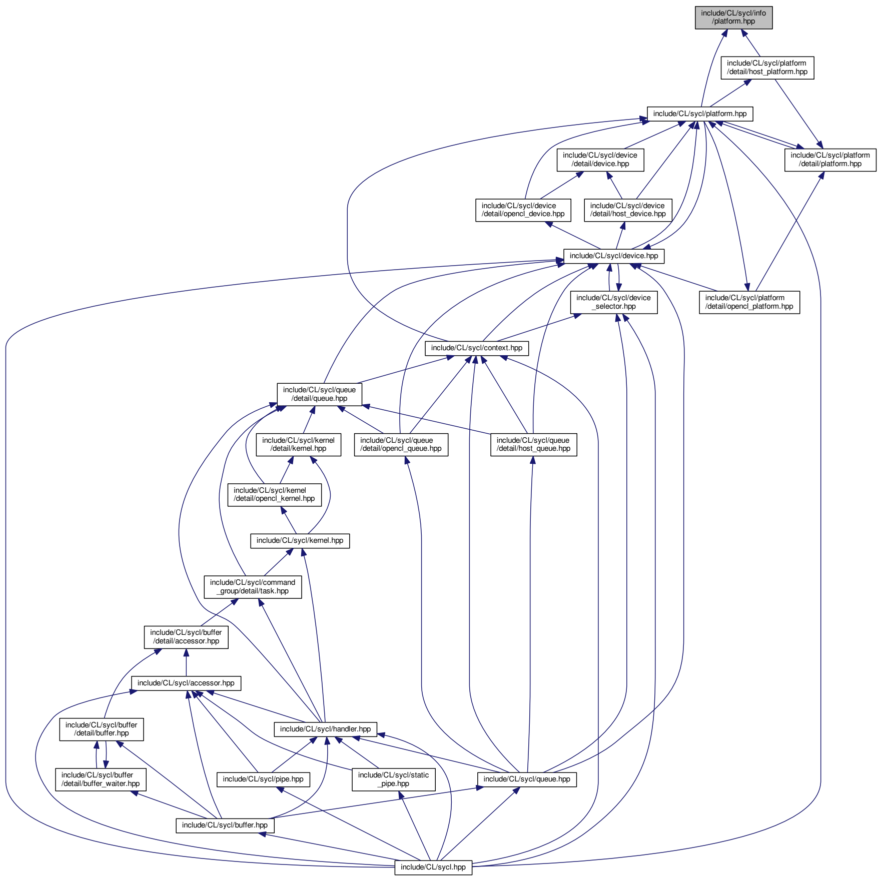
11.81 include/CL/sycl/info/platform.hpp File Reference

```
#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/info/param_traits.hpp"
```

Include dependency graph for platform.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/info/platform.hpp
include/CL/sycl/platform/detail/host_platform.hpp
include/CL/sycl/platform.hpp
include/CL/sycl/device/detail/device.hpp
include/CL/sycl/platform/detail/platform.hpp
include/CL/sycl/device/detail/opencl_device.hpp
include/CL/sycl/device/detail/host_device.hpp
include/CL/sycl/device.hpp
include/CL/sycl/device_selector.hpp
include/CL/sycl/platform/detail/opencl_platform.hpp
include/CL/sycl/context.hpp
include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info

Enumerations

- enum cl::sycl::info::platform : unsigned int {
  cl::sycl::info::platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_PROFILE), cl::sycl::info::platform::↩
  TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_VERSION), cl::sycl::info::platform::TRISYCL_SKIP_OPE↩
  NCL =(= CL_PLATFORM_NAME), cl::sycl::info::platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM↩
  _VENDOR),
  cl::sycl::info::platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_EXTENSIONS) }

  Platform information descriptors.

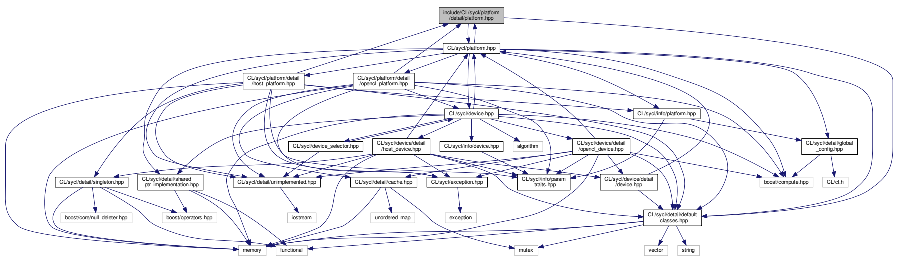
11.82 platform.hpp

```
#ifndef TRISYCL_SYCL_INFO_PLATFORM_HPP
#define TRISYCL_SYCL_INFO_PLATFORM_HPP

/** \file The OpenCL SYCL platform information parameters

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/info/param_traits.hpp"

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/
namespace info {

/** Platform information descriptors

    A SYCL platform can be queried for all of the following information
    using the get_info function.

    In this implementation, the values are mapped to OpenCL values to
    avoid further remapping later when OpenCL is used
*/
enum class platform : unsigned int {
  /** Returns the profile name (as a string_class) supported by the
      implementation.

      Can be either FULL PROFILE or EMBEDDED PROFILE.
  */
  profile TRISYCL_SKIP_OPENCL(= CL_PLATFORM_PROFILE),

  /** Returns the OpenCL software driver version string in the form major
      number.minor number (as a string_class)
  */
  version TRISYCL_SKIP_OPENCL(= CL_PLATFORM_VERSION),

  /** Returns the name of the platform (as a string_class)
  */
  name TRISYCL_SKIP_OPENCL(= CL_PLATFORM_NAME),

  /** Returns the string provided by the platform vendor (as a string_class)
  */
  vendor TRISYCL_SKIP_OPENCL(= CL_PLATFORM_VENDOR),

  /** Returns a space-separated list of extension names supported by the
      platform (as a string_class)
  */
  extensions TRISYCL_SKIP_OPENCL(= CL_PLATFORM_EXTENSIONS),

#if CL_SYCL_LANGUAGE_VERSION >= 220 && defined(CL_VERSION_2_1)
  /** Returns the resolution of the host timer in nanoseconds as used by
      clGetDeviceAndHostTimer
  */
  host_timer_resolution
    TRISYCL_SKIP_OPENCL(= CL_PLATFORM_HOST_TIMER_RESOLUTION)
#endif
};


/** Query the return type for get_info() on platform parameter type

    This defines the meta-function
    \code
    param_traits<info::platform x, string_class>::type == string_class
    \endcode

    for all x, which means that get_info() returns always a string_class
    when asked about platform info.
*/
TRISYCL_INFO_PARAM_TRAITS_ANY_T(info::platform,
string_class)

#if CL_SYCL_LANGUAGE_VERSION >= 220 && defined(CL_VERSION_2_1)
///  get_info<host_timer_resolution>() return a cl_ulong
#ifdef TRISYCL_OPENCL
TRISYCL_INFO_PARAM_TRAITS(info::platform::host_timer_resolution, cl_ulong)
#else
TRISYCL_INFO_PARAM_TRAITS(info::platform::host_timer_resolution,
                          unsigned long int)
#endif
#endif
}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_INFO_PLATFORM_HPP
```

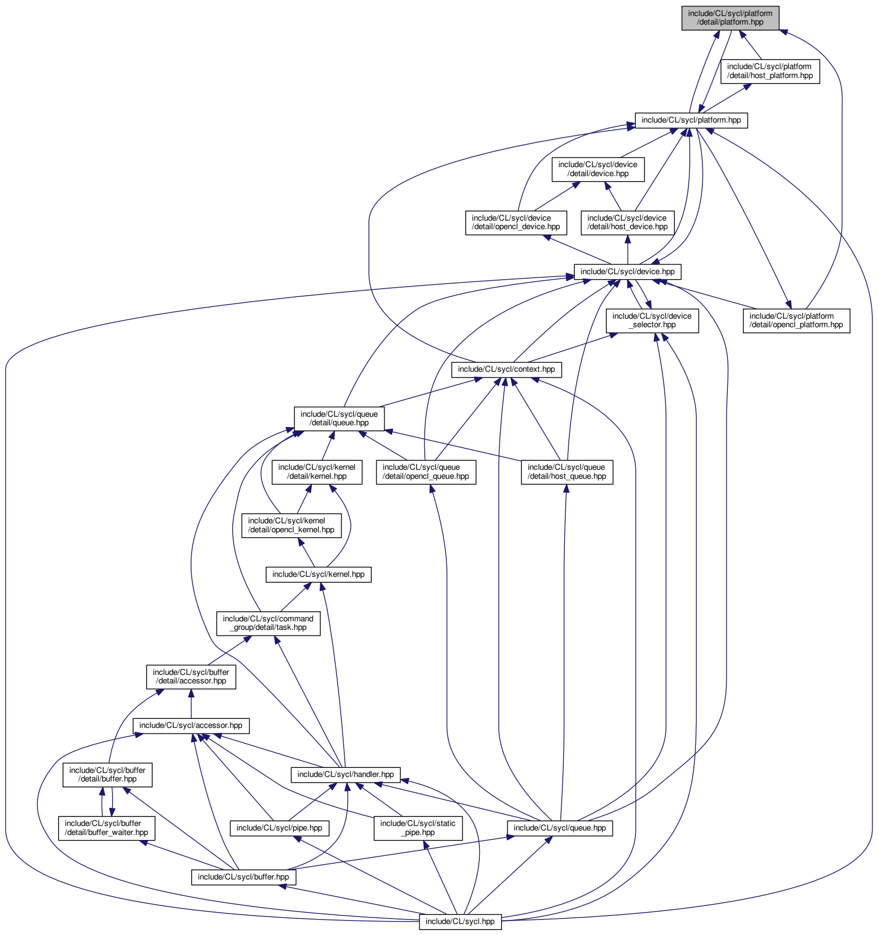
11.83 include/CL/sycl/platform/detail/platform.hpp File Reference

```
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for platform.hpp:


include/CL/sycl/platform/detail/platform.hpp
CL/sycl/platform.hpp
CL/sycl/platform/detail/host_platform.hpp
CL/sycl/platform/detail/opencl_platform.hpp
CL/sycl/device.hpp
CL/sycl/info/platform.hpp
CL/sycl/device_selector.hpp
CL/sycl/device/detail/host_device.hpp
CL/sycl/info/device.hpp
algorithm
CL/sycl/device/detail/opencl_device.hpp
CL/sycl/detail/global_config.hpp
CL/sycl/detail/singleton.hpp
CL/sycl/detail/shared_ptr_implementation.hpp
CL/sycl/detail/unimplemented.hpp
CL/sycl/detail/cache.hpp
CL/sycl/exception.hpp
CL/sycl/info/param_traits.hpp
CL/sycl/device/detail/device.hpp
boost/compute.hpp
CL/cl.h
boost/core/null_deleter.hpp
boost/operators.hpp
iostream
unordered_map
exception
CL/sycl/detail/default_classes.hpp
memory
functional
mutex
vector
string

This graph shows which files directly or indirectly include this file:


include/CL/sycl/platform/detail/platform.hpp
include/CL/sycl/platform/detail/host_platform.hpp
include/CL/sycl/platform.hpp
include/CL/sycl/device/detail/device.hpp
include/CL/sycl/device/detail/opencl_device.hpp
include/CL/sycl/device/detail/host_device.hpp
include/CL/sycl/device.hpp
include/CL/sycl/device_selector.hpp
include/CL/sycl/platform/detail/opencl_platform.hpp
include/CL/sycl/context.hpp
include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::detail::platform

  An abstract class representing various models of SYCL platforms. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.84 platform.hpp

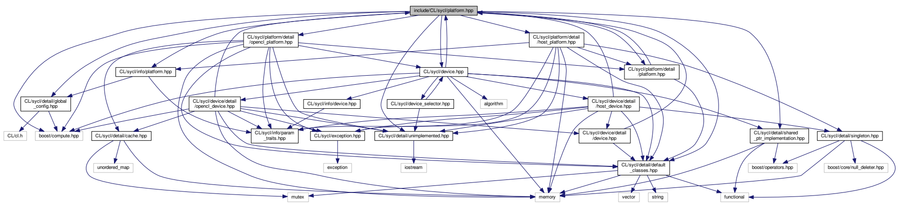
```
#ifndef TRISYCL_SYCL_PLATFORM_DETAIL_PLATFORM_HPP
#define TRISYCL_SYCL_PLATFORM_DETAIL_PLATFORM_HPP

/** \file The OpenCL SYCL abstract platform

```

```
    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/platform.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/// An abstract class representing various models of SYCL platforms
class platform {

public:

#ifdef TRISYCL_OPENCL
  /// Return the cl_platform_id of the underlying OpenCL platform
  virtual cl_platform_id get() const = 0;
#endif


  /// Return true if the platform is a SYCL host platform
  virtual bool is_host() const = 0;


  /// Query the platform for OpenCL string info::platform info
  virtual string_class get_info_string(info::platform param) const
 = 0;


  /// Specify whether a specific extension is supported on the platform.
  virtual bool has_extension(const string_class &extension) const = 0;


  // Virtual to call the real destructor
  virtual ~platform() {}

};

/// @} to end the execution Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PLATFORM_DETAIL_PLATFORM_HPP
```

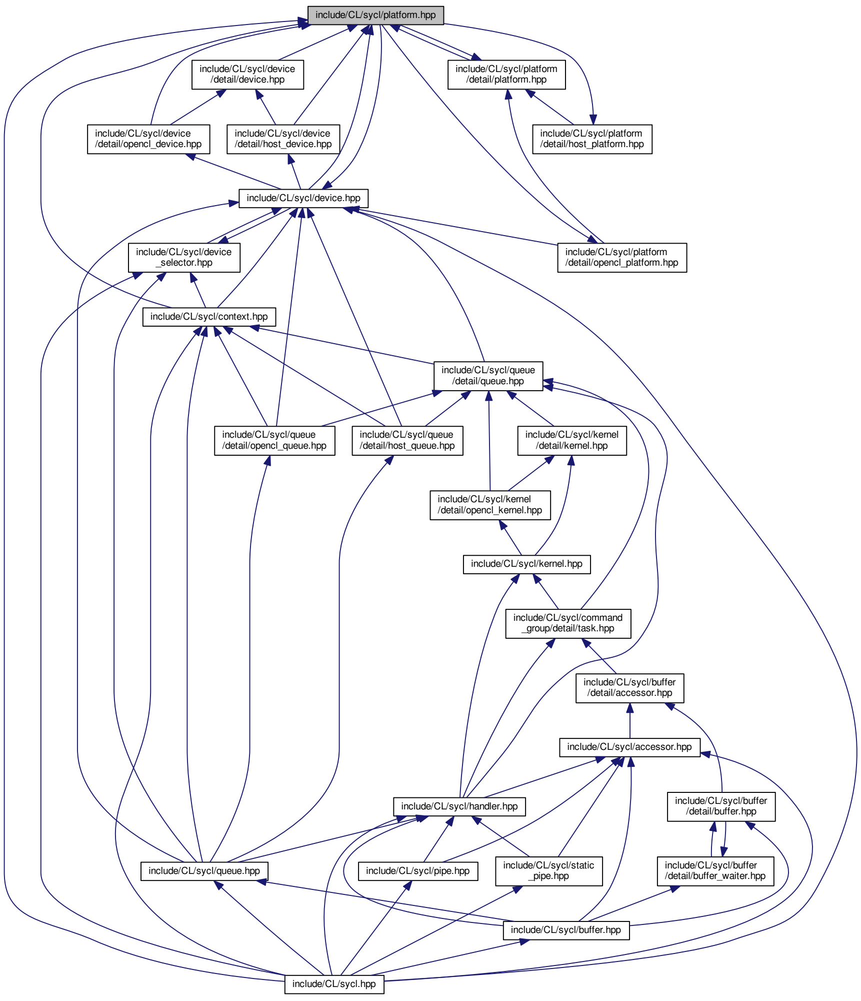
11.85 include/CL/sycl/platform.hpp File Reference

```
#include <boost/compute.hpp>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/platform/detail/host_platform.hpp"
#include "CL/sycl/platform/detail/opencl_platform.hpp"
#include "CL/sycl/platform/detail/platform.hpp"
#include "CL/sycl/info/platform.hpp"
```

Include dependency graph for platform.hpp:


include/CL/sycl/platform.hpp
CL/sycl/platform/detail/opencl_platform.hpp
CL/sycl/platform/detail/host_platform.hpp
CL/sycl/info/platform.hpp
CL/sycl/device.hpp
CL/sycl/platform/detail/platform.hpp
CL/sycl/detail/global_config.hpp
CL/sycl/device/detail/opencl_device.hpp
CL/sycl/info/device.hpp
CL/sycl/device_selector.hpp
algorithm
CL/sycl/device/detail/host_device.hpp
CL/cl.h
boost/compute.hpp
CL/sycl/detail/cache.hpp
CL/sycl/info/param_traits.hpp
CL/sycl/exception.hpp
CL/sycl/detail/unimplemented.hpp
CL/sycl/device/detail/device.hpp
CL/sycl/detail/shared_ptr_implementation.hpp
CL/sycl/detail/singleton.hpp
unordered_map
exception
iostream
CL/sycl/detail/default_classes.hpp
boost/operators.hpp
boost/core/null_deleter.hpp
mutex
memory
vector
string
functional

This graph shows which files directly or indirectly include this file:


include/CL/sycl/platform.hpp
include/CL/sycl/device/detail/device.hpp
include/CL/sycl/platform/detail/platform.hpp
include/CL/sycl/device/detail/opencl_device.hpp
include/CL/sycl/device/detail/host_device.hpp
include/CL/sycl/platform/detail/host_platform.hpp
include/CL/sycl/device.hpp
include/CL/sycl/device_selector.hpp
include/CL/sycl/platform/detail/opencl_platform.hpp
include/CL/sycl/context.hpp
include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::platform

  Abstract the OpenCL platform. More...

- struct std::hash< cl::sycl::platform >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- std

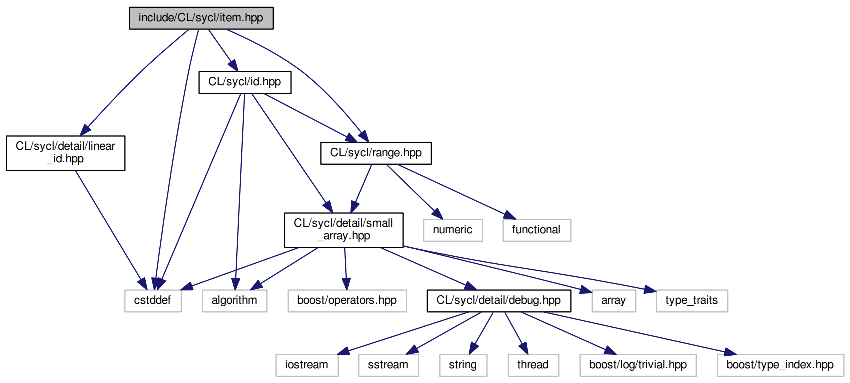
11.86 platform.hpp

```
#ifndef TRISYCL_SYCL_PLATFORM_HPP
#define TRISYCL_SYCL_PLATFORM_HPP

/** \file The OpenCL SYCL platform

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/global_config.hpp"

#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/platform/detail/host_platform.hpp"
#ifdef TRISYCL_OPENCL
#include "CL/sycl/platform/detail/opencl_platform.hpp"
#endif
#include "CL/sycl/platform/detail/platform.hpp"
#include "CL/sycl/info/platform.hpp"

namespace cl {
namespace sycl {

class device_selector;
class device;

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/** Abstract the OpenCL platform

    \todo triSYCL Implementation
*/
class platform
  /* Use the underlying platform implementation that can be shared in the
     SYCL model */
  : public detail::shared_ptr_implementation<platform, detail::platform> {

  // The type encapsulating the implementation
  using implementation_t =
    detail::shared_ptr_implementation<platform, detail::platform>
;

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;


public:

  /** Default constructor for platform which is the host platform

      Returns errors via the SYCL exception class.
  */
  platform() : implementation_t {
detail::host_platform::instance() } {}


#ifdef TRISYCL_OPENCL
  /** Construct a platform class instance using cl_platform_id of the
      OpenCL device

      Return synchronous errors via the SYCL exception class.

      Retain a reference to the OpenCL platform.
  */
  platform(cl_platform_id platform_id)
    : platform { boost::compute::platform { platform_id } } {}


  /** Construct a platform class instance using a boost::compute::platform

      This is a triSYCL extension for boost::compute interoperation.

      Return synchronous errors via the SYCL exception class.
  */
  platform(const boost::compute::platform &p)
    : implementation_t { detail::opencl_platform::instance (p) } {}
#endif


  /**  Construct a platform object from the device selected by a device
       selector of the user's choice

       Returns errors via the SYCL exception class.
  */
  explicit platform(const device_selector &dev_selector) {
    detail::unimplemented();
  }


#ifdef TRISYCL_OPENCL
  /** Returns the cl_platform_id of the underlying OpenCL platform

      If the platform is not a valid OpenCL platform, for example if it is
      the SYCL host, an exception is thrown

      \todo Define a SYCL exception for this
  */
  cl_platform_id get() const {
    return implementation->get();
  }
#endif


  /// Get the list of all the platforms available to the application
  static vector_class<platform> get_platforms() {
    // Start with the default platform
    vector_class<platform> platforms { {} };

#ifdef TRISYCL_OPENCL
    // Then add all the OpenCL platforms
    for (const auto &d : boost::compute::system::platforms())
      platforms.emplace_back(d);
#endif

    return platforms;
  }

#if 0
  /** Returns all the available devices for this platform, of type device
      type, which is defaulted to info::device_type::all

      By default returns all the devices.

      \todo To be implemented
  */
  vector_class<device>
  get_devices(info::device_type device_type =
info::device_type::all) const {
    detail::unimplemented();
    return {};
  }
#endif


  /** Get the OpenCL information about the requested parameter

      \todo Add to the specification
  */
  template <typename ReturnT>
  ReturnT get_info(info::platform param) const {
    // Only strings are needed here
    return implementation->get_info_string(param);
  }


  /// Get the OpenCL information about the requested template parameter
  template <info::platform Param>
  typename info::param_traits<info::platform, Param>::type
  get_info() const {
    /* Forward to the implementation without using template parameter
       but with a parameter instead, since it is incompatible with
       virtual function and because fortunately only strings are
       needed here */
    return get_info<typename info::param_traits<
info::platform,
                                                  Param>::type>(Param);
  }


  /// Test if an extension is available on the platform
  bool has_extension(const string_class &extension) const {
    return implementation->has_extension(extension);
  }


  /// Test if this platform is a host platform
  bool is_host() const {
    return implementation->is_host();
  }

};

/// @} to end the execution Doxygen group

}
}


/* Inject a custom specialization of std::hash to have the buffer
   usable into an unordered associative container

   \todo Add this to the spec
*/
namespace std {

template <> struct hash<cl::sycl::platform> {

  auto operator()(const cl::sycl::platform &p) const {
    // Forward the hashing to the implementation
    return p.hash();
  }

};

}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PLATFORM_HPP
```

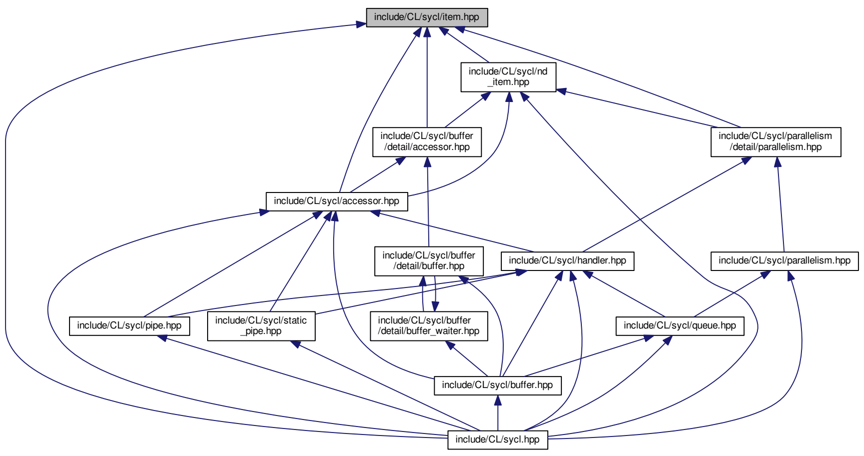
11.87 include/CL/sycl/item.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for item.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/item.hpp
include/CL/sycl/nd_item.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/parallelism/detail/parallelism.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/parallelism.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::item< dims >

  A SYCL item stores information on a work-item with some more context such as the definition range and offset. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.88 item.hpp

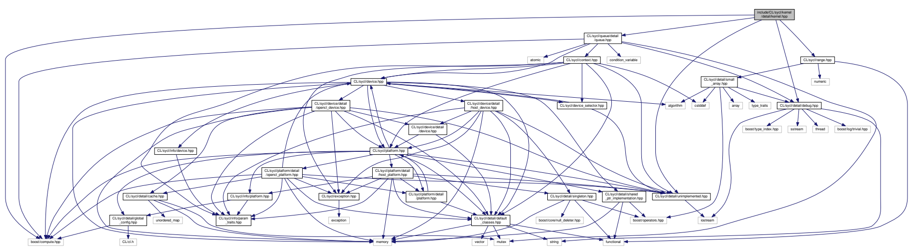
```
#ifndef TRISYCL_SYCL_ITEM_HPP
#define TRISYCL_SYCL_ITEM_HPP

/** \file The OpenCL SYCL item<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>

#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/range.hpp"

namespace cl {
namespace sycl {

/** \addtogroup parallelism Expressing parallelism through kernels
    @{
*/

/** A SYCL item stores information on a work-item with some more context
    such as the definition range and offset.
*/
template <std::size_t dims = 1>
class item {

public:

  /// \todo add this Boost::multi_array or STL concept to the
  /// specification?
  static constexpr auto dimensionality = dims;

private:

  range<dims> global_range;
  id<dims> global_index;
  id<dims> offset;

public:

  /** Create an item from a local size and an optional offset

      This constructor is used by the triSYCL implementation and the
      non-regression testing.
  */
  item(range<dims> global_size,
       id<dims> global_index,
       id<dims> offset = {}) :
    global_range { global_size },
    global_index { global_index },
    offset { offset }
  {}


  /** To be able to copy and assign item, use default constructors too

      \todo Make most of them protected, reserved to implementation
  */
  item() = default;


  /** Return the constituent local or global id<> representing the
      work-item's position in the iteration space
  */
  id<dims> get() const { return global_index; }


  /** Return the requested dimension of the constituent id<> representing
      the work-item's position in the iteration space
  */
  size_t get(int dimension) const { return get()[dimension]; }


  /** Return the constituent id<> l-value representing the work-item's
      position in the iteration space in the given dimension
  */
  auto &operator[](int dimension) { return global_index[dimension]; }


  /** Returns a range<> representing the dimensions of the range of
```

```
      possible values of the item
  */
  range<dims> get_range() const { return global_range; }


  /** Returns an id<> representing the n-dimensional offset provided to
      the parallel_for and that is added by the runtime to the global-ID
      of each work-item, if this item represents a global range

      For an item representing a local range of where no offset was passed
      this will always return an id of all 0 values.
  */
  id<dims> get_offset() const { return offset; }


  /** Return the linearized ID in the item's range

      Computed as the flatted ID after the offset is subtracted.
  */
  size_t get_linear_id() const {
    return detail::linear_id(get_range(), get(),
get_offset());
  }


  /** For the implementation, need to set the global index

      \todo Move to private and add friends
  */
  void set(id<dims> Index) { global_index = Index; }


  /// Display the value for debugging and validation purpose
  void display() const {
    global_range.display();
    global_index.display();
    offset.display();
  }

};

/// @} End the parallelism Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ITEM_HPP
```

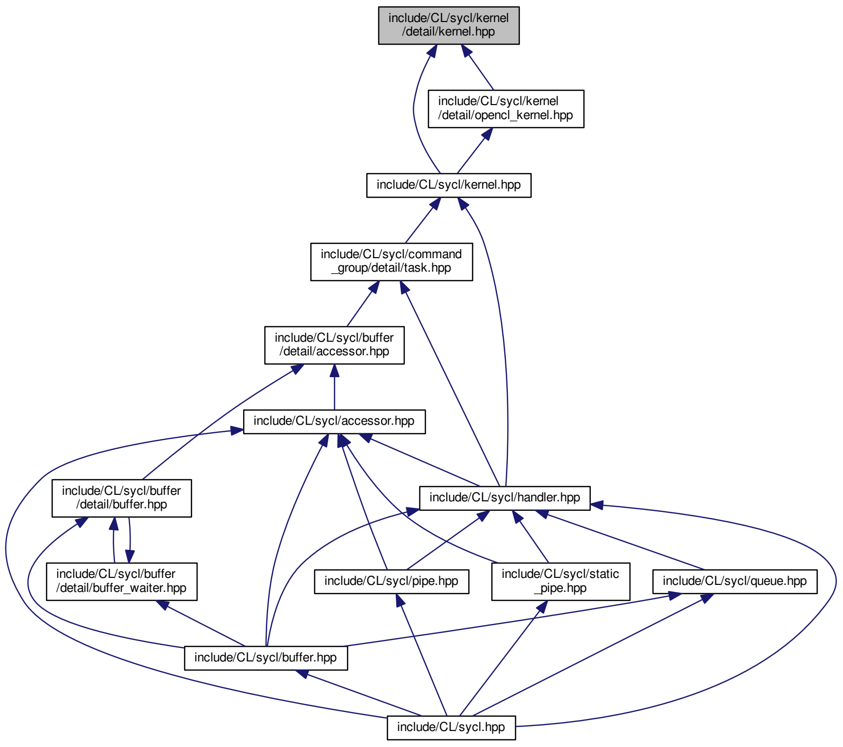
11.89 include/CL/sycl/kernel/detail/kernel.hpp File Reference

```
#include <boost/compute.hpp>
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/queue/detail/queue.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for kernel.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class cl::sycl::detail::kernel

  Abstract SYCL kernel. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Macros

- #define TRISYCL_ParallelForKernel_RANGE(N)

  Launch a kernel with a range<>

11.89.1 Macro Definition Documentation

11.89.1.1 #define TRISYCL_ParallelForKernel_RANGE(*N*)

Value:

```
virtual void parallel_for(std::shared_ptr<detail::task> task, std::shared_ptr<detail::queue> q,
           \
                            const range<N> &num_work_items) = 0;
```

Launch a kernel with a range<>

Do not use a template since it does not work with virtual functions

Todo Think to a cleaner solution

Definition at line 58 of file kernel.hpp.

11.90 kernel.hpp

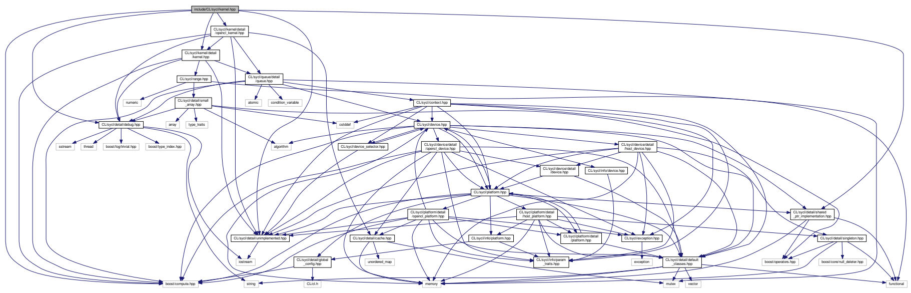
```
#ifndef TRISYCL_SYCL_KERNEL_DETAIL_KERNEL_HPP
#define TRISYCL_SYCL_KERNEL_DETAIL_KERNEL_HPP

/** \file The OpenCL SYCL kernel

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
//#include "CL/sycl/info/kernel.hpp"
#include "CL/sycl/queue/detail/queue.hpp"
#include "CL/sycl/range.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/// Abstract SYCL kernel
class kernel : detail::debug<detail::kernel> {

 public:

#ifdef TRISYCL_OPENCL
  /** Return the OpenCL kernel object for this kernel

      Retains a reference to the returned cl_kernel object. Caller
      should release it when finished.
  */
  virtual cl_kernel get() const = 0;


  /** Return the Boost.Compute OpenCL kernel object for this kernel

      This is an extension.
  */
  virtual boost::compute::kernel get_boost_compute() const = 0;
#endif
```

```


  /** Launch a kernel with a range<>

      Do not use a template since it does not work with virtual functions

      \todo Think to a cleaner solution
  */
#define TRISYCL_ParallelForKernel_RANGE(N)                                 \
  virtual void parallel_for(std::shared_ptr<detail::task> task, std::shared_ptr<detail::queue> q,      \
                            const range<N> &num_work_items) = 0;

  TRISYCL_ParallelForKernel_RANGE(1)
  TRISYCL_ParallelForKernel_RANGE(2)
  TRISYCL_ParallelForKernel_RANGE(3)
#undef TRISYCL_ParallelForKernel_RANGE


  /// Return the context that this kernel is defined for
  //virtual context get_context() const;

  /// Return the program that this kernel is part of
  //virtual program get_program() const;

  // Virtual to call the real destructor
  virtual ~kernel() {}

};

/// @} End the execution Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_KERNEL_KERNEL_HPP
```

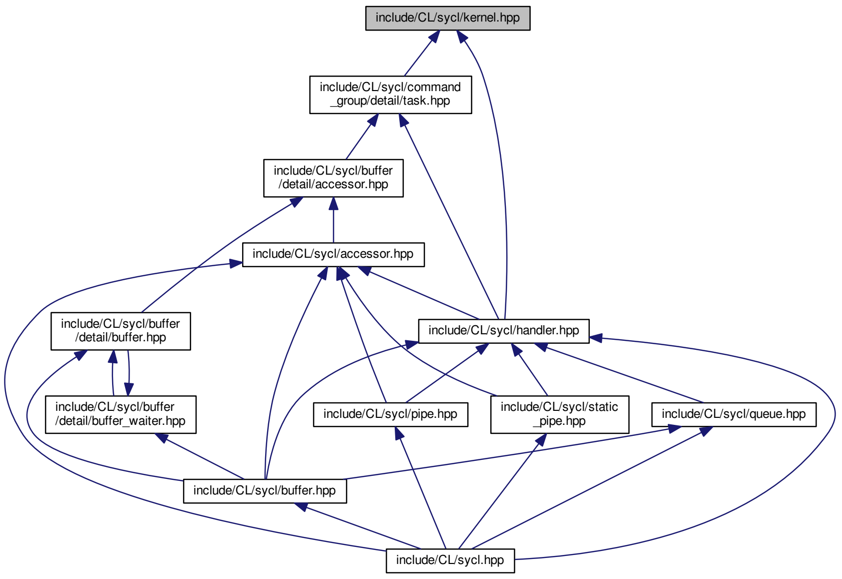
11.91 include/CL/sycl/kernel.hpp File Reference

```
#include <boost/compute.hpp>
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/kernel/detail/kernel.hpp"
#include "CL/sycl/kernel/detail/opencl_kernel.hpp"
```

Include dependency graph for kernel.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::kernel

    SYCL kernel. More...
- struct std::hash< cl::sycl::kernel >

Namespaces

- cl

    The vector type to be used as SYCL vector.
- cl::sycl
- std

11.92 kernel.hpp

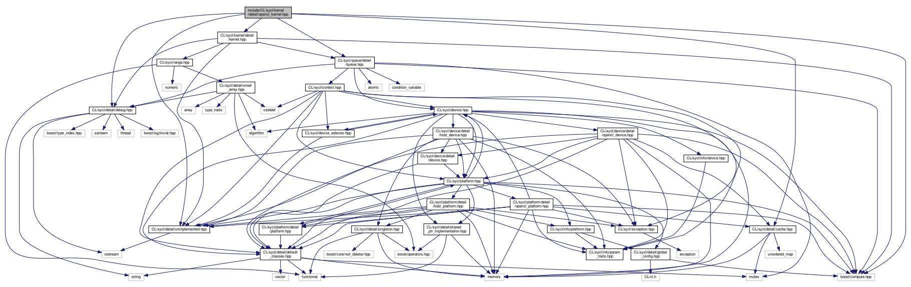
```
#ifndef TRISYCL_SYCL_KERNEL_HPP
#define TRISYCL_SYCL_KERNEL_HPP

/** \file The OpenCL SYCL kernel

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
//#include "CL/sycl/info/kernel.hpp"
```

```
#include "CL/sycl/kernel/detail/kernel.hpp"
#ifdef TRISYCL_OPENCL
#include "CL/sycl/kernel/detail/opencl_kernel.hpp"
#endif

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/** SYCL kernel

    \todo To be implemented

    \todo Check specification
*/
class kernel
  /* Use the underlying kernel implementation that can be shared in
     the SYCL model */
  : public detail::shared_ptr_implementation<kernel, detail::kernel> {

  // The type encapsulating the implementation
  using implementation_t =
    detail::shared_ptr_implementation<kernel, detail::kernel>
;

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

  // The handler class uses the implementation
  friend class handler;

 public:

  /** The default object is not valid because there is no program or
      \code cl_kernel \endcode associated with it */
  kernel() = delete;

#ifdef TRISYCL_OPENCL
  /** Constructor for SYCL kernel class given an OpenCL kernel object
      with set arguments, valid for enqueuing

      Retains a reference to the \p cl_kernel object. The Caller
      should release the passed cl_kernel object when it is no longer
      needed.
  */
  kernel(cl_kernel k) : kernel { boost::compute::kernel { k } } {}


  /** Construct a kernel class instance using a boost::compute::kernel

      This is a triSYCL extension for boost::compute interoperation.

      Return synchronous errors via the SYCL exception class.
  */
  kernel(const boost::compute::kernel &k)
    : implementation_t { detail::opencl_kernel::instance(k)
} {}


  /** Return the OpenCL kernel object for this kernel

      Retains a reference to the returned cl_kernel object. Caller
      should release it when finished.
  */
  cl_kernel get() const {
    return implementation->get();
  }
#endif


#if 0
  /// Return the context that this kernel is defined for
  //context get_context() const;

  /// Return the program that this kernel is part of
  //program get_program() const;

  /** Query information from the kernel object using the
      info::kernel_info descriptor.
  */
  template <info::kernel param>
  typename info::param_traits<info::kernel, param>::type
    get_info() const {
    detail::unimplemented();
  }
#endif

};

/// @} End the execution Doxygen group

}
}


/* Inject a custom specialization of std::hash to have the buffer
   usable into an unordered associative container

   \todo Add this to the spec
*/
namespace std {

template <> struct hash<cl::sycl::kernel> {

  auto operator()(const cl::sycl::kernel &k) const {
    // Forward the hashing to the implementation
    return k.hash();
  }

};

}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_KERNEL_HPP
```

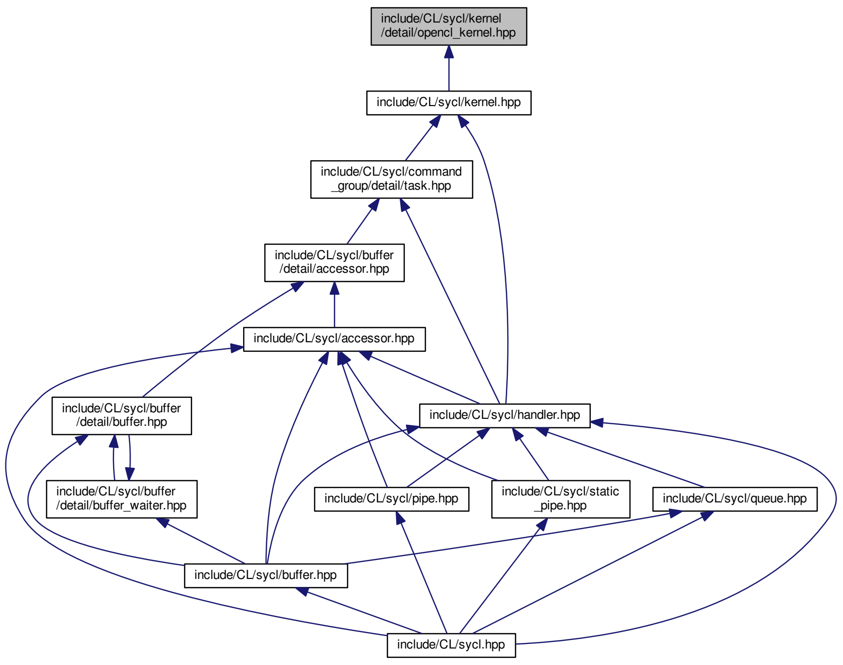
11.93 include/CL/sycl/kernel/detail/opencl_kernel.hpp File Reference

```
#include <boost/compute.hpp>
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/kernel/detail/kernel.hpp"
#include "CL/sycl/queue/detail/queue.hpp"
```

Include dependency graph for opencl_kernel.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/kernel /detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command _group/detail/task.hpp
include/CL/sycl/buffer /detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer /detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer /detail/buffer_waiter.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static _pipe.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::detail::opencl_kernel

    An abstraction of the OpenCL kernel.

Namespaces

- cl

    The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Macros

- #define TRISYCL_ParallelForKernel_RANGE(N)

    Launch an OpenCL kernel with a range<>

Functions

- detail::cache< cl_kernel, detail::opencl_kernel > opencl_kernel::cache cl::sycl::detail::__attribute__ ((weak))

11.93.1 Macro Definition Documentation

11.93.1.1 #define TRISYCL_ParallelForKernel_RANGE(*N*)

Value:

```
void parallel_for(std::shared_ptr<detail::task> task,\
  std::shared_ptr<detail::queue> q,                                  \
                    const range<N> &num_work_items) override {       \
    static_assert(sizeof(range<N>::value_type) == sizeof(size_t),    \
                  "num_work_items::value_type compatible with "      \
                  "Boost.Compute");                                  \
    q->get_boost_compute().enqueue_nd_range_kernel                   \
      (k,                                                            \
       static_cast<size_t>(N),                                       \
       NULL,                                                         \
       static_cast<const size_t *>(num_work_items.data()),           \
       NULL);                                                        \
    /* For now use a crude synchronization mechanism to map directly a  \
       host task to an accelerator task */                           \
    q->get_boost_compute().finish();                                 \
  };
```

Launch an OpenCL kernel with a range<>

Do not use a template since it does not work with virtual functions

Todo Think to a cleaner solution

Definition at line 92 of file opencl_kernel.hpp.

11.94 opencl_kernel.hpp

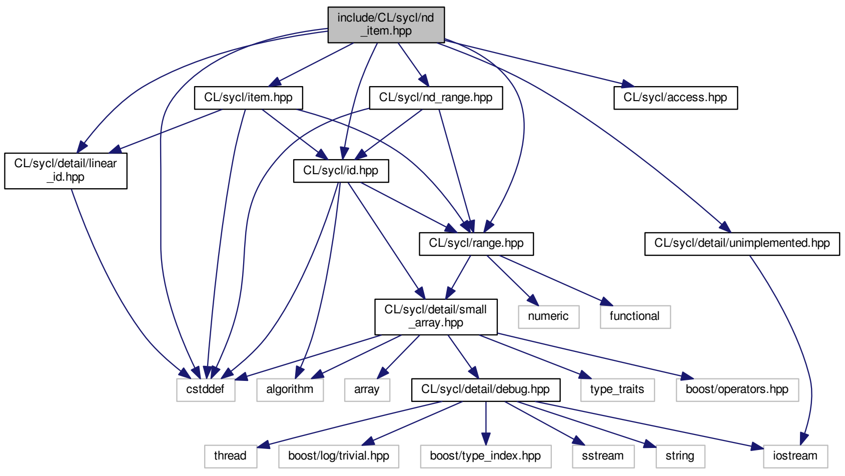
```
#ifndef TRISYCL_SYCL_KERNEL_DETAIL_OPENCL_KERNEL_HPP
#define TRISYCL_SYCL_KERNEL_DETAIL_OPENCL_KERNEL_HPP

/** \file The OpenCL SYCL kernel

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
//#include "CL/sycl/info/kernel.hpp"
#include "CL/sycl/kernel/detail/kernel.hpp"
#include "CL/sycl/queue/detail/queue.hpp"

namespace cl {
namespace sycl {
namespace detail {

/// An abstraction of the OpenCL kernel
class opencl_kernel : public detail::kernel,
                      detail::debug<opencl_kernel> {

  /// Use the Boost Compute abstraction of the OpenCL kernel
  boost::compute::kernel k;

  /** A cache to always return the same alive kernel for a given
      OpenCL kernel

      C++11 guaranties the static construction is thread-safe
  */
  static detail::cache<cl_kernel, detail::opencl_kernel>
cache;

  opencl_kernel(const boost::compute::kernel &k) : k { k } {}

 public:

  ///// Get a singleton instance of the opencl_device
  static std::shared_ptr<opencl_kernel>
  instance(const boost::compute::kernel &k) {
    return cache.get_or_register(k.get(),
                                 [&] { return new opencl_kernel { k }; });
  }

  /** Return the underlying OpenCL object

      \todo Improve the spec to deprecate C OpenCL host API and move
      to C++ instead to avoid this ugly ownership management
  */
  cl_kernel get() const override {
    /// \todo Test error and throw. Externalize this feature in Boost.Compute?
    clRetainKernel(k);
    return k.get();
  }


  /** Return the Boost.Compute OpenCL kernel object for this kernel

      This is an extension.
  */
  boost::compute::kernel get_boost_compute() const override {
    return k;
  }


  //context get_context() const override

  //program get_program() const override

#if 0
  template <info::kernel param>
  typename info::param_traits<info::kernel, param>::type
  get_info() const {
    detail::unimplemented();
  }
#endif


  /** Launch an OpenCL kernel with a range<>

      Do not use a template since it does not work with virtual functions

      \todo Think to a cleaner solution
  */
#define TRISYCL_ParallelForKernel_RANGE(N)                               \
  void parallel_for(std::shared_ptr<detail::task> task,\
  std::shared_ptr<detail::queue> q,                                      \
                    const range<N> &num_work_items) override {           \
    static_assert(sizeof(range<N>::value_type) == sizeof(size_t),        \
                  "num_work_items::value_type compatible with "          \
                  "Boost.Compute");                                      \
    q->get_boost_compute().enqueue_nd_range_kernel                       \
      (k,                                                                \
       static_cast<size_t>(N),                                           \
       NULL,                                                             \
       static_cast<const size_t *>(num_work_items.data()),               \
       NULL);                                                            \
    /* For now use a crude synchronization mechanism to map directly a  \
       host task to an accelerator task */                               \
    q->get_boost_compute().finish();                                     \
  };

  TRISYCL_ParallelForKernel_RANGE(1)
  TRISYCL_ParallelForKernel_RANGE(2)
  TRISYCL_ParallelForKernel_RANGE(3)
#undef TRISYCL_ParallelForKernel_RANGE


  /// Unregister from the cache on destruction
  ~opencl_kernel() override {
    cache.remove(k.get());
  }

};

```

```
/* Allocate the cache here but since this is a pure-header library,
   use a weak symbol so that only one remains when SYCL headers are
   used in different compilation units of a program
*/
detail::cache<cl_kernel, detail::opencl_kernel>
opencl_kernel::cache
  __attribute__((weak));

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_KERNEL_DETAIL_OPENCL_KERNEL_HPP
```

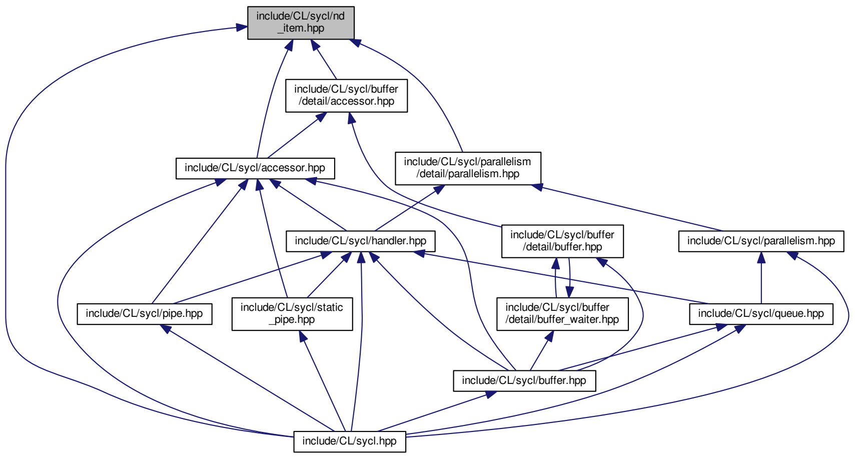
11.95 include/CL/sycl/nd_item.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/access.hpp"
#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for nd_item.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/nd_item.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/parallelism/detail/parallelism.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/parallelism.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

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

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

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.96 nd_item.hpp

```
#ifndef TRISYCL_SYCL_ND_ITEM_HPP
#define TRISYCL_SYCL_ND_ITEM_HPP

/** \file The OpenCL SYCL nd_item<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>

#include "CL/sycl/access.hpp"
#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"

namespace cl {
namespace sycl {

/** \addtogroup parallelism Expressing parallelism through kernels
    @{
*/
```

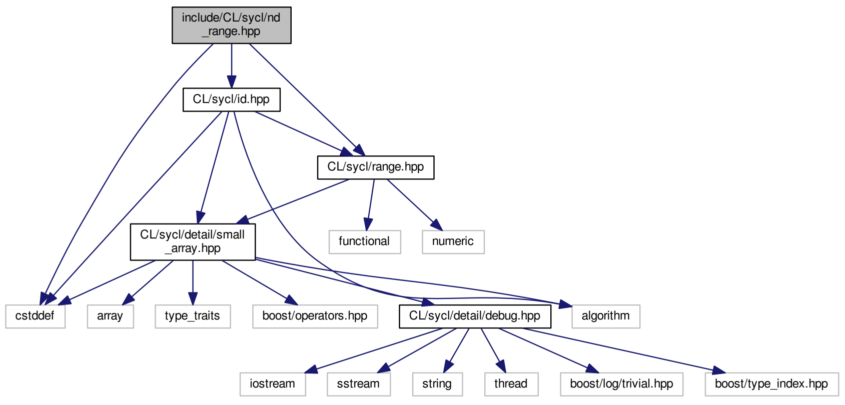
```

/** A SYCL nd_item stores information on a work-item within a work-group,
    with some more context such as the definition ranges.
*/
template <std::size_t dims = 1>
struct nd_item {
  /// \todo add this Boost::multi_array or STL concept to the
  /// specification?
  static constexpr auto dimensionality = dims;

private:

  id<dims> global_index;
  /* This is a cached value since it can be computed from global_index and
     ND_range */
  id<dims> local_index;
  nd_range<dims> ND_range;

public:

  /** Create an empty nd_item<> from an nd_range<>

      \todo This is for the triSYCL implementation which is expected to
      call set_global() and set_local() later. This should be hidden to
      the user.
  */
  nd_item(nd_range<dims> ndr) : ND_range { ndr } {}


  /** Create a full nd_item

      \todo This is for validation purpose. Hide this to the programmer
      somehow
  */
  nd_item(id<dims> global_index,
          nd_range<dims> ndr) :
    global_index { global_index },
    // Compute the local index using the offset and the group size
    local_index { (global_index - ndr.get_offset())%id<dims> { ndr.get_local() } },
    ND_range { ndr }
  {}


  /** To be able to copy and assign nd_item, use default constructors too

      \todo Make most of them protected, reserved to implementation
  */
  nd_item() = default;


  /** Return the constituent global id representing the work-item's
      position in the global iteration space
  */
  id<dims> get_global() const { return global_index; }


  /** Return the constituent element of the global id representing the
      work-item's position in the global iteration space in the given
      dimension
  */
  size_t get_global(int dimension) const { return get_global()[dimension]; }


  /** Return the flattened id of the current work-item after subtracting
      the offset
  */
  size_t get_global_linear_id() const {
    return detail::linear_id(get_global_range(),
get_global(), get_offset());
  }


  /** Return the constituent local id representing the work-item's
      position within the current work-group
  */
  id<dims> get_local() const { return local_index; }


  /** Return the constituent element of the local id representing the
      work-item's position within the current work-group in the given
      dimension
  */
  size_t get_local(int dimension) const { return get_local()[dimension]; }


  /** Return the flattened id of the current work-item within the current
      work-group
```

```
   */
  size_t get_local_linear_id() const {
    return detail::linear_id(get_local_range(), get_local());
  }


  /** Return the constituent group group representing the work-group's
      position within the overall nd_range
  */
  id<dims> get_group() const {
    /* Convert get_local_range() to an id<> to remove ambiguity into using
       implicit conversion either from range<> to id<> or the opposite */
    return get_global()/id<dims> { get_local_range() };
  }


  /** Return the constituent element of the group id representing the
      work-group;s position within the overall nd_range in the given
      dimension.
  */
  size_t get_group(int dimension) const {
    return get_group()[dimension];
  }


  /// Return the flattened id of the current work-group
  size_t get_group_linear_id() const {
    return detail::linear_id(get_num_groups(), get_group());
  }


  /// Return the number of groups in the nd_range
  id<dims> get_num_groups() const {
    return get_nd_range().get_group();
  }

  /// Return the number of groups for dimension in the nd_range
  size_t get_num_groups(int dimension) const {
     return get_num_groups()[dimension];
  }


  /// Return a range<> representing the dimensions of the nd_range<>
  range<dims> get_global_range() const {
    return get_nd_range().get_global();
  }


  /// Return a range<> representing the dimensions of the current work-group
  range<dims> get_local_range() const {
    return get_nd_range().get_local();
  }


  /** Return an id<> representing the n-dimensional offset provided to the
      constructor of the nd_range<> and that is added by the runtime to the
      global-ID of each work-item
  */
  id<dims> get_offset() const { return get_nd_range().get_offset(); }


  /// Return the nd_range<> of the current execution
  nd_range<dims> get_nd_range() const { return ND_range; }


  /** Allows projection down to an item

      \todo Add to the specification
  */
  item<dims> get_item() const {
    return { get_global_range(), get_global(), get_offset() };
  }


  /** Execute a barrier with memory ordering on the local address space,
      global address space or both based on the value of flag

      The current work-item will wait at the barrier until all work-items
      in the current work-group have reached the barrier.

      In addition, the barrier performs a fence operation ensuring that all
      memory accesses in the specified address space issued before the
      barrier complete before those issued after the barrier
  */
  void barrier(access::fence_space flag =
               access::fence_space::global_and_local) const {
#if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
    /* Use OpenMP barrier in the implementation with 1 OpenMP thread per
       work-item of the work-group */
#pragma omp barrier
#else
    // \todo To be implemented efficiently otherwise
    detail::unimplemented();
#endif
  }


  // For the triSYCL implementation, need to set the local index
  void set_local(id<dims> Index) { local_index = Index; }


  // For the triSYCL implementation, need to set the global index
  void set_global(id<dims> Index) { global_index = Index; }

};

/// @} End the parallelism Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ND_ITEM_HPP
```

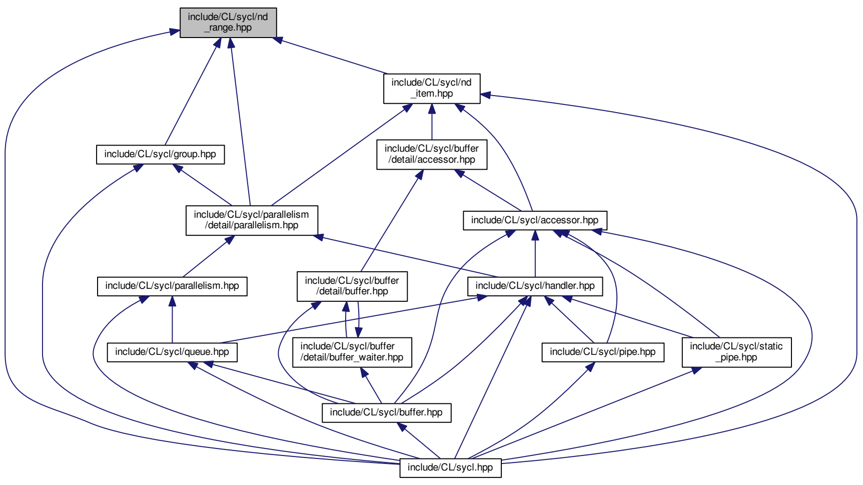
11.97 include/CL/sycl/nd_range.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/id.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for nd_range.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/nd_range.hpp
include/CL/sycl/nd_item.hpp
include/CL/sycl/group.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/parallelism/detail/parallelism.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/parallelism.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- 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...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.98 nd_range.hpp

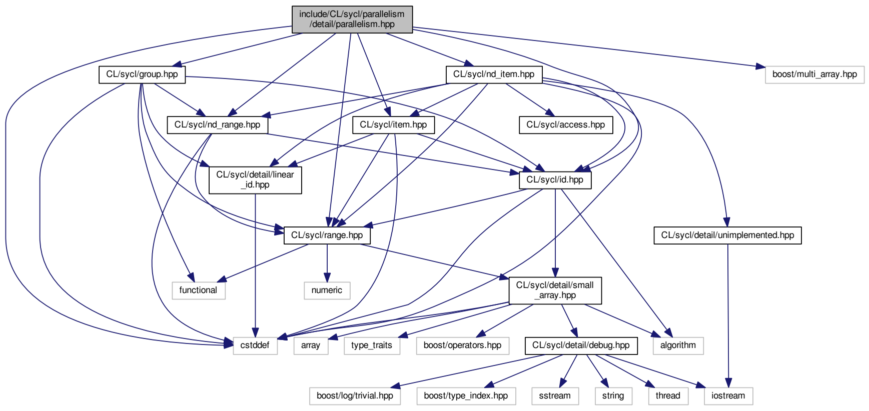
```
#ifndef TRISYCL_SYCL_ND_RANGE_HPP
#define TRISYCL_SYCL_ND_RANGE_HPP

/** \file The OpenCL SYCL nd_range<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>

#include "CL/sycl/id.hpp"
#include "CL/sycl/range.hpp"

namespace cl {
namespace sycl {

/** \addtogroup parallelism Expressing parallelism through kernels
    @{
*/

/** 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
*/
template <std::size_t dims = 1>
struct nd_range {
  /// \todo add this Boost::multi_array or STL concept to the
  /// specification?
  static constexpr auto dimensionality = dims;

private:

  range<dimensionality> global_range;
  range<dimensionality> local_range;
  id<dimensionality> offset;

public:

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

      By default use a zero offset, that is iterations start at 0
   */
  nd_range(range<dims> global_size,
           range<dims> local_size,
           id<dims> offset = {}) :
    global_range { global_size }, local_range { local_size }, offset { offset }
  { }


  /// Get the global iteration space range
  range<dims> get_global() const { return global_range; }


  /// Get the local part of the iteration space range
  range<dims> get_local() const { return local_range; }


  /// Get the range of work-groups needed to run this ND-range
  auto get_group() const {
    /* This is basically global_range/local_range, round up to the
       next integer, in case the global eange is not a multiple of the
       local range. Note this is a motivating example to build a range
       from a scalar with a broadcasting constructor. */
    return (global_range + local_range - range<dims>{ 1 })/local_range;
  }


  /// \todo get_offset() is lacking in the specification
  id<dims> get_offset() const { return offset; }


  /// Display the value for debugging and validation purpose
  void display() const {
    global_range.display();
    local_range.display();
    offset.display();
  }

};

/// @} End the parallelism Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ND_RANGE_HPP
```

11.99 include/CL/sycl/parallelism/detail/parallelism.hpp File Reference

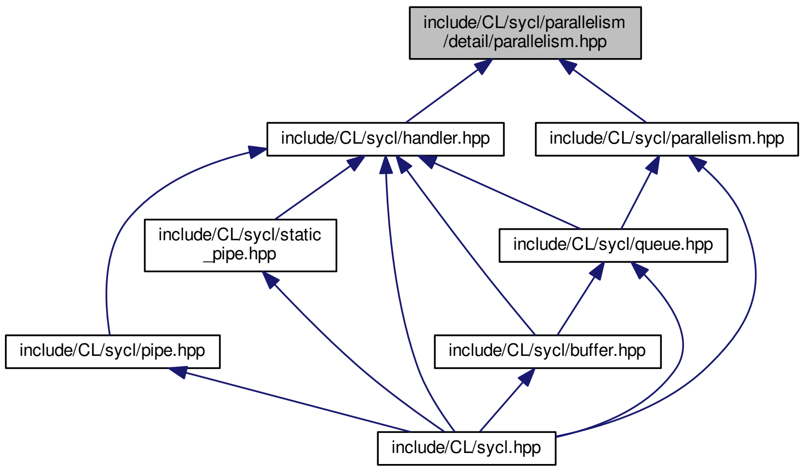
Implement the detail of the parallel constructions to launch kernels.

```
#include <cstddef>
#include <boost/multi_array.hpp>
#include "CL/sycl/group.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for parallelism.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/parallelism/detail/parallelism.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/parallelism.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- struct cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id >

  A recursive multi-dimensional iterator that ends up calling f. More...
- struct cl::sycl::detail::parallel_OpenMP_for_iterate< level, Range, ParallelForFunctor, Id >

  A top-level recursive multi-dimensional iterator variant using OpenMP. More...
- struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >

  Stop the recursion when level reaches 0 by simply calling the kernel functor with the constructed id. More...

Namespaces

- cl

    The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template< std::size_t Dimensions = 1, typename ParallelForFunctor , typename Id >
  void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFunctor f, Id)

    Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template< std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFunctor f, item< Dimensions >)

    Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template< std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFunctor f)

    Calls the appropriate ternary parallel_for overload based on the index type of the kernel function object f.
- template< std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_global_offset (range< Dimensions > global_size, id< Dimensions > offset, ParallelForFunctor f)

    Implementation of parallel_for with a range<> and an offset.
- template< std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

    Implement a variation of parallel_for to take into account a nd_range<>
- template< std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFunctor f)

    Implement the loop on the work-groups.
- template< std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_workitem (const group< Dimensions > &g, ParallelForFunctor f)

    Implement the loop on the work-items inside a work-group.

11.99.1 Detailed Description

Implement the detail of the parallel constructions to launch kernels.

Ronan at keryell dot FR

Definition in file parallelism.hpp.

11.100 parallelism.hpp

```
#ifndef TRISYCL_SYCL_PARALLELISM_DETAIL_PARALLELISM_HPP
#define TRISYCL_SYCL_PARALLELISM_DETAIL_PARALLELISM_HPP

/** \file

    Implement the detail of the parallel constructions to launch kernels

    Ronan at keryell dot FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <boost/multi_array.hpp>

#include "CL/sycl/group.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"

#ifdef _OPENMP
#include <omp.h>
#endif


/** \addtogroup parallelism
    @{
*/

namespace cl {
namespace sycl {
namespace detail {


/** A recursive multi-dimensional iterator that ends up calling f

    The iteration order may be changed later.

    Since partial specialization of function template is not possible in
    C++14, use a class template instead with everything in the
    constructor.
*/
template <std::size_t level, typename Range, typename ParallelForFunctor, typename Id>
struct parallel_for_iterate {
  parallel_for_iterate(Range r, ParallelForFunctor &f, Id &index) {
    for (boost::multi_array_types::index _sycl_index = 0,
           _sycl_end = r[Range::dimensionality - level];
         _sycl_index < _sycl_end;
         _sycl_index++) {
      // Set the current value of the index for this dimension
      index[Range::dimensionality - level] = _sycl_index;
      // Iterate further on lower dimensions
      parallel_for_iterate<level - 1,
                           Range,
                           ParallelForFunctor,
                           Id> { r, f, index };
    }
  }
};


/** A top-level recursive multi-dimensional iterator variant using OpenMP

    Only the top-level loop uses OpenMP and goes on with the normal
    recursive multi-dimensional.
*/
template <std::size_t level,
          typename Range,
          typename ParallelForFunctor,
          typename Id>
struct parallel_OpenMP_for_iterate {
  parallel_OpenMP_for_iterate(Range r, ParallelForFunctor &f) {
    // Create the OpenMP threads before the for-loop to avoid creating an
    // index in each iteration
#pragma omp parallel
    {
      // Allocate an OpenMP thread-local index
      Id index;
      // Make a simple loop end condition for OpenMP
      boost::multi_array_types::index _sycl_end =
        r[Range::dimensionality - level];
      /* Distribute the iterations on the OpenMP threads. Some OpenMP
         "collapse" could be useful for small iteration space, but it
         would need some template specialization to have real contiguous
         loop nests */
#pragma omp for
      for (boost::multi_array_types::index _sycl_index = 0;
           _sycl_index < _sycl_end;
           _sycl_index++) {
        // Set the current value of the index for this dimension
        index[Range::dimensionality - level] = _sycl_index;
        // Iterate further on lower dimensions
        parallel_for_iterate<level - 1,
                             Range,
                             ParallelForFunctor,
                             Id> { r, f, index };
      }
    }
  }
};


/** Stop the recursion when level reaches 0 by simply calling the
    kernel functor with the constructed id */
template <typename Range, typename ParallelForFunctor, typename Id>
struct parallel_for_iterate<0, Range, ParallelForFunctor, Id> {
  parallel_for_iterate(Range r, ParallelForFunctor &f, Id &index) {
    f(index);
  }
};


/** Implementation of a data parallel computation with parallelism
    specified at launch time by a range<>. Kernel index is id or int.

    This implementation use OpenMP 3 if compiled with the right flag.
*/
template <std::size_t Dimensions = 1, typename ParallelForFunctor, typename Id>
void parallel_for(range<Dimensions> r,
                  ParallelForFunctor f,
                  Id) {
#ifdef _OPENMP
  // Use OpenMP for the top loop level
  parallel_OpenMP_for_iterate<Dimensions,
                              range<Dimensions>,
                              ParallelForFunctor,
                              id<Dimensions>> { r, f };
#else
  // In a sequential execution there is only one index processed at a time
  id<Dimensions> index;
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       ParallelForFunctor,
                       id<Dimensions>> { r, f, index };
#endif
}


/** Implementation of a data parallel computation with parallelism
    specified at launch time by a range<>. Kernel index is item.

    This implementation use OpenMP 3 if compiled with the right flag.
*/
template <std::size_t Dimensions = 1, typename ParallelForFunctor>
void parallel_for(range<Dimensions> r,
                  ParallelForFunctor f,
                  item<Dimensions>) {
  auto reconstruct_item = [&] (id<Dimensions> l) {
    // Reconstruct the global item
    item<Dimensions> index { r, l };
    // Call the user kernel with the item<> instead of the id<>
    f(index);
  };
#ifdef _OPENMP
  // Use OpenMP for the top loop level
  parallel_OpenMP_for_iterate<Dimensions,
                              range<Dimensions>,
                              decltype(reconstruct_item),
                              id<Dimensions>> { r, reconstruct_item };
#else
  // In a sequential execution there is only one index processed at a time
  id<Dimensions> index;
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       decltype(reconstruct_item),
                       id<Dimensions>> { r, reconstruct_item, index };
#endif
}
```

```


/** Calls the appropriate ternary parallel_for overload based on the
    index type of the kernel function object f

*/
template <std::size_t Dimensions = 1, typename ParallelForFunctor>
void parallel_for(range<Dimensions> r, ParallelForFunctor f) {
  using mf_t  = decltype(std::mem_fn(&ParallelForFunctor::operator()));
  using arg_t = typename mf_t::second_argument_type;
  parallel_for(r,f,arg_t{});
}


/** Implementation of parallel_for with a range<> and an offset */
template <std::size_t Dimensions = 1, typename ParallelForFunctor>
void parallel_for_global_offset(range<Dimensions> global_size,
                                id<Dimensions> offset,
                                ParallelForFunctor f) {
  // Reconstruct the item from its id<> and its offset
  auto reconstruct_item = [&] (id<Dimensions> l) {
    // Reconstruct the global item
    item<Dimensions> index { global_size, l + offset, offset };
    // Call the user kernel with the item<> instead of the id<>
    f(index);
  };

  // First iterate on all the work-groups
  parallel_for(global_size, reconstruct_item);
}


/** Implement a variation of 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()
*/
template <std::size_t Dimensions = 1, typename ParallelForFunctor>
void parallel_for(nd_range<Dimensions> r,
                  ParallelForFunctor f) {
  // In a sequential execution there is only one index processed at a time
  nd_item<Dimensions> index { r };
  // To iterate on the work-group
  id<Dimensions> group;
  range<Dimensions> group_range = r.get_group();
  // To iterate on the local work-item
  id<Dimensions> local;

  range<Dimensions> local_range = r.get_local();

  // Reconstruct the nd_item from its group and local id
  auto reconstruct_item = [&] (id<Dimensions> l) {
    //local.display();
    // Reconstruct the global nd_item
    index.set_local(local);
    // Upgrade local_range to an id<> so that we can * with the group (an id<>)
    index.set_global(local + id<Dimensions>(local_range)*group);
    // Call the user kernel at last
    f(index);
  };

  /* To recycle the parallel_for on range<>, wrap the ParallelForFunctor f
     into another functor that iterates inside the work-group and then
     calls f */
  auto iterate_in_work_group = [&] (id<Dimensions> g) {
    //group.display();
    // Then iterate on the local work-groups
    parallel_for_iterate<Dimensions,
                         range<Dimensions>,
                         decltype(reconstruct_item),
                         id<Dimensions>> { local_range,
                                           reconstruct_item,
                                           local };
  };

  // First iterate on all the work-groups
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       decltype(iterate_in_work_group),
                       id<Dimensions>> { group_range,
                                         iterate_in_work_group,
      group };
}
```

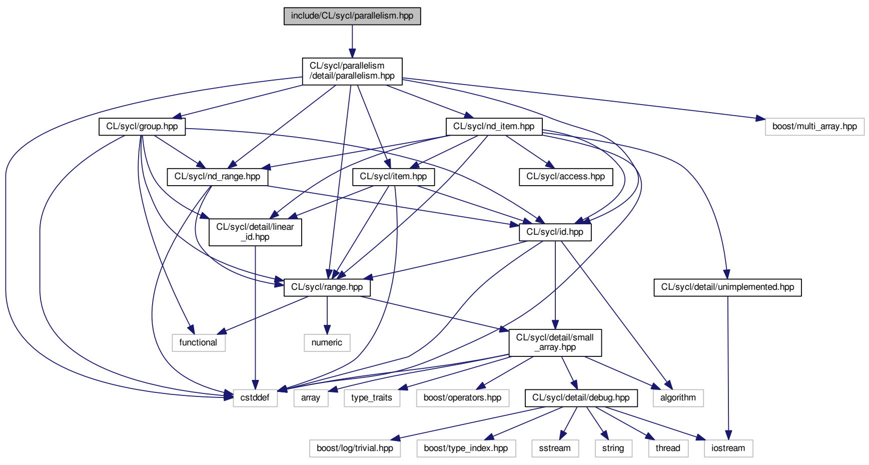
```


/// Implement the loop on the work-groups
template <std::size_t Dimensions = 1, typename ParallelForFunctor>
void parallel_for_workgroup(nd_range<Dimensions> r,
                            ParallelForFunctor f) {
  // In a sequential execution there is only one index processed at a time
  group<Dimensions> g { r };

  // First iterate on all the work-groups
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       ParallelForFunctor,
                       group<Dimensions>> {
    r.get_group(),
    f,
    g };
}


/** Implement the loop on the work-items inside a work-group

    \todo Better type the functor
*/
template <std::size_t Dimensions, typename ParallelForFunctor>
void parallel_for_workitem(const group<Dimensions> &g,
                           ParallelForFunctor f) {
#if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
  /* To implement barriers With OpenMP, one thread is created for each
     work-item in the group and thus an OpenMP barrier has the same effect
     of an OpenCL barrier executed by the work-items in a workgroup

     The issue is that the parallel_for_workitem() execution is slow even
     when nd_item::barrier() is not used
  */


  // Is the above comment true anymore ?
  // Maybe the following will be enough
  // #ifdef _OPENMP

  // With OMP, one task is created for each work-item in the group

  range<Dimensions> l_r = g.get_nd_range().get_local();
  int tot = l_r.get(0);
  for (int i = 1; i < (int) Dimensions; ++i){
    tot *= l_r.get(i);
  }
#pragma omp parallel
  {
#pragma omp single nowait
    {
      for (int th_id = 0; th_id < tot; ++th_id) {
#pragma omp task firstprivate(th_id)
        {
          nd_item<Dimensions> index { g.get_nd_range() };
          id<Dimensions> local; // to initialize correctly

          if (Dimensions ==1) {
            local[0] = th_id;
          } else if (Dimensions == 2) {
            local[0] = th_id / l_r.get(1);
            local[1] = th_id - local[0]*l_r.get(1);
          } else if (Dimensions == 3) {
            int tmp = l_r.get(1)*l_r.get(2);
            local[0] = th_id / tmp;
            local[1] = (th_id - local[0]*tmp) / l_r.get(1);
            local[2] = th_id - local[0]*tmp - local[1]*l_r.get(1);
          }
          index.set_local(local);
          index.set_global(local + id<Dimensions>(l_r)*g.get());
          f(index);
        }
      }
    }
  }
#else
  // In a sequential execution there is only one index processed at a time
  nd_item<Dimensions> index { g.get_nd_range() };
  // To iterate on the local work-item
  id<Dimensions> local;

  // Reconstruct the nd_item from its group and local id
  auto reconstruct_item = [&] (id<Dimensions> l) {
    //local.display();
    //l.display();
    // Reconstruct the global nd_item
    index.set_local(local);
    // \todo Some strength reduction here
    index.set_global(local + id<Dimensions>(g.get_local_range())*g.get());
    // Call the user kernel at last
    f(index);
  };

  // Then iterate on all the work-items of the work-group
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       decltype(reconstruct_item),
                       id<Dimensions>> {
    g.get_local_range(),
    reconstruct_item,
    local };
#endif
}
/// @} End the parallelism Doxygen group

} // namespace detail
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PARALLELISM_DETAIL_PARALLELISM_HPP
```

11.101 include/CL/sycl/parallelism.hpp File Reference

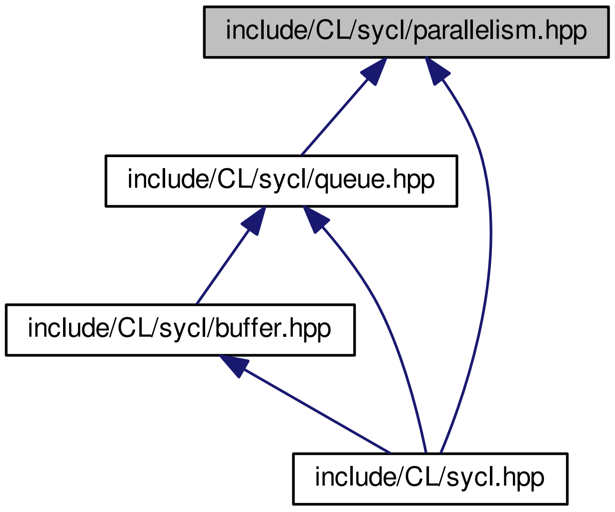
Implement parallel constructions to launch kernels.

`#include "CL/sycl/parallelism/detail/parallelism.hpp"`

Include dependency graph for parallelism.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/parallelism.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Namespaces

- cl

    The vector type to be used as SYCL vector.
- cl::sycl

Functions

- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_work_item (const group< Dimensions > &g, ParallelForFunctor f)

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

11.101.1 Detailed Description

Implement parallel constructions to launch kernels.

Ronan at keryell dot FR

Definition in file parallelism.hpp.

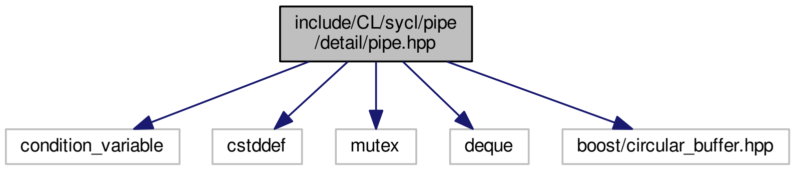
11.102 parallelism.hpp

```
#ifndef TRISYCL_SYCL_PARALLELISM_HPP
#define TRISYCL_SYCL_PARALLELISM_HPP

/** \file

    Implement parallel constructions to launch kernels

    Ronan at keryell dot FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/parallelism/detail/parallelism.hpp"

namespace cl {
namespace sycl {

/** \addtogroup parallelism
    @{
*/

/// SYCL parallel_for version that allows a Program object to be specified
/// \todo To be implemented
/* template <typename Range, typename Program, typename ParallelForFunctor>
void parallel_for(Range r, Program p, ParallelForFunctor f) {
  /// \todo deal with Program
  parallel_for(r, f);
}
*/

  /** Loop on the work-items inside a work-group

      \todo Deprecate this function in the specification to use
      instead the group method
  */
  template <std::size_t Dimensions = 1, typename ParallelForFunctor>
  void parallel_for_work_item(const group<Dimensions> &g,
                              ParallelForFunctor f) {
    detail::parallel_for_workitem(g, f);
  }



}
}

/// @} End the parallelism Doxygen group

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PARALLELISM_HPP
```

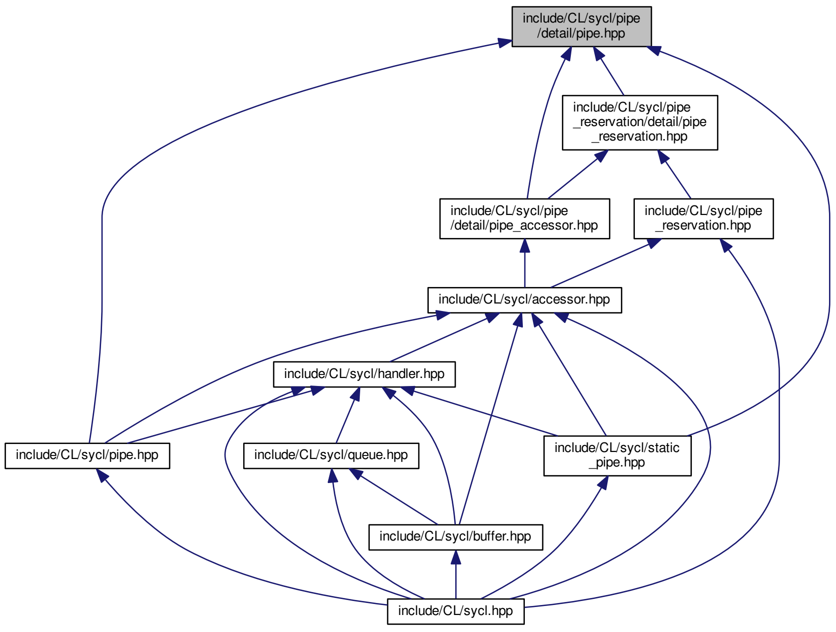
11.103 include/CL/sycl/pipe/detail/pipe.hpp File Reference

```
#include <condition_variable>
#include <cstddef>
#include <mutex>
#include <deque>
#include <boost/circular_buffer.hpp>
```

Include dependency graph for pipe.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct cl::sycl::detail::reserve_id< T >

  A private description of a reservation station. More...
- class cl::sycl::detail::pipe< T >

  Implement a pipe object. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.104 pipe.hpp

```
#ifndef TRISYCL_SYCL_PIPE_DETAIL_PIPE_HPP
#define TRISYCL_SYCL_PIPE_DETAIL_PIPE_HPP

/** \file The OpenCL SYCL pipe<> details

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <condition_variable>
#include <cstddef>
#include <mutex>
#include <deque>

#ifdef MAKE_BOOST_CIRCULARBUFFER_THREAD_SAFE
/* The debug mode of boost/circular_buffer.hpp has a nasty side effect
   in multithread applications using several iterators at the same
   time even in read-only mode because the library tracks them for
   debugging purpose in a... non-thread safe way

   This is described in https://svn.boost.org/trac/boost/ticket/6277
   and fixed with https://github.com/boostorg/circular_buffer/pull/9
*/
#define BOOST_CB_DISABLE_DEBUG
#endif
#include <boost/circular_buffer.hpp>

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/// A private description of a reservation station
template <typename T>
struct reserve_id {
  /// Start of the reservation in the pipe storage
  typename boost::circular_buffer<T>::iterator start;

  /// Number of elements in the reservation
  std::size_t size;

  /* True when the reservation has been committed and is ready to be
     released */
  bool ready = false;

  /** Track a reservation not committed yet

      \param[in] start point to the start of the reservation in the
      pipe storage

      \param[in] size is the number of elements in the reservation
  */
  reserve_id(typename boost::circular_buffer<T>::iterator start,
             std::size_t size) : start { start }, size { size } {}

};


/** Implement a pipe object

    Use some mutable members so that the pipe object can be changed even
    when the accessors are captured in a lambda.
*/
template <typename T>
class pipe : public detail::debug<pipe<T>> {

public:

  using value_type = T;

  /// Implement the pipe with a circular buffer
  using implementation_t = boost::circular_buffer<value_type>;

private:

  /// The circular buffer to store the elements
  boost::circular_buffer<value_type> cb;

  /** To protect the access to the circular buffer.
```

```

      In case the object is capture in a lambda per copy, make it
      mutable. */
  mutable std::mutex cb_mutex;

  /// The queue of pending write reservations
  std::deque<reserve_id<value_type>> w_rid_q;

public:

  using rid_iterator = typename decltype(w_rid_q)::iterator;

private:

  /// The queue of pending read reservations
  std::deque<reserve_id<value_type>> r_rid_q;

  /// Track the number of frozen elements related to read reservations
  std::size_t read_reserved_frozen;

  /// To signal that a read has been successful
  std::condition_variable read_done;

  /// To signal that a write has been successful
  std::condition_variable write_done;

  /// To control the debug mode, disabled by default
  bool debug_mode = false;

public:

  /// True when the pipe is currently used for reading
  bool used_for_reading = false;

  /// True when the pipe is currently used for writing
  bool used_for_writing = false;

  /// Create a pipe as a circular buffer of the required capacity
  pipe(std::size_t capacity) : cb { capacity }, read_reserved_frozen { 0 } { }


  /** Return the maximum number of elements that can fit in the pipe
   */
  std::size_t capacity() const {
    // No lock required since it is fixed and set at construction time
    return cb.capacity();
  }

private:

  /** Get the current number of elements in the pipe that can be read

      This is obviously a volatile value which is constrained by the
      theory of restricted relativity.

      Note that on some devices it may be costly to implement (for
      example on FPGA).
   */
  std::size_t size() const {
    TRISYCL_DUMP_T("size() cb.size() = " << cb.size()
                   << " cb.end() = " << (void *)&*cb.end()
                   << " reserved_for_reading() = " << reserved_for_reading()
                   << " reserved_for_writing() = " << reserved_for_writing());
    /* The actual number of available elements depends from the
       elements blocked by some reservations.
       This prevents a consumer to read into reserved area. */
    return cb.size() - reserved_for_reading() - reserved_for_writing();
  }


  /** Test if the pipe is empty

      This is obviously a volatile value which is constrained by
      restricted relativity.

      Note that on some devices it may be costly to implement on the
      write side (for example on FPGA).
  */
  bool empty() const {
    TRISYCL_DUMP_T("empty() cb.size() = " << cb.size()
                   << " size() = " << size());
    // It is empty when the size is zero, taking into account reservations
    return size() ==  0;
  }


  /** Test if the pipe is full
```

```

     This is obviously a volatile value which is constrained by
     restricted relativity.

     Note that on some devices it may be costly to implement on the
     read side (for example on FPGA).
  */
  bool full() const {
    return cb.full();
  }


public:

  /// The size() method used outside needs to lock the datastructure
  std::size_t size_with_lock() const {
    std::lock_guard<std::mutex> lg { cb_mutex };
    return size();
  }


  /// The empty() method used outside needs to lock the datastructure
  bool empty_with_lock() const {
    std::lock_guard<std::mutex> lg { cb_mutex };
    return empty();
  }


  // The full() method used outside needs to lock the datastructure
  bool full_with_lock() const {
    std::lock_guard<std::mutex> lg { cb_mutex };
    return full();
  }


  /** Try to write a value to the pipe

      \param[in] value is what we want to write

      \param[in] blocking specify if the call wait for the operation
      to succeed

      \return true on success

      \todo provide a && version
  */
  bool write(const T &value, bool blocking = false) {
    // Lock the pipe to avoid being disturbed
    std::unique_lock<std::mutex> ul { cb_mutex };
    TRISYCL_DUMP_T("Write pipe full = " << full()
                   << " value = " << value);

    if (blocking)
      /* If in blocking mode, wait for the not full condition, that
         may be changed when a read is done */
      read_done.wait(ul, [&] { return !full(); });
    else if (full())
      return false;

    cb.push_back(value);
    TRISYCL_DUMP_T("Write pipe front = " << cb.front()
                   << " back = " << cb.back()
                   << " cb.begin() = " << (void *)&*cb.begin()
                   << " cb.size() = " << cb.size()
                   << " cb.end() = " << (void *)&*cb.end()
                   << " reserved_for_reading() = " << reserved_for_reading()
                   << " reserved_for_writing() = " << reserved_for_writing());
    // Notify the clients waiting to read something from the pipe
    write_done.notify_all();
    return true;
  }


  /** Try to read a value from the pipe

      \param[out] value is the reference to where to store what is
      read

      \param[in] blocking specify if the call wait for the operation
      to succeed

      \return true on success
  */
  bool read(T &value, bool blocking = false) {
    // Lock the pipe to avoid being disturbed
    std::unique_lock<std::mutex> ul { cb_mutex };
    TRISYCL_DUMP_T("Read pipe empty = " << empty());

    if (blocking)
      /* If in blocking mode, wait for the not empty condition, that
         may be changed when a write is done */
      write_done.wait(ul, [&] { return !empty(); });
    else if (empty())
      return false;

    TRISYCL_DUMP_T("Read pipe front = " << cb.front()
                   << " back = " << cb.back()
                   << " reserved_for_reading() = " << reserved_for_reading());
    if (read_reserved_frozen)
      /** If there is a pending reservation, read the next element to
          be read and update the number of reserved elements */
      value = cb.begin()[read_reserved_frozen++];
    else {
      /* There is no pending read reservation, so pop the read value
         from the pipe */
      value = cb.front();
      cb.pop_front();
    }

    TRISYCL_DUMP_T("Read pipe value = " << value);
    // Notify the clients waiting for some room to write in the pipe
    read_done.notify_all();
    return true;
  }


  /** Compute the amount of elements blocked by read reservations, not yet
      committed

      This includes some normal reads to pipes between/after
      un-committed reservations

      This function assumes that the data structure is locked
  */
  std::size_t reserved_for_reading() const {
    return read_reserved_frozen;
  }


  /** Compute the amount of elements blocked by write reservations, not yet
      committed

      This includes some normal writes to pipes between/after
      un-committed reservations

      This function assumes that the data structure is locked
  */
  std::size_t reserved_for_writing() const {
    if (w_rid_q.empty())
      // No on-going reservation
      return 0;
    else
      /* The reserved size is from the first element of the first
         on-going reservation up to the end of the pipe content */
      return cb.end() - w_rid_q.front().start;
  }


  /** Reserve some part of the pipe for reading

      \param[in] s is the number of element to reserve

      \param[out] rid is an iterator to a description of the
      reservation that has been done if successful

      \param[in] blocking specify if the call wait for the operation
      to succeed

      \return true if the reservation was successful
  */
  bool reserve_read(std::size_t s,
                    rid_iterator &rid,
                    bool blocking = false)  {
    // Lock the pipe to avoid being disturbed
    std::unique_lock<std::mutex> ul { cb_mutex };

    TRISYCL_DUMP_T("Before read reservation cb.size() = " << cb.size()
                   << " size() = " << size());
    if (s == 0)
      // Empty reservation requested, so nothing to do
      return false;

    if (blocking)
      /* If in blocking mode, wait for enough elements to read in the
```

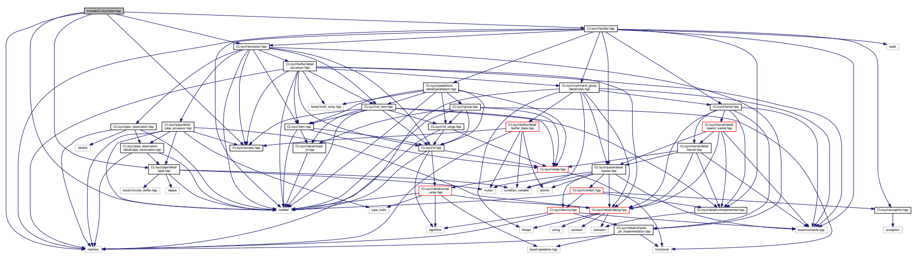
```
          pipe for the reservation. This condition can change when a
          write is done */
        write_done.wait(ul, [&] { return s <= size(); });
      else if (s > size())
        // Not enough elements to read in the pipe for the reservation
        return false;

      // Compute the location of the first element of the reservation
      auto first = cb.begin() + read_reserved_frozen;
      // Increment the number of frozen elements
      read_reserved_frozen += s;
      /* Add a description of the reservation at the end of the
         reservation queue */
      r_rid_q.emplace_back(first, s);
      // Return the iterator to the last reservation descriptor
      rid = r_rid_q.end() - 1;
      TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
                     << " size() = " << size());
      return true;
    }


    /** Reserve some part of the pipe for writing

        \param[in] s is the number of element to reserve

        \param[out] rid is an iterator to a description of the
        reservation that has been done if successful

        \param[in] blocking specify if the call wait for the operation
        to succeed

        \return true if the reservation was successful
    */
    bool reserve_write(std::size_t s,
                       rid_iterator &rid,
                       bool blocking = false)  {
      // Lock the pipe to avoid being disturbed
      std::unique_lock<std::mutex> ul { cb_mutex };

      TRISYCL_DUMP_T("Before write reservation cb.size() = " << cb.size()
                     << " size() = " << size());
      if (s == 0)
        // Empty reservation requested, so nothing to do
        return false;

      if (blocking)
        /* If in blocking mode, wait for enough room in the pipe, that
           may be changed when a read is done. Do not use a difference
           here because it is only about unsigned values */
        read_done.wait(ul, [&] { return cb.size() + s <= capacity(); });
      else if (cb.size() + s > capacity())
        // Not enough room in the pipe for the reservation
        return false;

      /* If there is enough room in the pipe, just create default values
         in it to do the reservation */
      for (std::size_t i = 0; i != s; ++i)
        cb.push_back();
      /* Compute the location of the first element a posteriori since it
         may not exist a priori if cb was empty before */
      auto first = cb.end() - s;
      /* Add a description of the reservation at the end of the
         reservation queue */
      w_rid_q.emplace_back(first, s);
      // Return the iterator to the last reservation descriptor
      rid = w_rid_q.end() - 1;
      TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
                     << " size() = " << size());
      return true;
    }


    /** Process the read reservations that are ready to be released in the
        reservation queue
    */
    void move_read_reservation_forward() {
      // Lock the pipe to avoid nuisance
      std::lock_guard<std::mutex> lg { cb_mutex };

      for (;;) {
        if (r_rid_q.empty())
          // No pending reservation, so nothing to do
          break;
        if (!r_rid_q.front().ready)
          /* If the first reservation is not ready to be released, stop
             because it is blocking all the following in the queue
```

```
            anyway */
          break;
        // Remove the reservation to be released from the queue
        r_rid_q.pop_front();
        std::size_t n_to_pop;
        if (r_rid_q.empty())
          // If it was the last one, remove all the reservation
          n_to_pop = read_reserved_frozen;
        else
          // Else remove everything up to the next reservation
          n_to_pop =  r_rid_q.front().start - cb.begin();
        // No longer take into account these reserved slots
        read_reserved_frozen -= n_to_pop;
        // Release the elements from the FIFO
        while (n_to_pop--)
          cb.pop_front();
        // Notify the clients waiting for some room to write in the pipe
        read_done.notify_all();
        /* ...and process the next reservation to see if it is ready to
           be released too */
      }
    }


    /** Process the write reservations that are ready to be released in the
        reservation queue
    */
    void move_write_reservation_forward() {
      // Lock the pipe to avoid nuisance
      std::lock_guard<std::mutex> lg { cb_mutex };

      for (;;) {
        if (w_rid_q.empty())
          // No pending reservation, so nothing to do
          break;
        // Get the first reservation
        const auto &rid = w_rid_q.front();
        if (!rid.ready)
          /* If the reservation is not ready to be released, stop
             because it is blocking all the following in the queue
             anyway */
          break;
        // Remove the reservation to be released from the queue
        w_rid_q.pop_front();
        // Notify the clients waiting to read something from the pipe
        write_done.notify_all();
        /* ...and process the next reservation to see if it is ready to
           be released too */
      }
    }

  };

  /// @} End the execution Doxygen group

  }
  }
  }

  /*
      # Some Emacs stuff:
      ### Local Variables:
      ### ispell-local-dictionary: "american"
      ### eval: (flyspell-prog-mode)
      ### End:
  */

  #endif // TRISYCL_SYCL_PIPE_DETAIL_PIPE_HPP
```

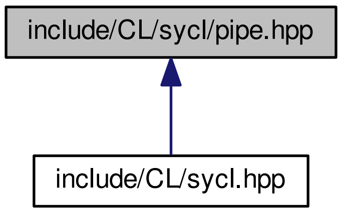
11.105 include/CL/sycl/pipe.hpp File Reference

```
#include <cstddef>
#include <memory>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/pipe/detail/pipe.hpp"
```

Include dependency graph for pipe.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class cl::sycl::pipe< T >

  A SYCL pipe. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.106 pipe.hpp

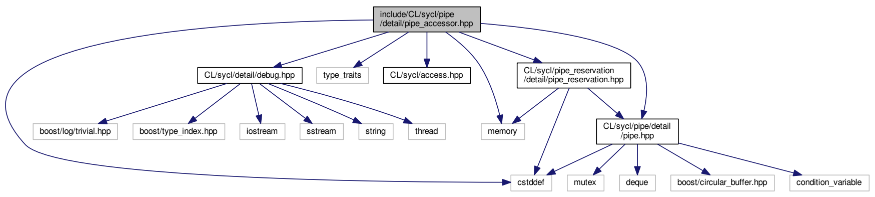
```
#ifndef TRISYCL_SYCL_PIPE_HPP
#define TRISYCL_SYCL_PIPE_HPP

/** \file The OpenCL SYCL pipe<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <memory>

```

```
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/pipe/detail/pipe.hpp"

namespace cl {
namespace sycl {

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** A SYCL pipe

    Implement a FIFO-style object that can be used through accessors
    to send some objects T from the input to the output
*/
template <typename T>
class pipe
    /* Use the underlying pipe implementation that can be shared in
       the SYCL model */
  : public detail::shared_ptr_implementation<pipe<T>, detail::pipe<T>>,
    detail::debug<pipe<T>> {

  // The type encapsulating the implementation
  using implementation_t =
    detail::shared_ptr_implementation<pipe<T>,
detail::pipe<T>>;

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

public:

  /// The STL-like types
  /* Since a pipe element cannot be directly addressed without
     accessor, only define value_type here */
  using value_type = T;


  /// Construct a pipe able to store up to capacity T objects
  pipe(std::size_t capacity)
    : implementation_t { new detail::pipe<T> { capacity } } { }


  /** Get an accessor to the pipe with the required mode

      \param Mode is the requested access mode

      \param Target is the type of pipe access required

      \param[in] command_group_handler is the command group handler in
      which the kernel is to be executed
  */
  template <access::mode Mode,
            access::target Target = access::target::pipe>
  accessor<value_type, 1, Mode, Target>
  get_access(handler &command_group_handler) {
    static_assert(Target == access::target::pipe
                  || Target == access::target::blocking_pipe,
                  "get_access(handler) with pipes can only deal with "
                  "access::pipe or access::blocking_pipe");
    return { implementation, command_group_handler };
  }


  /// Return the maximum number of elements that can fit in the pipe
  std::size_t capacity() const {
    return implementation->capacity();
    }

};

/// @} End the execution Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PIPE_HPP
```

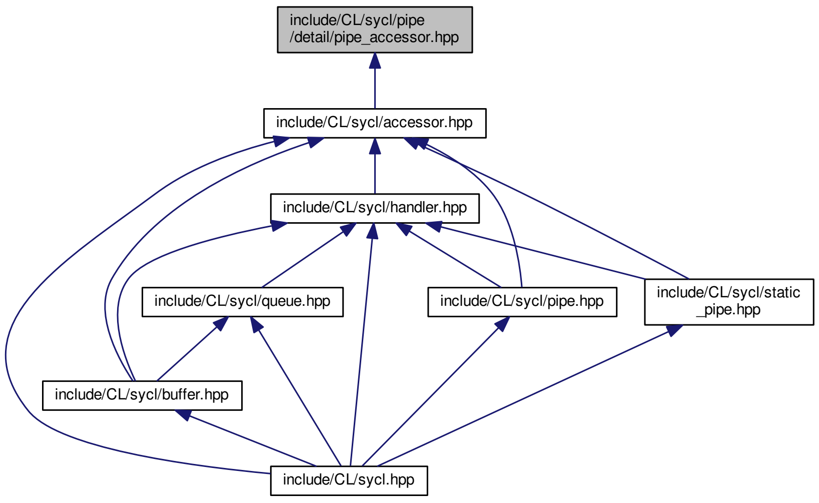
11.107 include/CL/sycl/pipe/detail/pipe_accessor.hpp File Reference

```
#include <cstddef>
#include <memory>
#include <type_traits>
#include "CL/sycl/access.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/pipe/detail/pipe.hpp"
#include "CL/sycl/pipe_reservation/detail/pipe_reservation.hpp"
```

Include dependency graph for pipe_accessor.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/pipe
/detail/pipe_accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static
_pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::detail::accessor< T, Dimensions, Mode, Target >

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

- class cl::sycl::detail::pipe_accessor< T, AccessMode, Target >

  The accessor abstracts the way pipe data are accessed inside a kernel. More...

Namespaces

- cl

    The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.108 pipe_accessor.hpp

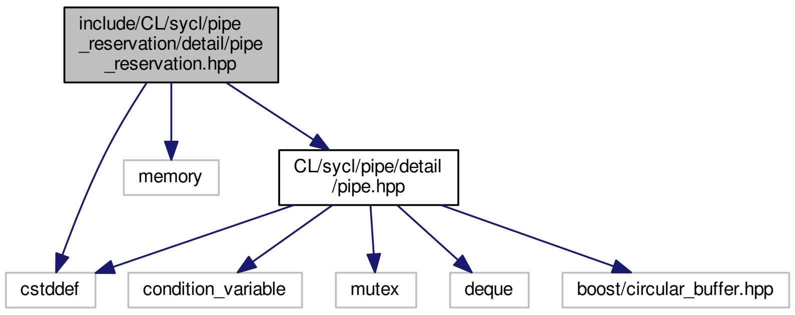
```
#ifndef TRISYCL_SYCL_PIPE_DETAIL_PIPE_ACCESSOR_HPP
#define TRISYCL_SYCL_PIPE_DETAIL_PIPE_ACCESSOR_HPP

/** \file The OpenCL SYCL pipe accessor<> detail behind the scene

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <memory>
#include <type_traits>

#include "CL/sycl/access.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/pipe/detail/pipe.hpp"
#include "CL/sycl/pipe_reservation/detail/pipe_reservation.hpp
"

namespace cl {
namespace sycl {

class handler;

namespace detail {

// Forward declaration of detail::accessor to declare the specialization
template <typename T,
          std::size_t Dimensions,
          access::mode Mode,
          access::target Target>
class accessor;
/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** The accessor abstracts the way pipe data are accessed inside a
    kernel
*/
template <typename T,
          access::mode AccessMode,
          access::target Target>
class pipe_accessor :
    public detail::debug<detail::pipe_accessor<T, AccessMode, Target>> {

public:

  static constexpr auto rank = 1;
  static constexpr auto mode = AccessMode;
  static constexpr auto target = Target;

  static constexpr bool blocking =
    (target == cl::sycl::access::target::blocking_pipe);

  /// The STL-like types
  using value_type = T;
  using reference = value_type&;
  using const_reference = const value_type&;

private:

  /// The real pipe implementation behind the hood
  std::shared_ptr<detail::pipe<T>> implementation;

  /** Store the success status of last pipe operation

      It is not impacted by reservation success.

      It does exist even if the pipe accessor is not evaluated in a
      boolean context for, but a use-def analysis can optimise it out
      in that case and not use some storage

      Use a mutable state here so that it can work with a [=] lambda
      capture without having to declare the whole lambda as mutable
  */
  bool mutable ok = false;

public:

  /** Construct a pipe accessor from an existing pipe
   */
  pipe_accessor(const std::shared_ptr<detail::pipe<T>> &p,
                handler &command_group_handler) :
    implementation { p } {
    //    TRISYCL_DUMP_T("Create a kernel pipe accessor write = "
    //                   << is_write_access());
    // Verify that the pipe is not already used in the requested mode
    if (mode == access::mode::write)
      if (implementation->used_for_writing)
        /// \todo Use pipe_exception instead
        throw std::logic_error { "The pipe is already used for writing." };
      else
        implementation->used_for_writing = true;
    else
      if (implementation->used_for_reading)
        throw std::logic_error { "The pipe is already used for reading." };
      else
        implementation->used_for_reading = true;
  }


  pipe_accessor() = default;


  /// Return the maximum number of elements that can fit in the pipe
  std::size_t capacity() const {
    return implementation->capacity();
  }

  /** Get the current number of elements in the pipe

      This is obviously a volatile value which is constrained by
      restricted relativity.

      Note that on some devices it may be costly to implement (for
      example on FPGA).
   */
  std::size_t size() const {
    return implementation->size_with_lock();
  }


  /** Test if the pipe is empty

      This is obviously a volatile value which is constrained by
      restricted relativity.

      Note that on some devices it may be costly to implement on the
      write side (for example on FPGA).
   */
  bool empty() const {
    return implementation->empty_with_lock();
  }


  /** Test if the pipe is full

      This is obviously a volatile value which is constrained by
      restricted relativity.

      Note that on some devices it may be costly to implement on the
      read side (for example on FPGA).
  */
  bool full() const {
    return implementation->full_with_lock();
  }


  /** In an explicit bool context, the accessor gives the success
      status of the last access

      It is not impacted by reservation success.

      The explicitness is related to avoid \code some_pipe <<
      some_value \endcode to be interpreted as \code some_bool <<
      some_value \endcode when the type of \code some_value \endcode
      is not the same type as the pipe type.

      \return true on success of the previous read or write operation
  */
  explicit operator bool() const {
    return ok;
  }


  /** Try to write a value to the pipe

      \param[in] value is what we want to write

      \return this so we can apply a sequence of write for example
      (but do not do this on a non blocking pipe...)

      \todo provide a && version

      This function is const so it can work when the accessor is
      passed by copy in the [=] kernel lambda, which is not mutable by
      default
  */
  const pipe_accessor &write(const value_type &value) const {
    static_assert(mode == access::mode::write,
                  "'.write(const value_type &value)' method on a pipe accessor"
                  " is only possible with write access mode");
    ok = implementation->write(value, blocking);
    // Return a reference to *this so we can apply a sequence of write
    return *this;
  }


  /** Some syntactic sugar to use \code a << v \endcode instead of
      \code a.write(v) \endcode */
  const pipe_accessor &operator<<(const value_type &value) const {
    static_assert(mode == access::mode::write,
                  "'<<' operator on a pipe accessor is only possible"
                  " with write access mode");
    // Return a reference to *this so we can apply a sequence of >>
    return write(value);
  }


  /** Try to read a value from the pipe

      \param[out] value is the reference to where to store what is
      read

      \return \code this \endcode so we can apply a sequence of read
      for example (but do not do this on a non blocking pipe...)

      This function is const so it can work when the accessor is
      passed by copy in the [=] kernel lambda, which is not mutable by
      default
  */
  const pipe_accessor &read(value_type &value) const {
    static_assert(mode == access::mode::read,
                  "'.read(value_type &value)' method on a pipe accessor"
                  " is only possible with read access mode");
    ok = implementation->read(value, blocking);
    // Return a reference to *this so we can apply a sequence of read
    return *this;
  }


  /** Read a value from a blocking pipe

      \return the read value directly, since it cannot fail on
      blocking pipe

      This function is const so it can work when the accessor is
      passed by copy in the [=] kernel lambda, which is not mutable by
      default
  */
  value_type read() const {
    static_assert(mode == access::mode::read,
                  "'.read()' method on a pipe accessor is only possible"
                  " with read access mode");
    static_assert(blocking,
                  "'.read()' method on a pipe accessor is only possible"
                  " with a blocking pipe");
    value_type value;
    implementation->read(value, blocking);
    return value;
  }

```

```

  /** Some syntactic sugar to use \code a >> v \endcode instead of
      \code a.read(v) \endcode */
  const pipe_accessor &operator>>(value_type &value) const {
    static_assert(mode == access::mode::read,
                  "'>>' operator on a pipe accessor is only possible"
                  " with read access mode");
    // Return a reference to *this so we can apply a sequence of >>
    return read(value);
  }


  detail::pipe_reservation<pipe_accessor>
reserve(std::size_t size) const {
    return { *implementation, size };
  }


  /// Set debug mode
  void set_debug(bool enable) const {
    implementation->debug_mode = enable;
  }


  auto &get_pipe_detail() {
    return implementation;
  }


  ~pipe_accessor() {
    /// Free the pipe for a future usage for the current mode
    if (mode == access::mode::write)
      implementation->used_for_writing = false;
    else
      implementation->used_for_reading = false;
  }

};

/// @} End the data Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PIPE_DETAIL_PIPE_ACCESSOR_HPP
```

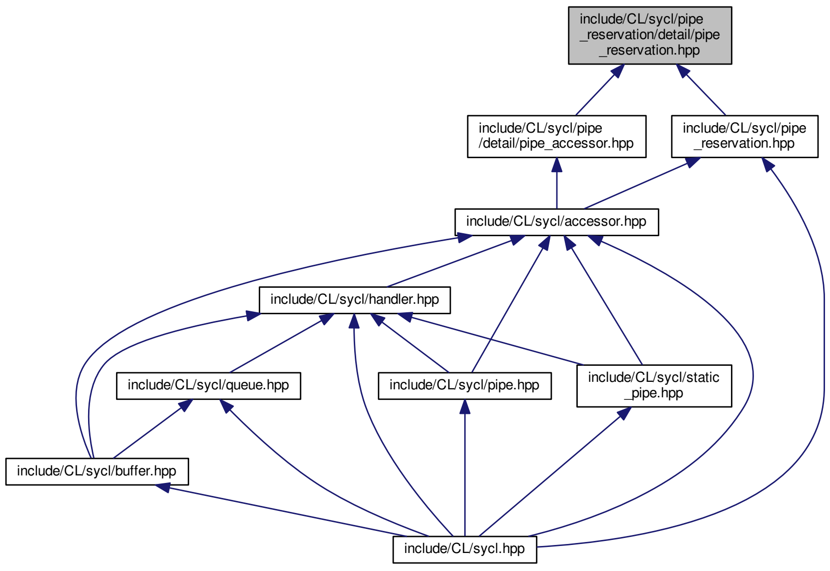
11.109 include/CL/sycl/pipe_reservation/detail/pipe_reservation.hpp File Reference

```
#include <cstddef>
#include <memory>
#include "CL/sycl/pipe/detail/pipe.hpp"
```

Include dependency graph for pipe_reservation.hpp:


include/CL/sycl/pipe _reservation/detail/pipe _reservation.hpp
memory
CL/sycl/pipe/detail /pipe.hpp
cstddef
condition_variable
mutex
deque
boost/circular_buffer.hpp

This graph shows which files directly or indirectly include this file:


include/CL/sycl/pipe _reservation/detail/pipe _reservation.hpp
include/CL/sycl/pipe /detail/pipe_accessor.hpp
include/CL/sycl/pipe _reservation.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static _pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::detail::accessor< T, Dimensions, Mode, Target >

  The buffer accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way. More...
- class cl::sycl::detail::pipe_reservation< PipeAccessor >

  The implementation of the pipe reservation station. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.110 pipe_reservation.hpp

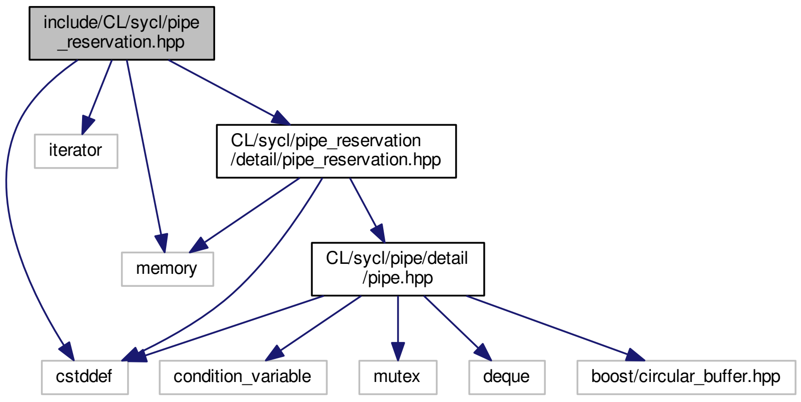
```
#ifndef TRISYCL_SYCL_PIPE_RESERVATION_DETAIL_PIPE_RESERVATION_HPP
#define TRISYCL_SYCL_PIPE_RESERVATION_DETAIL_PIPE_RESERVATION_HPP

/** \file The OpenCL SYCL pipe reservation detail behind the scene

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <memory>

#include "CL/sycl/pipe/detail/pipe.hpp"

namespace cl {
namespace sycl {
namespace detail {

template <typename T,
          std::size_t Dimensions,
          access::mode Mode,
          access::target Target>
class accessor;

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/// The implementation of the pipe reservation station
template <typename PipeAccessor>
class pipe_reservation :
    public detail::debug<detail::pipe_reservation<PipeAccessor>> {
  using accessor_type = PipeAccessor;
  static constexpr bool blocking =
    (accessor_type::target ==
cl::sycl::access::target::blocking_pipe);
  using value_type = typename accessor_type::value_type;
  using reference = typename accessor_type::reference;

public:

  using iterator =
    typename detail::pipe<value_type>::implementation_t::iterator
;
  using const_iterator =
    typename detail::pipe<value_type>::implementation_t::const_iterator
;

  // \todo Add to the specification
  static constexpr access::mode mode = accessor_type::mode;
  static constexpr access::target target =
accessor_type::target;

  /**  True if the reservation was successful and still uncommitted. B
       default a pipe_reservation is not reserved and cannot be
       committed */
  bool ok = false;

  /// Point into the reservation buffer. Only valid if ok is true
  typename detail::pipe<value_type>::rid_iterator
rid;

  /** Keep a reference on the pipe to access to the data and methods

      Note that with inlining and CSE it should not use more register
      when compiler optimization is in use. */
  detail::pipe<value_type> &p;


  /** Test that the reservation is in a usable state

      \todo Throw exception instead
  */
  void assume_validity() {
    assert(ok);
  }

public:

  /// Create a pipe reservation station that reserves the pipe itself
  pipe_reservation(detail::pipe<value_type> &p, std::size_t s) : p
{ p } {
    static_assert(mode == access::mode::write
                  || mode == access::mode::read,
                  "A pipe can only be accesed in read or write mode,"
                  " exclusively");

    /* Since this test is constexpr and dependent of a template
       parameter, it should be equivalent to a specialization of the
       method but in a clearer way */
    if (mode == access::mode::write)
      ok = p.reserve_write(s, rid, blocking);
    else
      ok = p.reserve_read(s, rid, blocking);
  }


  /** No copy constructor with some spurious commit in the destructor
      of the original object
  */
  pipe_reservation(const pipe_reservation &) = delete;


  /// Only a move constructor is required to move it into the shared_ptr
  pipe_reservation(pipe_reservation &&orig) :
    ok {orig.ok },
    rid {orig.rid },
    p { orig.p } {
      /* Even when an object is moved, the destructor of the old
         object is eventually called, so leave the old object in a
         destructable state but without any commit capability */
      orig.ok = false;
    }


  /** Keep the default constructors too

      Otherwise there is no move semantics and the copy is made by
      creating a new reservation and destructing the old one with a
      spurious commit in the meantime...
  */
  pipe_reservation() = default;


  /** Test if the reservation succeeded and thus if the reservation
      can be committed

      Note that it is up to the user to ensure that all the
      reservation elements have been initialized correctly in the case
      of a write for example
  */
  operator bool() {
    return ok;
  }


  /// Start of the reservation area
  iterator begin() {
    assume_validity();
    return rid->start;
  }


  /// Past the end of the reservation area
  iterator end() {
    assume_validity();
    return rid->start + rid->size;
  }


  /// Get the number of elements in the reservation station
  std::size_t size() {
    assume_validity();
    return rid->size;
  }


  /// Access to an element of the reservation
  reference operator[](std::size_t index) {
    assume_validity();
    TRISYCL_DUMP_T("[] index = " << index
                   << " Reservation write address = " << &(rid->start[index]));

    return rid->start[index];
  }


  /** Commit the reservation station

```

```
      \todo Add to the specification that for simplicity a reservation
      can be commited several times but only the first one is taken
      into account
  */
  void commit() {
    if (ok) {
      // If the reservation is in a committable state, commit
      TRISYCL_DUMP_T("Commit");
      rid->ready = true;
      if (mode == access::mode::write)
        p.move_write_reservation_forward();
      else
        p.move_read_reservation_forward();
      ok = false;
    }
  }


  /// An implicit commit is made in the destructor
  ~pipe_reservation() {
    commit();
  }

};

/// @} End the data Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PIPE_RESERVATION_DETAIL_PIPE_RESERVATION_HPP
```

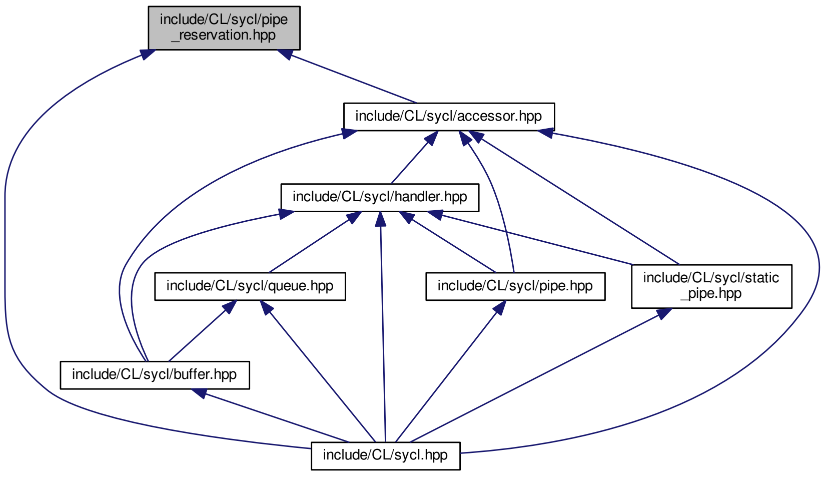
11.111 include/CL/sycl/pipe_reservation.hpp File Reference

```
#include <cstddef>
#include <iterator>
#include <memory>
#include "CL/sycl/pipe_reservation/detail/pipe_reservation.hpp"
```

Include dependency graph for pipe_reservation.hpp:


include/CL/sycl/pipe_reservation.hpp
iterator
CL/sycl/pipe_reservation/detail/pipe_reservation.hpp
memory
CL/sycl/pipe/detail/pipe.hpp
cstddef
condition_variable
mutex
deque
boost/circular_buffer.hpp

This graph shows which files directly or indirectly include this file:


include/CL/sycl/pipe_reservation.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- struct cl::sycl::pipe_reservation< PipeAccessor >

  The pipe reservation station allows to reserve an array-like view inside the pipe for ordered race-free access from various work-items for example. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.112 pipe_reservation.hpp

```
#ifndef TRISYCL_SYCL_PIPE_RESERVATION_HPP
#define TRISYCL_SYCL_PIPE_RESERVATION_HPP

/** \file The reservation station for OpenCL SYCL pipe accessor<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <iterator>
#include <memory>

#include "CL/sycl/pipe_reservation/detail/pipe_reservation.hpp
"

namespace cl {
namespace sycl {

/** \addtogroup data Data access and storage in SYCL
    @{
*/

```

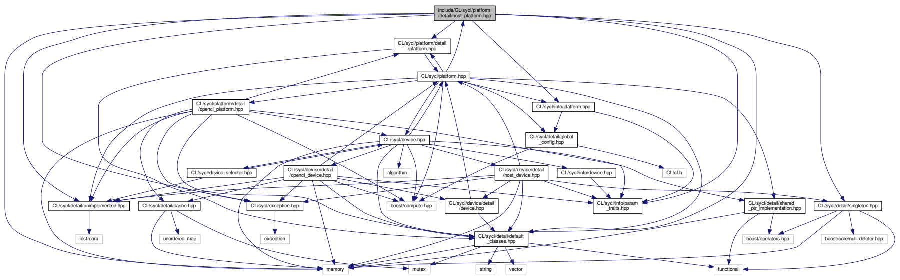
```
/** The pipe reservation station allows to reserve an array-like view
    inside the pipe for ordered race-free access from various
    work-items for example
 */
template <typename PipeAccessor>
struct pipe_reservation {
  using accessor_type = PipeAccessor;
  static constexpr bool blocking =
    (accessor_type::target ==
cl::sycl::access::target::blocking_pipe);
  using accessor_detail = typename accessor_type::accessor_detail;
  /// The STL-like types
  using value_type = typename accessor_type::value_type;
  using reference = value_type&;
  using const_reference = const value_type&;
  using pointer = value_type*;
  using const_pointer = const value_type*;
  using size_type = std::size_t;
  using difference_type = ptrdiff_t;
  using iterator =
    typename detail::pipe_reservation<accessor_detail>::iterator
;
  using const_iterator =
    typename detail::pipe_reservation<accessor_detail>::const_iterator
;
  using reverse_iterator = std::reverse_iterator<iterator>;
  using const_reverse_iterator = std::reverse_iterator<const_iterator>;

  /** Point to the underlying implementation that can be shared in the
      SYCL model with a handler semantics */
  typename std::shared_ptr<detail::pipe_reservation<accessor_detail>>
  implementation;

  /** Use default constructors so that we can create a new buffer copy
      from another one, with either a l-value or a r-value (for
      std::move() for example).

      Since we just copy the shared_ptr<> above, this is where/how the
      sharing magic is happening with reference counting in this case.
  */
  pipe_reservation() = default;


  /// Create a pipe_reservation for an accessor and a number of elements
  pipe_reservation(accessor_type &accessor, std::size_t s)
    : implementation {
    new detail::pipe_reservation<accessor_detail> {
      get_pipe_detail(accessor), s }
  } {}


  /** Create a pipe_reservation from the implementation detail

      This is an internal constructor to allow reserve() on the
      implementation to lift a full-fledged object through
      accessor::reserve().

      \todo Make it private and add required friends
   */
  pipe_reservation(detail::pipe_reservation<accessor_detail>
 &&pr)
    : implementation {
    new detail::pipe_reservation<accessor_detail> { std::move(pr)
} }
  {}


  /** Test if the pipe_reservation has been correctly allocated

      \return true if the pipe_reservation can be used and committed
  */
  operator bool() const {
    return *implementation;
  }


  /// Get the number of reserved element(s)
  std::size_t size() const {
    return implementation->size();
  }


  /// Access to a given element of the reservation
  reference operator[](std::size_t index) const {
    return (*implementation)[index];
  }


  /** Force a commit operation

      Normally the commit is implicitly done in the destructor, but
      sometime it is useful to do it earlier.
  */
  void commit() const {
    return implementation->commit();
  }


  /// Get an iterator on the first element of the reservation station
  iterator begin() const {
    return implementation->begin();
  }


  /// Get an iterator past the end of the reservation station
  iterator end() const {
    return implementation->end();
  }


  /// Build a constant iterator on the first element of the reservation station
  const_iterator cbegin() const {
    return implementation->begin();
  }


  /// Build a constant iterator past the end of the reservation station
  const_iterator cend() const {
    return implementation->end();
  }


  /// Get a reverse iterator on the last element of the reservation station
  reverse_iterator rbegin() const {
    return std::make_reverse_iterator(end());
  }


  /** Get a reverse iterator on the first element past the end of the
      reservation station */
  reverse_iterator rend() const {
    return std::make_reverse_iterator(begin());
  }


  /** Get a constant reverse iterator on the last element of the
      reservation station */
  const_reverse_iterator crbegin() const {
    return std::make_reverse_iterator(cend());
  }


  /** Get a constant reverse iterator on the first element past the
      end of the reservation station */
  const_reverse_iterator crend() const {
    return std::make_reverse_iterator(cbegin());
  }

};

/// @} End the data Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PIPE_RESERVATION_HPP
```

11.113 include/CL/sycl/platform/detail/host_platform.hpp File Reference

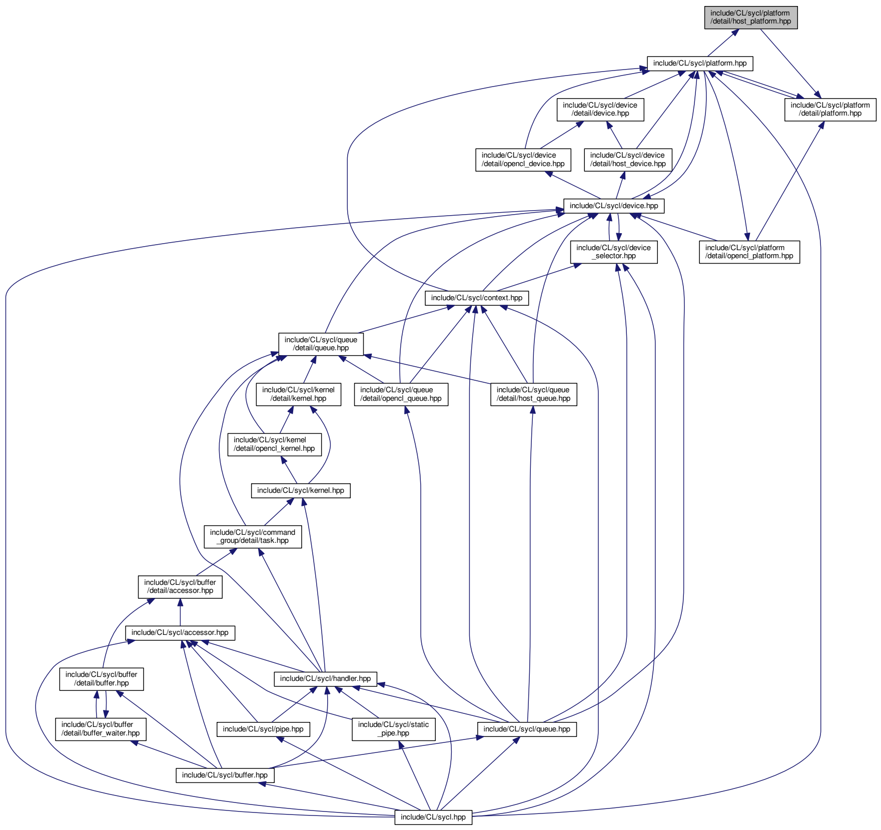
```
#include <memory>
```

```
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/singleton.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/info/platform.hpp"
#include "CL/sycl/platform/detail/platform.hpp"
```

Include dependency graph for host_platform.hpp:


include/CL/sycl/platform/detail/host_platform.hpp
CL/sycl/platform/detail/platform.hpp
CL/sycl/platform.hpp
CL/sycl/platform/detail/opencl_platform.hpp
CL/sycl/info/platform.hpp
CL/sycl/detail/global_config.hpp
CL/sycl/device.hpp
CL/sycl/device_selector.hpp
CL/sycl/device/detail/opencl_device.hpp
algorithm
CL/sycl/device/detail/host_device.hpp
CL/sycl/info/device.hpp
CL/cl.h
CL/sycl/detail/unimplemented.hpp
CL/sycl/detail/cache.hpp
CL/sycl/exception.hpp
boost/compute.hpp
CL/sycl/device/detail/device.hpp
CL/sycl/info/param_traits.hpp
CL/sycl/detail/shared_ptr_implementation.hpp
CL/sycl/detail/singleton.hpp
iostream
unordered_map
exception
CL/sycl/detail/default_classes.hpp
boost/operators.hpp
boost/core/null_deleter.hpp
memory
mutex
string
vector
functional

This graph shows which files directly or indirectly include this file:


include/CL/sycl/platform/detail/host_platform.hpp
include/CL/sycl/platform.hpp
include/CL/sycl/device/detail/device.hpp
include/CL/sycl/platform/detail/platform.hpp
include/CL/sycl/device/detail/opencl_device.hpp
include/CL/sycl/device/detail/host_device.hpp
include/CL/sycl/device.hpp
include/CL/sycl/device_selector.hpp
include/CL/sycl/platform/detail/opencl_platform.hpp
include/CL/sycl/context.hpp
include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::detail::host_platform

  SYCL host platform. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

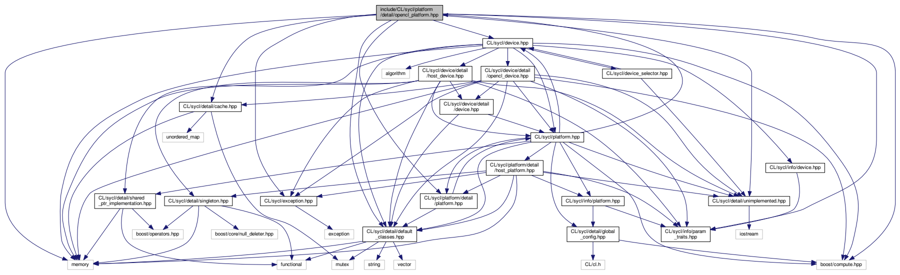
11.114 host_platform.hpp

```
#ifndef TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_HPP
#define TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_HPP

/** \file The OpenCL triSYCL host platform implementation

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/
#include <memory>

#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/detail/singleton.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/info/platform.hpp"
#include "CL/sycl/platform/detail/platform.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/// SYCL host platform
class host_platform : public detail::platform,
                      public detail::singleton<host_platform> {

// \todo Have this compatible with has_extension
auto static constexpr platform_extensions = "Xilinx_blocking_pipes";

public:

#ifdef TRISYCL_OPENCL
  /** Return the cl_platform_id of the underlying OpenCL platform

      This throws an error since there is no OpenCL platform associated
      to the host platform.
  */
  cl_platform_id get() const override {
    throw non_cl_error("The host platform has no OpenCL platform");
  }
#endif


  /// Return true since this platform is the SYCL host platform
  bool is_host() const override {
    return true;
  }


#if 0
  /** Returns at most the host device for this platform, according to
      the requested kind

      By default returns all the devices, which is obviously the host
      one here

      \todo To be implemented
  */
  vector_class<device>
  get_devices(info::device_type device_type =
info::device_type::all)
    const override
  {
    detail::unimplemented();
    return {};
  }
#endif


  /** Returning the information parameters for the host platform
      implementation
  */
  string_class get_info_string(info::platform param) const
 override {
    switch (param) {
    case info::platform::profile:
      /* Well... Is the host platform really a full profile whereas it
         is not really OpenCL? */
      return "FULL_PROFILE";

    case info::platform::version:
      // \todo I guess it should include the software version too...
      return "2.2";

    case info::platform::name:
      return "triSYCL host platform";

    case info::platform::vendor:
      return "triSYCL Open Source project";

    case info::platform::extensions:
      return platform_extensions;

    default:
      // \todo Define some SYCL exception type for this type of errors
      throw std::invalid_argument {
        "Unknown parameter value for SYCL platform information" };
    }
  }


  /** Specify whether a specific extension is supported on the platform

      \todo To be implemented
  */
  bool has_extension(const string_class &extension) const override {
    detail::unimplemented();
    return {};
  }

};

/// @} to end the execution Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_HPP
```

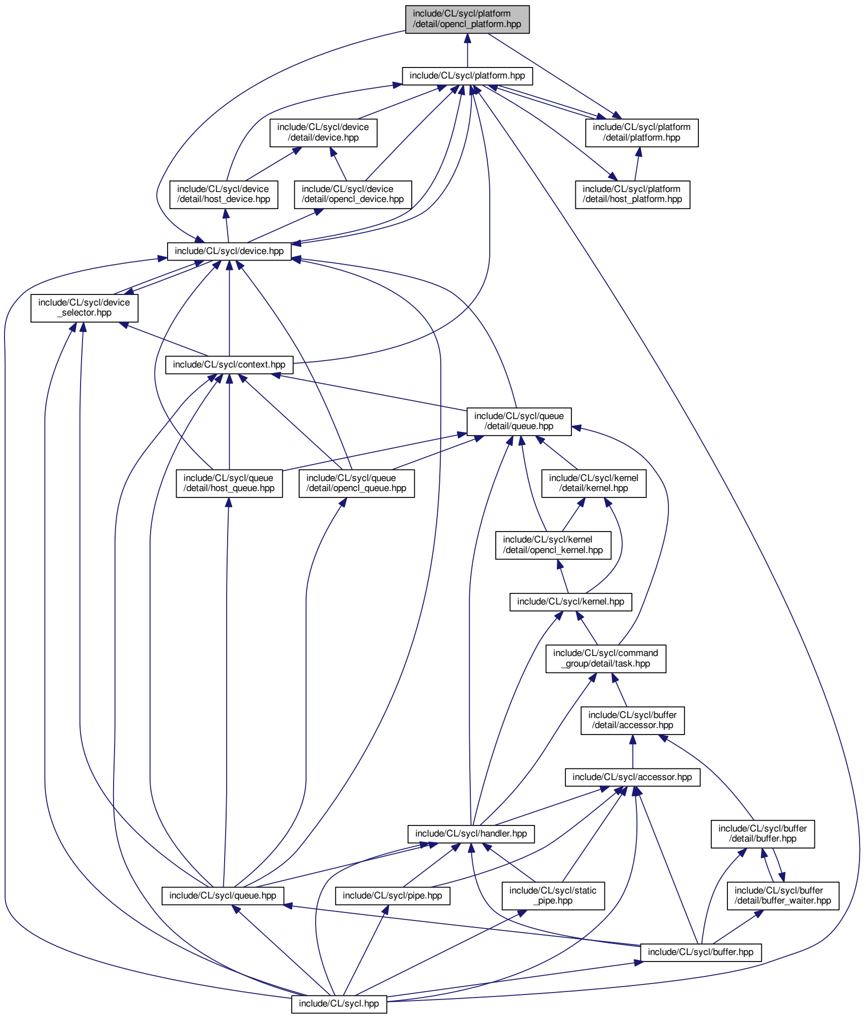
11.115 include/CL/sycl/platform/detail/opencl_platform.hpp File Reference

```
#include <memory>
#include <boost/compute.hpp>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform/detail/platform.hpp"
```

Include dependency graph for opencl_platform.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class cl::sycl::detail::opencl_platform

  SYCL OpenCL platform. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- detail::cache< cl_kernel, detail::opencl_kernel > opencl_kernel::cache cl::sycl::detail::__attribute__ ((weak))

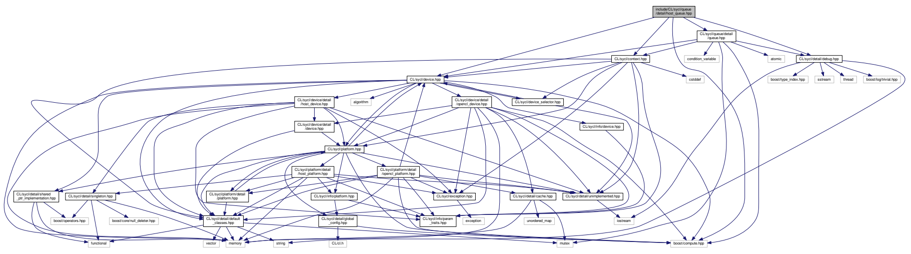
11.116 opencl_platform.hpp

```
#ifndef TRISYCL_SYCL_PLATFORM_DETAIL_OPENCL_PLATFORM_HPP
#define TRISYCL_SYCL_PLATFORM_DETAIL_OPENCL_PLATFORM_HPP

/** \file The OpenCL triSYCL OpenCL platform implementation

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/
#include <memory>

#include <boost/compute.hpp>

#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform/detail/platform.hpp"

namespace cl {
namespace sycl {

class device;

namespace detail {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/// SYCL OpenCL platform
class opencl_platform : public detail::platform {

  /// Use the Boost Compute abstraction of the OpenCL platform
  boost::compute::platform p;

  /** A cache to always return the same live platform for a given OpenCL
      platform

      C++11 guaranties the static construction is thread-safe
  */
  static detail::cache<cl_platform_id, detail::opencl_platform>
 cache;

public:

  /// Return the cl_platform_id of the underlying OpenCL platform
  cl_platform_id get() const override {
    return p.id();
  }


  /// Return false since an OpenCL platform is not the SYCL host platform
  bool is_host() const override {
    return false;
  }


#if 0
  /** Returns at most the host device for this platform, according to
      the requested kind

      By default returns all the devices, which is obviously the host
      one here

      \todo To be implemented
  */
  vector_class<cl::sycl::device>
  get_devices(info::device_type device_type = info::device_type::all)
    const override
  {
    detail::unimplemented();
    return {};
  }
#endif


  /// Returning the information string parameters for the OpenCL platform
  string_class get_info_string(info::platform param) const override {
    /* Use the fact that the triSYCL info values are the same as the
       OpenCL ones used in Boost.Compute to just cast the enum class
       to the int value */
    return p.get_info<std::string>(static_cast<cl_platform_info>(param));
  }


  /// Specify whether a specific extension is supported on the platform
  bool has_extension(const string_class &extension) const override {
    return p.supports_extension(extension);
  }


  ///// Get a singleton instance of the opencl_platform
  static std::shared_ptr<opencl_platform>
  instance(const boost::compute::platform &p) {
    return cache.get_or_register(p.id(),
                                 [&] { return new opencl_platform { p }; });
  }

private:

  /// Only the instance factory can built it
  opencl_platform(const boost::compute::platform &p) : p { p } {}

public:

  /// Unregister from the cache on destruction
  ~opencl_platform() override {
    cache.remove(p.id());
  }

};

/* Allocate the cache here but since this is a pure-header library,
   use a weak symbol so that only one remains when SYCL headers are
   used in different compilation units of a program
*/
detail::cache<cl_platform_id, detail::opencl_platform> opencl_platform::cache
  __attribute__((weak));

/// @} to end the execution Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_HPP
```

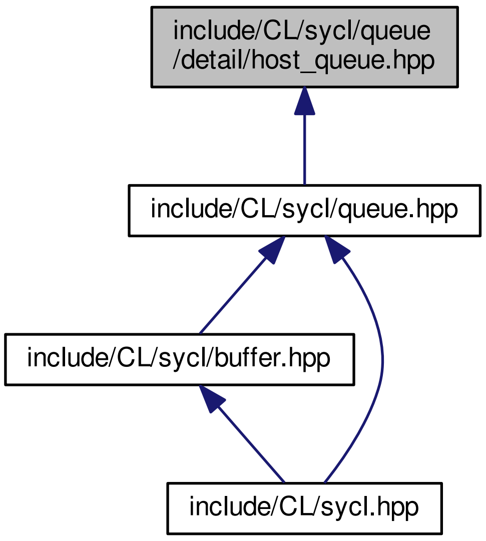
11.117 include/CL/sycl/queue/detail/host_queue.hpp File Reference

```
#include <boost/compute.hpp>
#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/queue/detail/queue.hpp"
```

Include dependency graph for host_queue.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class cl::sycl::detail::host_queue

    Some implementation details about the SYCL queue.

Namespaces

- cl

    The vector type to be used as SYCL vector.

- cl::sycl
- cl::sycl::detail

11.118 host_queue.hpp

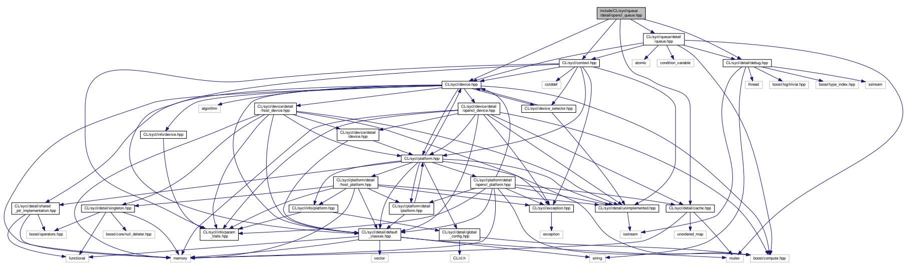
```
#ifndef TRISYCL_SYCL_QUEUE_DETAIL_HOST_QUEUE_HPP
#define TRISYCL_SYCL_QUEUE_DETAIL_HOST_QUEUE_HPP

/** \file Some implementation details of the host queue

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/queue/detail/queue.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** Some implementation details about the SYCL queue

    \todo Once a triSYCL queue is no longer blocking, make this a singleton
 */
class host_queue : public detail::queue,
                   detail::debug<host_queue> {

#ifdef TRISYCL_OPENCL
  /** Return the cl_command_queue of the underlying OpenCL queue

      This throws an error since there is no OpenCL queue associated
      to the host queue.
  */
  cl_command_queue get() const override {
    throw non_cl_error("The host queue has no OpenCL command queue");
  }


  /** Return the underlying Boost.Compute command queue

      This throws an error since there is no OpenCL queue associated
      to the host queue.
  */
  boost::compute::command_queue &get_boost_compute() override {
    throw non_cl_error("The host queue has no OpenCL command queue");
  }
#endif


  /// Return the SYCL host queue's host context
  cl::sycl::context get_context() const override {
    // Return the default context which is the host context
    return {};
  }


  /// Return the SYCL host device the host queue is associated with
  cl::sycl::device get_device() const override {
    // Return the default device which is the host device
    return {};
  }


  /// Claim proudly that the queue is executing on the SYCL host device
  bool is_host() const override {
    return true;
  }


};

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
```

```
00085    ### End:
00086 */
00087
00088 #endif // TRISYCL_SYCL_QUEUE_DETAIL_HOST_QUEUE_HPP
```

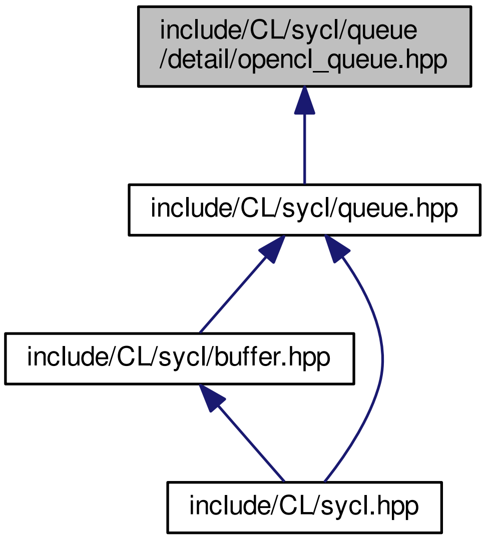
11.119 include/CL/sycl/queue/detail/opencl_queue.hpp File Reference

```
#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/queue/detail/queue.hpp"
```

Include dependency graph for opencl_queue.hpp:



This graph shows which files directly or indirectly include this file:



include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- class cl::sycl::detail::opencl_queue

  Some implementation details about the SYCL queue.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- detail::cache< cl_kernel, detail::opencl_kernel > opencl_kernel::cache cl::sycl::detail::__attribute__ ((weak))

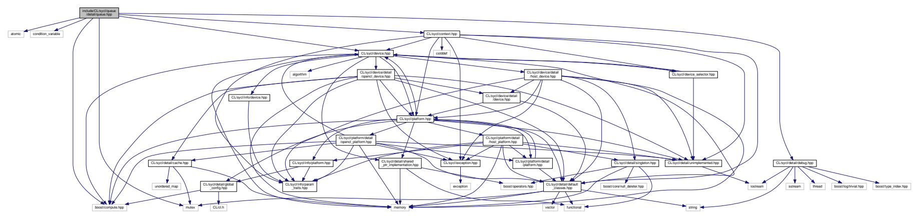
11.120 opencl_queue.hpp

```
#ifndef TRISYCL_SYCL_QUEUE_DETAIL_OPENCL_QUEUE_HPP
#define TRISYCL_SYCL_QUEUE_DETAIL_OPENCL_QUEUE_HPP

/** \file Some implementation details of the OpenCL queue

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/queue/detail/queue.hpp"

namespace cl {
namespace sycl {
namespace detail {

/// Some implementation details about the SYCL queue
class opencl_queue : public detail::queue,
                     detail::debug<opencl_queue> {
  /// Use the Boost Compute abstraction of the OpenCL command queue
  boost::compute::command_queue q;

  /** A cache to always return the same alive queue for a given OpenCL
      command queue

      C++11 guaranties the static construction is thread-safe
  */
  static detail::cache<cl_command_queue, detail::opencl_queue>
 cache;

  /// Return the cl_command_queue of the underlying OpenCL queue
  cl_command_queue get() const override {
    return q.get();
  }


  /// Return the underlying Boost.Compute command queue
  boost::compute::command_queue &get_boost_compute() override {
    return q;
  }


  /// Return the SYCL context associated to the queue
  /// \todo Finish context
  cl::sycl::context get_context() const override {
//    return q.get_context();
    return {};
  }


  /// Return the SYCL device associated to the queue
  cl::sycl::device get_device() const override {
    return q.get_device();
  }


  /// Claim proudly that an OpenCL queue cannot be the SYCL host queue
  bool is_host() const override {
    return false;
  }

private:

  /// Only the instance factory can built it
  opencl_queue(const boost::compute::command_queue &q) : q { q } {}

public:

  ///// Get a singleton instance of the opencl_queue
  static std::shared_ptr<opencl_queue>
  instance(const boost::compute::command_queue &q) {
    return cache.get_or_register(q.get(),
                                 [&] { return new opencl_queue { q }; });
  }


  /// Unregister from the cache on destruction
  ~opencl_queue() override {
    cache.remove(q.get());
  }

};

/* Allocate the cache here but since this is a pure-header library,
   use a weak symbol so that only one remains when SYCL headers are
   used in different compilation units of a program
*/
detail::cache<cl_command_queue, detail::opencl_queue> opencl_queue::cache
  __attribute__((weak));

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_QUEUE_DETAIL_OPENCL_QUEUE_HPP
```

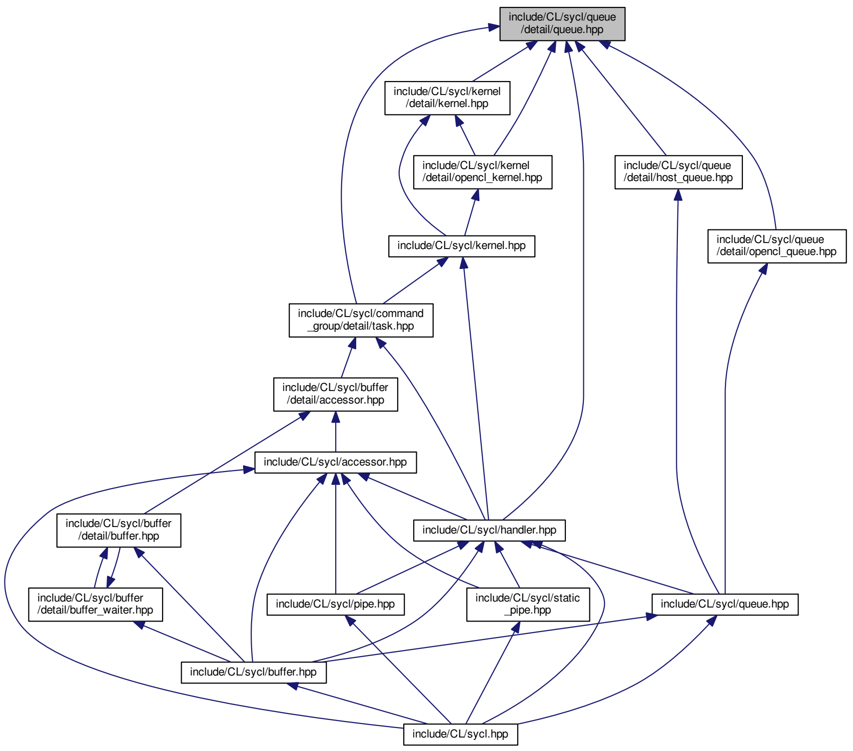
11.121 include/CL/sycl/queue/detail/queue.hpp File Reference

```
#include <atomic>
#include <condition_variable>
#include <mutex>
#include <boost/compute.hpp>
#include "CL/sycl/context.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/detail/debug.hpp"
```

Include dependency graph for queue.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/queue/detail/queue.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/queue/detail/host_queue.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/queue/detail/opencl_queue.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

- struct cl::sycl::detail::queue

  Some implementation details about the SYCL queue.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.122 queue.hpp

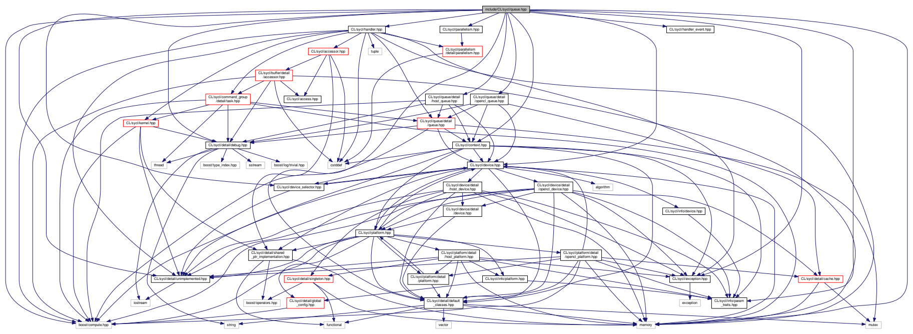
```
#ifndef TRISYCL_SYCL_QUEUE_DETAIL_QUEUE_HPP
#define TRISYCL_SYCL_QUEUE_DETAIL_QUEUE_HPP

/** \file Some implementation details of queue.

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <atomic>
#include <condition_variable>
```

```
#include <mutex>

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/context.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/detail/debug.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** Some implementation details about the SYCL queue
 */
struct queue : detail::debug<detail::queue> {
  /// Track the number of kernels still running to wait for their completion
  std::atomic<size_t> running_kernels;

  /// To signal when all the kernels have completed
  std::condition_variable finished;
  /// To protect the access to the condition variable
  std::mutex finished_mutex;


  /// Initialize the queue with 0 running kernel
  queue() {
    running_kernels = 0;
  }


  /// Wait for all kernel completion
  void wait_for_kernel_execution() {
    TRISYCL_DUMP_T("Queue waiting for kernel completion");
    std::unique_lock<std::mutex> ul { finished_mutex };
    finished.wait(ul, [&] {
        // When there is no kernel running in this queue, we are ready to go
        return running_kernels == 0;
      });
  }


  /// Signal that a new kernel started on this queue
  void kernel_start() {
    TRISYCL_DUMP_T("A kernel has been added to the queue");
    // One more kernel
    ++running_kernels;
  }


  /// Signal that a new kernel finished on this queue
  void kernel_end() {
    TRISYCL_DUMP_T("A kernel of the queue ended");
    if (--running_kernels == 0) {
      /* It was the last kernel running, so signal the queue just in
         case it was working for it for completion */
      finished.notify_one();
    }
  }


#ifdef TRISYCL_OPENCL
  /** Return the underlying OpenCL command queue after doing a retain

      This memory object is expected to be released by the developer.

      Retain a reference to the returned cl_command_queue object.

      Caller should release it when finished.

      If the queue is a SYCL host queue then an exception is thrown.
  */
  virtual cl_command_queue get() const = 0;

  /// Return the underlying Boost.Compute command queue
  virtual boost::compute::command_queue &get_boost_compute() = 0;
#endif


  /** Return the SYCL queue's context

      Report errors using SYCL exception classes.
  */
  virtual cl::sycl::context get_context() const = 0;


  /** Return the SYCL device the queue is associated with

      Report errors using SYCL exception classes.
  */
  virtual cl::sycl::device get_device() const = 0;


  /// Return whether the queue is executing on a SYCL host device
  virtual bool is_host() const = 0;


  /// Wait for all kernel completion before the queue destruction
  /// \todo Update according spec since queue destruction is non blocking
  virtual ~queue() {
    wait_for_kernel_execution();
  }

};

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_QUEUE_DETAIL_QUEUE_HPP
```

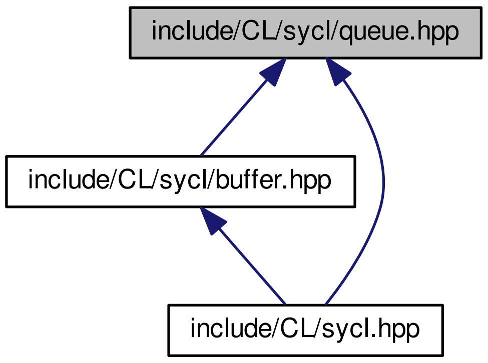
11.123 include/CL/sycl/queue.hpp File Reference

```
#include <memory>
#include <boost/compute.hpp>
#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/handler_event.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/parallelism.hpp"
#include "CL/sycl/queue/detail/host_queue.hpp"
#include "CL/sycl/queue/detail/opencl_queue.hpp"
```

Include dependency graph for queue.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class cl::sycl::queue

  SYCL queue, similar to the OpenCL queue concept. More...
- struct std::hash< cl::sycl::queue >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info
- std

Typedefs

- using cl::sycl::info::queue_profiling = bool

Enumerations

- enum cl::sycl::info::queue : int { cl::sycl::info::queue::context, cl::sycl::info::queue::device, cl::sycl::info↩ ::queue::reference_count, cl::sycl::info::queue::properties }

  Queue information descriptors.

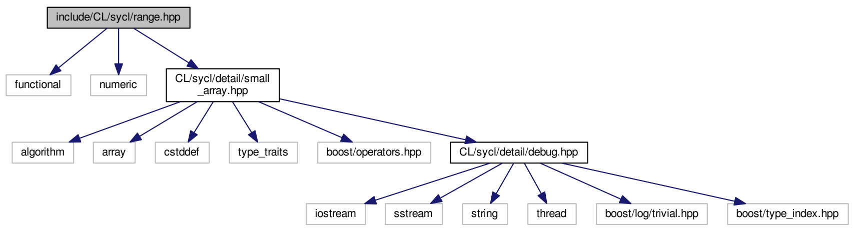
11.124 queue.hpp

```
#ifndef TRISYCL_SYCL_QUEUE_HPP
#define TRISYCL_SYCL_QUEUE_HPP

/** \file The OpenCL SYCL queue

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <memory>

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/handler_event.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/parallelism.hpp"
#include "CL/sycl/queue/detail/host_queue.hpp"
#ifdef TRISYCL_OPENCL
#include "CL/sycl/queue/detail/opencl_queue.hpp"
#endif

namespace cl {
namespace sycl {

class context;
class device_selector;

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

namespace info {

using queue_profiling = bool;

/** Queue information descriptors

    From specification C.4

    \todo unsigned int?

    \todo To be implemented
*/
enum class queue : int {
  context,
  device,
  reference_count,
  properties
};

/** Dummy example for get_info() on queue::context that would return a
    context

    \todo Describe all the types
*/
TRISYCL_INFO_PARAM_TRAITS(queue::context,
context)

}


/** SYCL queue, similar to the OpenCL queue concept.

    \todo The implementation is quite minimal for now. :-)

    \todo All the queue methods should return a queue& instead of void
    to it is possible to chain opoerations
*/
class queue
    /* Use the underlying queue implementation that can be shared in
       the SYCL model */
  : public detail::shared_ptr_implementation<queue, detail::queue>,
    detail::debug<queue> {
  // The type encapsulating the implementation
  using implementation_t =
    detail::shared_ptr_implementation<queue, detail::queue>
;

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

public:

  /** Default constructor for platform which is the host platform

      Returns errors via the SYCL exception class.
  */
  queue() : implementation_t { new detail::host_queue } {}


  /** This constructor creates a SYCL queue from an OpenCL queue

      At construction it does a retain on the queue memory object.

      Retain a reference to the cl_command_queue object. Caller should
      release the passed cl_command_queue object when it is no longer
      needed.

      Return synchronous errors regarding the creation of the queue and
      report asynchronous errors via the async_handler callback function
      in conjunction with the synchronization and throw methods.

      Note that the default case asyncHandler = nullptr is handled by the
      default constructor.

  */
  explicit queue(async_handler asyncHandler) : queue { } {
    detail::unimplemented();
  }


  /** Creates a queue for the device provided by the device selector

      If no device is selected, an error is reported.

      Return synchronous errors regarding the creation of the queue and
      report asynchronous errors via the async_handler callback
      function if and only if there is an async_handler provided.
  */
  queue(const device_selector &deviceSelector,
        async_handler asyncHandler = nullptr) : queue { } {
    detail::unimplemented();
  }


  /** A queue is created for syclDevice

      Return asynchronous errors via the async_handler callback function.
  */
  queue(const device &syclDevice,
        async_handler asyncHandler = nullptr) : queue { } {
    detail::unimplemented();
  };


  /** This constructor chooses a device based on the provided
      device_selector, which needs to be in the given context.

      If no device is selected, an error is reported.

      Return synchronous errors regarding the creation of the queue.

      If and only if there is an asyncHandler provided, it reports
      asynchronous errors via the async_handler callback function in
      conjunction with the synchronization and throw methods.
  */
  queue(const context &syclContext,
        const device_selector &deviceSelector,
        async_handler asyncHandler = nullptr) : queue { } {
    detail::unimplemented();
  }


  /** Creates a command queue using clCreateCommandQueue from a context
      and a device

      Return synchronous errors regarding the creation of the queue.

      If and only if there is an asyncHandler provided, it reports
      asynchronous errors via the async_handler callback function in
      conjunction with the synchronization and throw methods.
  */
  queue(const context &syclContext,
        const device &syclDevice,
        async_handler asyncHandler = nullptr) : queue { } {
    detail::unimplemented();
  }


  /** Creates a command queue using clCreateCommandQueue from a context
      and a device

      It enables profiling on the queue if the profilingFlag is set to
      true.

      Return synchronous errors regarding the creation of the queue. If
      and only if there is an asyncHandler provided, it reports
      asynchronous errors via the async_handler callback function in
      conjunction with the synchronization and throw methods.
  */
  queue(const context &syclContext,
        const device &syclDevice,
        info::queue_profiling profilingFlag,
        async_handler asyncHandler = nullptr) : queue { } {
    detail::unimplemented();
  }


#ifdef TRISYCL_OPENCL
  /** This constructor creates a SYCL queue from an OpenCL queue

      At construction it does a retain on the queue memory object.

      Return synchronous errors regarding the creation of the queue. If
      and only if there is an async_handler provided, it reports
      asynchronous errors via the async_handler callback function in
      conjunction with the synchronization and throw methods.
  */
  queue(const cl_command_queue &q, async_handler ah = nullptr)
    : queue { boost::compute::command_queue { q }, ah } {}


  /** Construct a queue instance using a boost::compute::command_queue

      This is a triSYCL extension for boost::compute interoperation.

      Return synchronous errors via the SYCL exception class.

      \todo Deal with handler
  */
  queue(const boost::compute::command_queue &q, async_handler ah = nullptr)
    : implementation_t { detail::opencl_queue::instance(q) }
{}
#endif


#ifdef TRISYCL_OPENCL
  /** Return the underlying OpenCL command queue after doing a retain

      This memory object is expected to be released by the developer.

      Retain a reference to the returned cl_command_queue object.

      Caller should release it when finished.

      If the queue is a SYCL host queue then an exception is thrown.
  */
  cl_command_queue get() const {
    return implementation->get();
  }
#endif


  /** Return the SYCL queue's context

      Report errors using SYCL exception classes.
  */
  context get_context() const {
    return implementation->get_context();
  }


  /** Return the SYCL device the queue is associated with

      Report errors using SYCL exception classes.
  */
  device get_device() const {
    return implementation->get_device();
  }


  /// Return whether the queue is executing on a SYCL host device
  bool is_host() const {
    return implementation->is_host();
  }


  /** Performs a blocking wait for the completion all enqueued tasks in
      the queue

      Synchronous errors will be reported through SYCL exceptions.
  */
  void wait() {
    implementation->wait_for_kernel_execution();
  }


  /** Perform a blocking wait for the completion all enqueued tasks in the queue

      Synchronous errors will be reported via SYCL exceptions.

      Asynchronous errors will be passed to the async_handler passed to the
      queue on construction.

      If no async_handler was provided then asynchronous exceptions will
      be lost.
  */
  void wait_and_throw() {
    detail::unimplemented();
  }


  /** Checks to see if any asynchronous errors have been produced by the
      queue and if so reports them by passing them to the async_handler
      passed to the queue on construction

      If no async_handler was provided then asynchronous exceptions will
      be lost.
  */
  void throw_asynchronous() {
    detail::unimplemented();
  }


  /// Queries the platform for cl_command_queue info
  template <info::queue param>
  typename info::param_traits<info::queue, param>::type
get_info() const  {
    detail::unimplemented();
    return {};
  }


  /** Submit a command group functor to the queue, in order to be
      scheduled for execution on the device

      Use an explicit functor parameter taking a handler& so we can use
      "auto" in submit() lambda parameter.

      \todo Add in the spec an implicit conversion of handler_event to
      queue& so it is possible to chain operations on the queue

      \todo Update the spec to replace std::function by a templated
      type to avoid memory allocation
  */
  handler_event submit(std::function<void(handler &)> cgf) {
    handler command_group_handler { implementation };
    cgf(command_group_handler);
    return {};
  }


  /** Submit a command group functor to the queue, in order to be
      scheduled for execution on the device

      On kernel error, this command group functor, then it is scheduled
      for execution on the secondary queue.

      Return a command group functor event, which is corresponds to the
      queue the command group functor is being enqueued on.
  */
  handler_event submit(std::function<void(handler &)> cgf,
queue &secondaryQueue) {
    detail::unimplemented();
    // Since it is not implemented, always submit on the main queue
    return submit(cgf);
  }

};

/// @} to end the execution Doxygen group

}
}

/* Inject a custom specialization of std::hash to have the buffer
   usable into an unordered associative container

   \todo Add this to the spec
*/
namespace std {

template <> struct hash<cl::sycl::queue> {

  auto operator()(const cl::sycl::queue &q) const {
    // Forward the hashing to the implementation
    return q.hash();
  }

};

}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_QUEUE_HPP
```

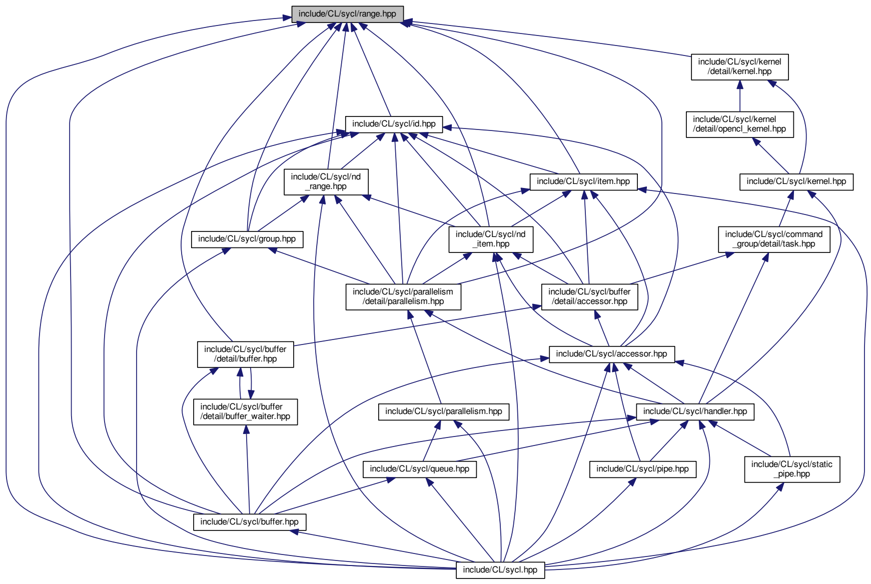
11.125 include/CL/sycl/range.hpp File Reference

```
#include <functional>
#include <numeric>
#include "CL/sycl/detail/small_array.hpp"
```

Include dependency graph for range.hpp:



This graph shows which files directly or indirectly include this file:


include/CL/sycl/range.hpp
include/CL/sycl/kernel/detail/kernel.hpp
include/CL/sycl/kernel/detail/opencl_kernel.hpp
include/CL/sycl/id.hpp
include/CL/sycl/nd_range.hpp
include/CL/sycl/item.hpp
include/CL/sycl/kernel.hpp
include/CL/sycl/group.hpp
include/CL/sycl/nd_item.hpp
include/CL/sycl/command_group/detail/task.hpp
include/CL/sycl/parallelism/detail/parallelism.hpp
include/CL/sycl/buffer/detail/accessor.hpp
include/CL/sycl/buffer/detail/buffer.hpp
include/CL/sycl/accessor.hpp
include/CL/sycl/buffer/detail/buffer_waiter.hpp
include/CL/sycl/parallelism.hpp
include/CL/sycl/handler.hpp
include/CL/sycl/queue.hpp
include/CL/sycl/pipe.hpp
include/CL/sycl/static_pipe.hpp
include/CL/sycl/buffer.hpp
include/CL/sycl.hpp

Classes

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

  A SYCL range defines a multi-dimensional index range that can be used to define launch parallel computation extent or buffer sizes. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Functions

- auto cl::sycl::make_range (range< 1 > r)

  Implement a make_range to construct a range<> of the right dimension with implicit conversion from an initializer list for example.
- auto cl::sycl::make_range (range< 2 > r)
- auto cl::sycl::make_range (range< 3 > r)
- template<typename... BasicType>
  auto cl::sycl::make_range (BasicType...Args)

  Construct a range<> from a function call with arguments, like make_range(1, 2, 3)

11.126 range.hpp

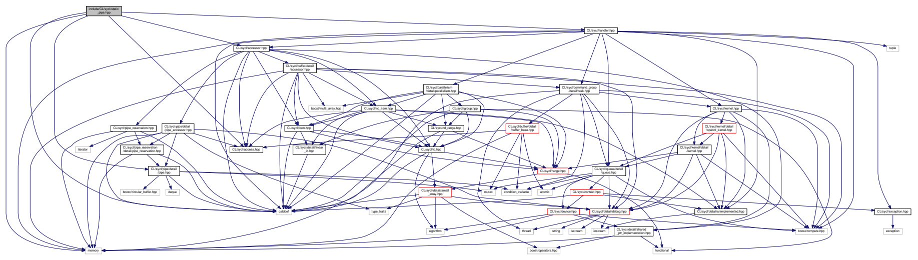
```cpp
#ifndef TRISYCL_SYCL_RANGE_HPP
#define TRISYCL_SYCL_RANGE_HPP

/** \file The OpenCL SYCL range<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <functional>
#include <numeric>
#include "CL/sycl/detail/small_array.hpp"

namespace cl {
namespace sycl {

/** \addtogroup parallelism Expressing parallelism through kernels
    @{
*/

/** A SYCL range defines a multi-dimensional index range that can be used
    to define launch parallel computation extent or buffer sizes.

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

    \todo add to the specification this default parameter value?

    \todo add to the specification some way to specify an offset?
*/
template <std::size_t dims = 1>
class range : public detail::small_array_123<std::size_t, range<dims>, dims> {

public:

  // Inherit of all the constructors
  using detail::small_array_123<std::size_t,
                                range<dims>,
                                dims>::small_array_123;


  /** Return the number of elements in the range

      \todo Give back size() its real meaning in the specification

      \todo add this method to the specification
  */
  size_t get_count() {
    // Return the product of the sizes in each dimension
    return std::accumulate(this->cbegin(),
                           this->cend(),
                           1,
                           std::multiplies<size_t> {});
  }
};


/** Implement a make_range to construct a range<> of the right dimension
    with implicit conversion from an initializer list for example.

    Cannot use a template on the number of dimensions because the implicit
    conversion would not be tried.
*/
inline auto make_range(range<1> r) { return r; }
inline auto make_range(range<2> r) { return r; }
inline auto make_range(range<3> r) { return r; }


/** Construct a range<> from a function call with arguments, like
    make_range(1, 2, 3)
*/
template<typename... BasicType>
auto make_range(BasicType... Args) {
  // Call constructor directly to allow narrowing
  return range<sizeof...(Args)>(Args...);
}

/// @} End the parallelism Doxygen group

}
}

/*
```

```
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_RANGE_HPP
```

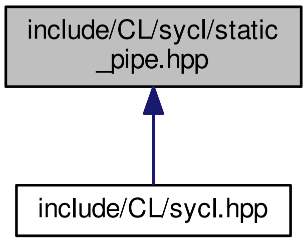
11.127 include/CL/sycl/static_pipe.hpp File Reference

```
#include <cstddef>
#include <memory>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/pipe/detail/pipe.hpp"
```

Include dependency graph for static_pipe.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class cl::sycl::static_pipe< T, Capacity >

    A SYCL static-scoped pipe equivalent to an OpenCL program-scoped pipe. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.128 static_pipe.hpp

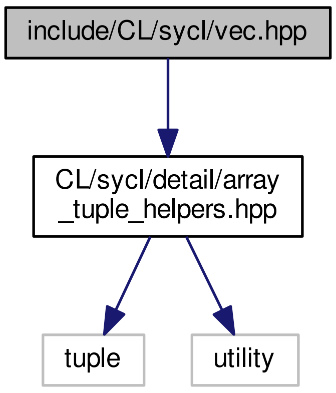
```
#ifndef TRISYCL_SYCL_STATIC_PIPE_HPP
#define TRISYCL_SYCL_STATIC_PIPE_HPP

/** \file The OpenCL SYCL static-scoped pipe equivalent to an OpenCL
    program-scoped pipe

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <memory>

#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/pipe/detail/pipe.hpp"

namespace cl {
namespace sycl {

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** A SYCL static-scoped pipe equivalent to an OpenCL program-scoped
    pipe

    Implement a FIFO-style object that can be used through accessors
    to send some objects T from the input to the output.

    Compared to a normal pipe, a static_pipe takes a constexpr size
    and is expected to be declared in a compile-unit static context so
    the compiler can generate everything at compile time.

    This is useful to generate a fixed and optimized hardware
    implementation on FPGA for example, where the interconnection
    graph can be also inferred at compile time.

    It is not directly mapped to the OpenCL program-scoped pipe
    because in SYCL there is not this concept of separated
    program. But the SYCL device compiler is expected to generate some
    OpenCL program(s) with program-scoped pipes when a SYCL
    static-scoped pipe is used. These details are implementation
    defined.
*/
template <typename T, std::size_t Capacity>
class static_pipe
    /* Use the underlying pipe implementation that can be shared in
       the SYCL model */
  : public detail::shared_ptr_implementation<static_pipe<T, Capacity>,
                                             detail::pipe<T>>,
    detail::debug<static_pipe<T, Capacity>> {

  // The type encapsulating the implementation
  using implementation_t =
    detail::shared_ptr_implementation<static_pipe<T, Capacity>
,
                                      detail::pipe<T>>;

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

public:

  /// The STL-like types
  using value_type = T;


  /// Construct a static-scoped pipe able to store up to Capacity T objects
  static_pipe()
    : implementation_t { new detail::pipe<T> { Capacity } } { }


  /** Get an accessor to the pipe with the required mode

      \param Mode is the requested access mode

      \param Target is the type of pipe access required

      \param[in] command_group_handler is the command group handler in
      which the kernel is to be executed
  */
  template <access::mode Mode,
            access::target Target = access::target::pipe>
  accessor<value_type, 1, Mode, Target>
  get_access(handler &command_group_handler) {
    static_assert(Target == access::target::pipe
                  || Target == access::target::blocking_pipe,
                  "get_access(handler) with pipes can only deal with "
                  "access::pipe or access::blocking_pipe");
    return { implementation, command_group_handler };
  }


  /** Return the maximum number of elements that can fit in the pipe

      This is a constexpr since the capacity is in the type.
  */
  std::size_t constexpr capacity() const {
    return Capacity;
    }

};

/// @} End the execution Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_STATIC_PIPE_HPP
```

11.129 include/CL/sycl/vec.hpp File Reference

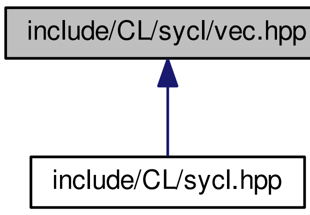
Implement the small OpenCL vector class.

```
#include "CL/sycl/detail/array_tuple_helpers.hpp"
```

Include dependency graph for vec.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class cl::sycl::vec< DataType, NumElements >

  Small OpenCL vector class. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Macros

- #define TRISYCL_DEFINE_VEC_TYPE_SIZE(type, size, actual_type) using type##size = vec<actual_type, size>;

  A macro to define type alias, such as for type=uchar, size=4 and real_type=unsigned char, uchar4 is equivalent to vec<float, 4>
- #define TRISYCL_DEFINE_VEC_TYPE(type, actual_type)

  Declare the vector types of a type for all the sizes.

11.129.1 Detailed Description

Implement the small OpenCL vector class.

Ronan at Keryell point FR

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file vec.hpp.

11.130 vec.hpp

```
#ifndef TRISYCL_SYCL_VEC_HPP
#define TRISYCL_SYCL_VEC_HPP

/** \file

    Implement the small OpenCL vector class

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/detail/array_tuple_helpers.hpp"

namespace cl {
namespace sycl {

/** \addtogroup vector Vector types in SYCL

    @{
*/


/** Small OpenCL vector class

    \todo add [] operator

    \todo add iterators on elements, with begin() and end()

    \todo having vec<> sub-classing array<> instead would solve the
    previous issues

    \todo move the implementation elsewhere

    \todo simplify the helpers by removing some template types since there
    are now inside the vec<> class.

    \todo rename in the specification element_type to value_type
*/
template <typename DataType, size_t NumElements>
class vec : public detail::small_array<DataType,
                                       vec<DataType, NumElements>,
                                       NumElements> {
  using basic_type = typename detail::small_array<DataType,
                                                  vec<DataType, NumElements>,
                                                  NumElements>;

public:

  /** Construct a vec from anything from a scalar (to initialize all the
      elements with this value) up to an aggregate of scalar and vector
      types (in this case the total number of elements must match the size
      of the vector)
  */
  template <typename... Types>
  vec(const Types... args)
    : basic_type { detail::expand<vec>(flatten_to_tuple<vec>(args...)) } { }


/// Use classical constructors too
  vec() = default;


  // Inherit of all the constructors
  using typename basic_type::small_array;

private:

  /** Flattening helper that does not change scalar values but flatten a
      vec<T, n> v into a tuple<T, T,..., T>{ v[0], v[1],..., v[n-1] }

      If we have a vector, just forward its array content since an array
      has also a tuple interface :-) (23.3.2.9 Tuple interface to class
      template array [array.tuple])
  */
  template <typename V, typename Element, size_t s>
  static auto flatten(const vec<Element, s> i) {
    static_assert(s <= V::dimension,
                  "The element i will not fit in the vector");
    return static_cast<std::array<Element, s>>(i);
  }


  /** If we do not have a vector, just forward it as a tuple up to the
      final initialization.

      \return typically tuple<double>{ 2.4 } from 2.4 input for example
  */
  template <typename V, typename Type>
  static auto flatten(const Type i) {
    return std::make_tuple(i);
  }


 /** Take some initializer values and apply flattening on each value

      \return a tuple of scalar initializer values
   */
  template <typename V, typename... Types>
  static auto flatten_to_tuple(const Types... i) {
    // Concatenate the tuples returned by each flattening
    return std::tuple_cat(flatten<V>(i)...);
  }


  /// \todo To implement
#if 0
  vec<dataT,
      numElements>
  operator+(const vec<dataT, numElements> &rhs) const;
  vec<dataT, numElements>
  operator-(const vec<dataT, numElements> &rhs) const;
  vec<dataT, numElements>
  operator*(const vec<dataT, numElements> &rhs) const;
  vec<dataT, numElements>
  operator/(const vec<dataT, numElements> &rhs) const;
  vec<dataT, numElements>
  operator+=(const vec<dataT, numElements> &rhs);
  vec<dataT, numElements>
  operator-=(const vec<dataT, numElements> &rhs);
  vec<dataT, numElements>
  operator*=(const vec<dataT, numElements> &rhs);
  vec<dataT, numElements>
  operator/=(const vec<dataT, numElements> &rhs);
  vec<dataT, numElements>
  operator+(const dataT &rhs) const;
  vec<dataT, numElements>
  operator-(const dataT &rhs) const;
  vec<dataT, numElements>
  operator*(const dataT &rhs) const;
  vec<dataT, numElements>
  operator/(const dataT &rhs) const;
  vec<dataT, numElements>
  operator+=(const dataT &rhs);
  vec<dataT, numElements>
  operator-=(const dataT &rhs);
  vec<dataT, numElements>
  operator*=(const dataT &rhs);
  vec<dataT, numElements>
  operator/=(const dataT &rhs);
  vec<dataT, numElements> &operator=(const
vec<dataT, numElements> &rhs);
  vec<dataT, numElements> &operator=(const dataT &rhs);
  bool operator==(const vec<dataT, numElements> &rhs) const;
  bool operator!=(const vec<dataT, numElements> &rhs) const;
  // Swizzle methods (see notes)
  swizzled_vec<T, out_dims> swizzle<int s1, ...>();
#ifdef SYCL_SIMPLE_SWIZZLES
  swizzled_vec<T, 4> xyzw();
  ...
#endif // #ifdef SYCL_SIMPLE_SWIZZLES
#endif
};

  /** A macro to define type alias, such as for type=uchar, size=4 and
      real_type=unsigned char, uchar4 is equivalent to vec<float, 4>
  */
#define TRISYCL_DEFINE_VEC_TYPE_SIZE(type, size, actual_type) \
  using type##size = vec<actual_type, size>;

  /// Declare the vector types of a type for all the sizes
#define TRISYCL_DEFINE_VEC_TYPE(type, actual_type)           \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 1, actual_type)         \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 2, actual_type)         \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 3, actual_type)         \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 4, actual_type)         \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 8, actual_type)         \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 16, actual_type)

  /// Declare all the possible vector type aliases
  TRISYCL_DEFINE_VEC_TYPE(char, char)
  TRISYCL_DEFINE_VEC_TYPE(uchar, unsigned char)
  TRISYCL_DEFINE_VEC_TYPE(short, short int)
  TRISYCL_DEFINE_VEC_TYPE(ushort, unsigned short int)
  TRISYCL_DEFINE_VEC_TYPE(int, int)
  TRISYCL_DEFINE_VEC_TYPE(uint, unsigned int)
  TRISYCL_DEFINE_VEC_TYPE(long, long int)
  TRISYCL_DEFINE_VEC_TYPE(ulong, unsigned long int)
  TRISYCL_DEFINE_VEC_TYPE(float, float)
  TRISYCL_DEFINE_VEC_TYPE(double, double)

/// @} End the vector Doxygen group


}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_VEC_HPP
```

Index