

triSYCL implementation of OpenCL SYCL

Generated by Doxygen 1.8.11

Contents

1	Main Page	1
2	Todo List	3
3	Module Index	13
3.1	Modules	13
4	Namespace Index	15
4.1	Namespace List	15
5	Hierarchical Index	17
5.1	Class Hierarchy	17
6	Class Index	23
6.1	Class List	23
7	File Index	25
7.1	File List	25
8	Module Documentation	27
8.1	Data access and storage in SYCL	27
8.1.1	Detailed Description	28
8.1.2	Class Documentation	28
8.1.2.1	class <code>cl::sycl::accessor</code>	28
8.1.2.2	class <code>cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe ></code>	39
8.1.2.3	class <code>cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_↔ pipe ></code>	42
8.1.2.4	class <code>cl::sycl::detail::accessor</code>	44

8.1.2.5	class cl::sycl::detail::buffer	61
8.1.2.6	class cl::sycl::detail::buffer_waiter	69
8.1.2.7	struct cl::sycl::image	71
8.1.2.8	struct cl::sycl::detail::reserve_id	71
8.1.2.9	class cl::sycl::detail::pipe	72
8.1.2.10	class cl::sycl::detail::pipe_accessor	84
8.1.2.11	class cl::sycl::pipe	93
8.1.2.12	class cl::sycl::detail::pipe_reservation	96
8.1.2.13	struct cl::sycl::pipe_reservation	106
8.1.2.14	class cl::sycl::static_pipe	115
8.1.3	Typedef Documentation	118
8.1.3.1	buffer_allocator	118
8.1.4	Function Documentation	118
8.1.4.1	buffer_add_to_task(BufferDetail buf, handler *command_group_handler, bool is_write_mode)	118
8.1.4.2	get_pipe_detail(Accessor &a)	119
8.1.4.3	waiter(detail::buffer< T, Dimensions > *b)	120
8.2	Dealing with OpenCL address spaces	121
8.2.1	Detailed Description	122
8.2.2	Class Documentation	122
8.2.2.1	struct cl::sycl::detail::opencl_type	122
8.2.2.2	struct cl::sycl::detail::opencl_type< T, constant_address_space >	123
8.2.2.3	struct cl::sycl::detail::opencl_type< T, generic_address_space >	123
8.2.2.4	struct cl::sycl::detail::opencl_type< T, global_address_space >	124
8.2.2.5	struct cl::sycl::detail::opencl_type< T, local_address_space >	124
8.2.2.6	struct cl::sycl::detail::opencl_type< T, private_address_space >	124
8.2.2.7	struct cl::sycl::detail::address_space_array	125
8.2.2.8	struct cl::sycl::detail::address_space_fundamental	127
8.2.2.9	struct cl::sycl::detail::address_space_object	130
8.2.2.10	struct cl::sycl::detail::address_space_ptr	132
8.2.2.11	struct cl::sycl::detail::address_space_base	135

8.2.2.12	struct cl::sycl::detail::address_space_variable	137
8.2.3	Typedef Documentation	140
8.2.3.1	addr_space	140
8.2.3.2	constant	140
8.2.3.3	generic	141
8.2.3.4	global	141
8.2.3.5	local	141
8.2.3.6	multi_ptr	141
8.2.3.7	priv	142
8.2.4	Enumeration Type Documentation	142
8.2.4.1	address_space	142
8.2.5	Function Documentation	142
8.2.5.1	make_multi(multi_ptr< T, AS > pointer)	142
8.3	Platforms, contexts, devices and queues	144
8.3.1	Detailed Description	147
8.3.2	Class Documentation	147
8.3.2.1	class cl::sycl::context	147
8.3.2.2	class cl::sycl::detail::device	153
8.3.2.3	class cl::sycl::device	155
8.3.2.4	class cl::sycl::device_type_selector	163
8.3.2.5	class cl::sycl::device_typename_selector	166
8.3.2.6	class cl::sycl::device_selector	167
8.3.2.7	class cl::sycl::handler	170
8.3.2.8	class cl::sycl::detail::kernel	179
8.3.2.9	class cl::sycl::kernel	180
8.3.2.10	class cl::sycl::detail::host_platform	183
8.3.2.11	class cl::sycl::detail::opencl_platform	186
8.3.2.12	class cl::sycl::detail::platform	191
8.3.2.13	class cl::sycl::platform	192
8.3.2.14	class cl::sycl::queue	198

8.3.3	Typedef Documentation	207
8.3.3.1	cpu_selector	207
8.3.3.2	default_selector	207
8.3.3.3	device_exec_capabilities	207
8.3.3.4	device_fp_config	207
8.3.3.5	device_queue_properties	207
8.3.3.6	gpu_selector	207
8.3.3.7	host_selector	208
8.3.4	Enumeration Type Documentation	208
8.3.4.1	device	208
8.3.4.2	device_affinity_domain	211
8.3.4.3	device_execution_capabilities	211
8.3.4.4	device_partition_property	211
8.3.4.5	device_partition_type	212
8.3.4.6	device_type	212
8.3.4.7	fp_config	213
8.3.4.8	global_mem_cache_type	213
8.3.4.9	local_mem_type	214
8.3.4.10	platform	214
8.3.5	Function Documentation	215
8.3.5.1	__attribute__((weak))	215
8.3.5.2	get_devices(info::device_type device_type=info::device_type::all) __attribute__((weak))	215
8.3.6	Variable Documentation	216
8.3.6.1	__attribute__	216
8.4	Helpers to do array and tuple conversion	217
8.4.1	Detailed Description	217
8.4.2	Class Documentation	217
8.4.2.1	struct cl::sycl::detail::expand_to_vector	217
8.4.2.2	struct cl::sycl::detail::expand_to_vector< V, Tuple, true >	218
8.4.3	Function Documentation	218

8.4.3.1	expand(Tuple t)	218
8.4.3.2	expand(Tuple t)	218
8.4.3.3	expand(Tuple t)	219
8.4.3.4	fill_tuple(Value e, std::index_sequence< Is... >)	219
8.4.3.5	tuple_to_array(Tuple t)	220
8.4.3.6	tuple_to_array_iterate(Tuple t, std::index_sequence< Is... >)	220
8.5	Debugging and tracing support	221
8.5.1	Detailed Description	221
8.5.2	Class Documentation	221
8.5.2.1	struct cl::sycl::detail::debug	221
8.5.2.2	struct cl::sycl::detail::display_vector	221
8.5.3	Function Documentation	222
8.5.3.1	trace_kernel(const Functor &f)	222
8.6	Manage default configuration and types	224
8.6.1	Detailed Description	224
8.6.2	Macro Definition Documentation	224
8.6.2.1	__SYCL_SINGLE_SOURCE__	224
8.6.2.2	CL_SYCL_LANGUAGE_VERSION	224
8.6.2.3	CL_TRISYCL_LANGUAGE_VERSION	225
8.6.2.4	TRISYCL_ASYNC	225
8.6.2.5	TRISYCL_SKIP_OPENCL	225
8.7	Some helpers for the implementation	226
8.7.1	Detailed Description	226
8.7.2	Class Documentation	226
8.7.2.1	struct cl::sycl::detail::small_array	226
8.7.2.2	struct cl::sycl::detail::small_array_123	231
8.7.2.3	struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >	231
8.7.2.4	struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >	233
8.7.2.5	struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >	235
8.7.3	Macro Definition Documentation	237

8.7.3.1	TRISYCL_BOOST_OPERATOR_VECTOR_OP	237
8.7.4	Function Documentation	237
8.7.4.1	linear_id(Range range, Id id, Id offset={})	237
8.7.4.2	unimplemented()	238
8.8	Error handling	240
8.8.1	Detailed Description	241
8.8.2	Class Documentation	241
8.8.2.1	struct cl::sycl::error_handler	241
8.8.2.2	struct cl::sycl::exception_list	242
8.8.2.3	class cl::sycl::exception	243
8.8.2.4	class cl::sycl::cl_exception	246
8.8.2.5	struct cl::sycl::async_exception	248
8.8.2.6	class cl::sycl::runtime_error	248
8.8.2.7	class cl::sycl::kernel_error	249
8.8.2.8	class cl::sycl::accessor_error	250
8.8.2.9	class cl::sycl::nd_range_error	252
8.8.2.10	class cl::sycl::event_error	253
8.8.2.11	class cl::sycl::invalid_parameter_error	254
8.8.2.12	class cl::sycl::device_error	255
8.8.2.13	class cl::sycl::compile_program_error	256
8.8.2.14	class cl::sycl::link_program_error	257
8.8.2.15	class cl::sycl::invalid_object_error	258
8.8.2.16	class cl::sycl::memory_allocation_error	259
8.8.2.17	class cl::sycl::pipe_error	260
8.8.2.18	class cl::sycl::platform_error	261
8.8.2.19	class cl::sycl::profiling_error	262
8.8.2.20	class cl::sycl::feature_not_supported	263
8.8.2.21	class cl::sycl::non_cl_error	264
8.8.3	Typedef Documentation	265
8.8.3.1	async_handler	265

8.8.3.2	<code>exception_ptr</code>	265
8.9	Expressing parallelism through kernels	266
8.9.1	Detailed Description	267
8.9.2	Class Documentation	267
8.9.2.1	<code>struct cl::sycl::group</code>	267
8.9.2.2	<code>class cl::sycl::id</code>	276
8.9.2.3	<code>class cl::sycl::item</code>	278
8.9.2.4	<code>struct cl::sycl::nd_item</code>	284
8.9.2.5	<code>struct cl::sycl::nd_range</code>	300
8.9.2.6	<code>struct cl::sycl::detail::parallel_for_iterate</code>	303
8.9.2.7	<code>struct cl::sycl::detail::parallel_OpenMP_for_iterate</code>	304
8.9.2.8	<code>struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFuncor, Id ></code>	305
8.9.2.9	<code>class cl::sycl::range</code>	306
8.9.3	Function Documentation	307
8.9.3.1	<code>make_id(id< 1 > i)</code>	307
8.9.3.2	<code>make_id(id< 2 > i)</code>	307
8.9.3.3	<code>make_id(id< 3 > i)</code>	308
8.9.3.4	<code>make_id(BasicType...Args)</code>	308
8.9.3.5	<code>make_range(range< 1 > r)</code>	308
8.9.3.6	<code>make_range(range< 2 > r)</code>	308
8.9.3.7	<code>make_range(range< 3 > r)</code>	308
8.9.3.8	<code>make_range(BasicType...Args)</code>	309
8.9.3.9	<code>parallel_for(range< Dimensions > r, ParallelForFuncor f, Id)</code>	309
8.9.3.10	<code>parallel_for(range< Dimensions > r, ParallelForFuncor f, item< Dimensions >)</code>	310
8.9.3.11	<code>parallel_for(range< Dimensions > r, ParallelForFuncor f)</code>	310
8.9.3.12	<code>parallel_for(nd_range< Dimensions > r, ParallelForFuncor f)</code>	311
8.9.3.13	<code>parallel_for_global_offset(range< Dimensions > global_size, id< Dimensions > offset, ParallelForFuncor f)</code>	312
8.9.3.14	<code>parallel_for_work_item(const group< Dimensions > &g, ParallelForFuncor f)</code>	313
8.9.3.15	<code>parallel_for_workgroup(nd_range< Dimensions > r, ParallelForFuncor f)</code>	313
8.9.3.16	<code>parallel_for_workitem(const group< Dimensions > &g, ParallelForFuncor f)</code>	314
8.10	Vector types in SYCL	317
8.10.1	Detailed Description	317
8.10.2	Class Documentation	317
8.10.2.1	<code>class cl::sycl::vec</code>	317
8.10.3	Macro Definition Documentation	321
8.10.3.1	<code>TRISYCL_DEFINE_VEC_TYPE</code>	321
8.10.3.2	<code>TRISYCL_DEFINE_VEC_TYPE_SIZE</code>	321

9	Namespace Documentation	323
9.1	cl Namespace Reference	323
9.1.1	Detailed Description	323
9.2	cl::sycl Namespace Reference	323
9.2.1	Typedef Documentation	327
9.2.1.1	function_class	327
9.2.1.2	mutex_class	327
9.2.1.3	shared_ptr_class	327
9.2.1.4	string_class	327
9.2.1.5	unique_ptr_class	328
9.2.1.6	vector_class	328
9.2.1.7	weak_ptr_class	328
9.3	cl::sycl::access Namespace Reference	328
9.3.1	Detailed Description	328
9.3.2	Enumeration Type Documentation	329
9.3.2.1	fence_space	329
9.3.2.2	mode	329
9.3.2.3	target	330
9.4	cl::sycl::detail Namespace Reference	330
9.4.1	Function Documentation	334
9.4.1.1	add_buffer_to_task(handler *command_group_handler, std::shared_ptr<detail::buffer_base > b, bool is_write_mode)	334
9.4.1.2	add_buffer_to_task(handler *command_group_handler, std::shared_ptr<detail::buffer_base > b, bool is_write_mode)	334
9.5	cl::sycl::info Namespace Reference	334
9.5.1	Typedef Documentation	336
9.5.1.1	gl_context_interop	336
9.5.1.2	queue_profiling	336
9.5.2	Enumeration Type Documentation	336
9.5.2.1	context	336
9.5.2.2	queue	337
9.6	cl::sycl::trisycl Namespace Reference	337
9.6.1	Detailed Description	337
9.7	std Namespace Reference	337

10 Class Documentation	339
10.1 <code>cl::sycl::buffer< T, Dimensions, Allocator ></code> Class Template Reference	339
10.1.1 Detailed Description	341
10.1.2 Member Typedef Documentation	341
10.1.2.1 <code>allocator_type</code>	341
10.1.2.2 <code>const_reference</code>	341
10.1.2.3 <code>implementation_t</code>	342
10.1.2.4 <code>reference</code>	342
10.1.2.5 <code>value_type</code>	342
10.1.3 Constructor & Destructor Documentation	342
10.1.3.1 <code>buffer()</code> =default	342
10.1.3.2 <code>buffer(const range< Dimensions > &r, Allocator allocator={})</code>	342
10.1.3.3 <code>buffer(const T *host_data, const range< Dimensions > &r, Allocator allocator={})</code>	343
10.1.3.4 <code>buffer(T *host_data, const range< Dimensions > &r, Allocator allocator={})</code>	344
10.1.3.5 <code>buffer(shared_ptr_class< T > &host_data, const range< Dimensions > &buffer_range, cl::sycl::mutex_class &m, Allocator allocator={})</code>	344
10.1.3.6 <code>buffer(shared_ptr_class< T > host_data, const range< Dimensions > &buffer_range, Allocator allocator={})</code>	345
10.1.3.7 <code>buffer(unique_ptr_class< T, D > &&host_data, const range< Dimensions > &buffer_range, Allocator allocator={})</code>	346
10.1.3.8 <code>buffer(InputIterator start_iterator, InputIterator end_iterator, Allocator allocator={})</code>	347
10.1.3.9 <code>buffer(buffer< T, Dimensions, Allocator > &b, const id< Dimensions > &base_index, const range< Dimensions > &sub_range, Allocator allocator={})</code>	348
10.1.3.10 <code>buffer(cl_mem mem_object, queue from_queue, event available_event={}, Allocator allocator={})</code>	348
10.1.4 Member Function Documentation	349
10.1.4.1 <code>get_access(handler &command_group_handler)</code>	349
10.1.4.2 <code>get_access()</code>	350
10.1.4.3 <code>get_count() const</code>	350
10.1.4.4 <code>get_range() const</code>	351
10.1.4.5 <code>get_size() const</code>	351
10.1.4.6 <code>is_read_only() const</code>	351
10.1.4.7 <code>set_final_data(weak_ptr_class< T > finalData)</code>	352

10.1.4.8	<code>use_count() const</code>	352
10.2	<code>cl::sycl::detail::buffer_base</code> Struct Reference	353
10.2.1	Detailed Description	354
10.2.2	Constructor & Destructor Documentation	354
10.2.2.1	<code>buffer_base(bool read_only)</code>	354
10.2.2.2	<code>~buffer_base()</code>	355
10.2.3	Member Function Documentation	355
10.2.3.1	<code>add_to_task(handler *command_group_handler, bool is_write_mode)</code>	355
10.2.3.2	<code>get_latest_producer()</code>	356
10.2.3.3	<code>release()</code>	356
10.2.3.4	<code>set_latest_producer(std::weak_ptr< detail::task > newer_latest_producer)</code>	356
10.2.3.5	<code>use()</code>	356
10.2.3.6	<code>wait()</code>	357
10.2.4	Member Data Documentation	357
10.2.4.1	<code>latest_producer</code>	357
10.2.4.2	<code>latest_producer_mutex</code>	357
10.2.4.3	<code>notify_buffer_destructor</code>	357
10.2.4.4	<code>number_of_users</code>	358
10.2.4.5	<code>read_only</code>	358
10.2.4.6	<code>ready</code>	358
10.2.4.7	<code>ready_mutex</code>	358
10.3	<code>cl::sycl::detail::cache< Key, Value ></code> Class Template Reference	358
10.3.1	Detailed Description	359
10.3.2	Member Typedef Documentation	359
10.3.2.1	<code>key_type</code>	359
10.3.2.2	<code>value_type</code>	359
10.3.3	Member Function Documentation	359
10.3.3.1	<code>get_or_register(const key_type &k, Functor &&create_element)</code>	359
10.3.3.2	<code>remove(const key_type &k)</code>	360
10.3.4	Member Data Documentation	361

10.3.4.1	<code>c</code>	361
10.3.4.2	<code>m</code>	361
10.4	<code>cl::sycl::trisycl::default_error_handler</code> Struct Reference	362
10.4.1	Detailed Description	362
10.4.2	Member Function Documentation	363
10.4.2.1	<code>report_error(exception &) override</code>	363
10.5	<code>cl::sycl::event</code> Class Reference	363
10.5.1	Detailed Description	363
10.5.2	Constructor & Destructor Documentation	363
10.5.2.1	<code>event()=default</code>	363
10.6	<code>handler_event</code> Class Reference	363
10.6.1	Detailed Description	364
10.7	<code>std::hash< cl::sycl::buffer< T, Dimensions, Allocator > ></code> Struct Template Reference	364
10.7.1	Detailed Description	364
10.7.2	Member Function Documentation	364
10.7.2.1	<code>operator()(const cl::sycl::buffer< T, Dimensions, Allocator > &b) const</code>	364
10.8	<code>std::hash< cl::sycl::device ></code> Struct Template Reference	365
10.8.1	Detailed Description	365
10.8.2	Member Function Documentation	365
10.8.2.1	<code>operator()(const cl::sycl::device &d) const</code>	365
10.9	<code>std::hash< cl::sycl::kernel ></code> Struct Template Reference	366
10.9.1	Detailed Description	366
10.9.2	Member Function Documentation	366
10.9.2.1	<code>operator()(const cl::sycl::kernel &k) const</code>	366
10.10	<code>std::hash< cl::sycl::platform ></code> Struct Template Reference	367
10.10.1	Detailed Description	367
10.10.2	Member Function Documentation	367
10.10.2.1	<code>operator()(const cl::sycl::platform &p) const</code>	367
10.11	<code>std::hash< cl::sycl::queue ></code> Struct Template Reference	368
10.11.1	Detailed Description	368

10.11.2 Member Function Documentation	368
10.11.2.1 operator()(const cl::sycl::queue &q) const	368
10.12cl::sycl::detail::host_device Class Reference	369
10.12.1 Detailed Description	370
10.12.2 Member Function Documentation	371
10.12.2.1 get() const override	371
10.12.2.2 get_platform() const override	371
10.12.2.3 has_extension(const string_class &extension) const override	372
10.12.2.4 is_accelerator() const override	372
10.12.2.5 is_cpu() const override	372
10.12.2.6 is_gpu() const override	373
10.12.2.7 is_host() const override	373
10.13cl::sycl::detail::host_queue Class Reference	373
10.13.1 Detailed Description	374
10.13.2 Member Function Documentation	374
10.13.2.1 get() const override	374
10.13.2.2 get_boost_compute() override	375
10.13.2.3 get_context() const override	375
10.13.2.4 get_device() const override	375
10.13.2.5 is_host() const override	375
10.14cl::sycl::detail::opencl_device Class Reference	376
10.14.1 Detailed Description	377
10.14.2 Constructor & Destructor Documentation	377
10.14.2.1 opencl_device(const boost::compute::device &d)	377
10.14.2.2 ~opencl_device() override	377
10.14.3 Member Function Documentation	378
10.14.3.1 get() const override	378
10.14.3.2 get_platform() const override	378
10.14.3.3 has_extension(const string_class &extension) const override	379
10.14.3.4 instance(const boost::compute::device &d)	379

10.14.3.5 <code>is_accelerator()</code> const override	380
10.14.3.6 <code>is_cpu()</code> const override	380
10.14.3.7 <code>is_gpu()</code> const override	380
10.14.3.8 <code>is_host()</code> const override	381
10.14.4 Member Data Documentation	381
10.14.4.1 <code>cache</code>	381
10.14.4.2 <code>d</code>	381
10.15 <code>cl::sycl::detail::opencl_kernel</code> Class Reference	382
10.15.1 Detailed Description	383
10.15.2 Constructor & Destructor Documentation	383
10.15.2.1 <code>opencl_kernel(const boost::compute::kernel &k)</code>	383
10.15.3 Member Function Documentation	383
10.15.3.1 <code>get()</code> const override	383
10.15.3.2 <code>get_boost_compute()</code> const override	384
10.15.3.3 <code>instance(const boost::compute::kernel &k)</code>	384
10.15.3.4 <code>TRISYCL_ParallelForKernel_RANGE(1) TRISYCL_ParallelForKernel_RANGE(2)</code> <code>TRISYCL_ParallelForKernel_RANGE(3)~opencl_kernel()</code> override	385
10.15.4 Member Data Documentation	385
10.15.4.1 <code>cache</code>	385
10.15.4.2 <code>k</code>	386
10.16 <code>cl::sycl::detail::opencl_queue</code> Class Reference	386
10.16.1 Detailed Description	388
10.16.2 Constructor & Destructor Documentation	388
10.16.2.1 <code>opencl_queue(const boost::compute::command_queue &q)</code>	388
10.16.2.2 <code>~opencl_queue()</code> override	388
10.16.3 Member Function Documentation	389
10.16.3.1 <code>get()</code> const override	389
10.16.3.2 <code>get_boost_compute()</code> override	389
10.16.3.3 <code>get_context()</code> const override	389
10.16.3.4 <code>get_device()</code> const override	390
10.16.3.5 <code>instance(const boost::compute::command_queue &q)</code>	390

10.16.3.6 <code>is_host()</code> const override	391
10.16.4 Member Data Documentation	391
10.16.4.1 <code>cache</code>	391
10.16.4.2 <code>q</code>	391
10.17 <code>cl::sycl::info::param_traits< T, Param ></code> Struct Template Reference	391
10.17.1 Detailed Description	392
10.18 <code>cl::sycl::detail::queue</code> Struct Reference	392
10.18.1 Detailed Description	393
10.18.2 Constructor & Destructor Documentation	393
10.18.2.1 <code>queue()</code>	393
10.18.2.2 <code>~queue()</code>	394
10.18.3 Member Function Documentation	394
10.18.3.1 <code>get()</code> const =0	394
10.18.3.2 <code>get_boost_compute()</code> =0	395
10.18.3.3 <code>get_context()</code> const =0	395
10.18.3.4 <code>get_device()</code> const =0	396
10.18.3.5 <code>is_host()</code> const =0	396
10.18.3.6 <code>kernel_end()</code>	397
10.18.3.7 <code>kernel_start()</code>	397
10.18.3.8 <code>wait_for_kernel_execution()</code>	398
10.18.4 Member Data Documentation	398
10.18.4.1 <code>finished</code>	398
10.18.4.2 <code>finished_mutex</code>	398
10.18.4.3 <code>running_kernels</code>	398
10.19 <code>cl::sycl::detail::shared_ptr_implementation< Parent, Implementation ></code> Struct Template Reference	399
10.19.1 Detailed Description	400
10.19.2 Constructor & Destructor Documentation	400
10.19.2.1 <code>shared_ptr_implementation(std::shared_ptr< Implementation > i)</code>	400
10.19.2.2 <code>shared_ptr_implementation(Implementation *i)</code>	401
10.19.2.3 <code>shared_ptr_implementation()</code> =default	401

10.19.3 Member Function Documentation	401
10.19.3.1 hash() const	401
10.19.3.2 operator<(const Parent &other) const	402
10.19.3.3 operator==(const Parent &other) const	402
10.19.4 Member Data Documentation	403
10.19.4.1 implementation	403
10.20 cl::sycl::detail::singleton< T > Struct Template Reference	403
10.20.1 Detailed Description	403
10.20.2 Member Function Documentation	403
10.20.2.1 instance()	403
10.21 cl::sycl::detail::task Struct Reference	404
10.21.1 Detailed Description	406
10.21.2 Constructor & Destructor Documentation	407
10.21.2.1 task(const std::shared_ptr< detail::queue > &q)	407
10.21.3 Member Function Documentation	407
10.21.3.1 add_buffer(std::shared_ptr< detail::buffer_base > &buf, bool is_write_mode)	407
10.21.3.2 add_postlude(const std::function< void(void)> &f)	407
10.21.3.3 add_prelude(const std::function< void(void)> &f)	408
10.21.3.4 get_kernel()	408
10.21.3.5 get_queue()	408
10.21.3.6 notify_consumers()	409
10.21.3.7 postlude()	409
10.21.3.8 prelude()	410
10.21.3.9 release_buffers()	410
10.21.3.10 schedule(std::function< void(void)> f)	411
10.21.3.11 set_kernel(const std::shared_ptr< cl::sycl::detail::kernel > &k)	412
10.21.3.12 wait()	412
10.21.3.13 wait_for_producers()	413
10.21.4 Member Data Documentation	413
10.21.4.1 buffers_in_use	413
10.21.4.2 epilogues	413
10.21.4.3 execution_ended	413
10.21.4.4 kernel	414
10.21.4.5 owner_queue	414
10.21.4.6 producer_tasks	414
10.21.4.7 prologues	414
10.21.4.8 ready	414
10.21.4.9 ready_mutex	414

11 File Documentation	415
11.1 include/CL/sycl.hpp File Reference	415
11.2 sycl.hpp	416
11.3 include/CL/sycl/access.hpp File Reference	417
11.4 access.hpp	418
11.5 include/CL/sycl/accessor.hpp File Reference	419
11.6 accessor.hpp	420
11.7 include/CL/sycl/buffer/detail/accessor.hpp File Reference	425
11.8 accessor.hpp	426
11.9 include/CL/sycl/address_space/detail/address_space.hpp File Reference	431
11.9.1 Detailed Description	433
11.10 address_space.hpp	433
11.11 include/CL/sycl/address_space.hpp File Reference	438
11.11.1 Detailed Description	440
11.12 address_space.hpp	440
11.13 include/CL/sycl/buffer/detail/buffer.hpp File Reference	442
11.14 buffer.hpp	443
11.15 include/CL/sycl/buffer.hpp File Reference	446
11.16 buffer.hpp	448
11.17 include/CL/sycl/buffer/detail/buffer_base.hpp File Reference	454
11.18 buffer_base.hpp	455
11.19 include/CL/sycl/buffer/detail/buffer_waiter.hpp File Reference	457
11.20 buffer_waiter.hpp	459
11.21 include/CL/sycl/buffer_allocator.hpp File Reference	460
11.22 buffer_allocator.hpp	461
11.23 include/CL/sycl/command_group/detail/task.hpp File Reference	462
11.24 task.hpp	463
11.25 include/CL/sycl/context.hpp File Reference	466
11.26 context.hpp	468
11.27 include/CL/sycl/detail/array_tuple_helpers.hpp File Reference	470

11.27.1 Detailed Description	472
11.28array_tuple_helpers.hpp	472
11.29include/CL/sycl/detail/cache.hpp File Reference	474
11.30cache.hpp	475
11.31include/CL/sycl/detail/debug.hpp File Reference	477
11.31.1 Macro Definition Documentation	478
11.31.1.1 TRISYCL_DUMP	478
11.31.1.2 TRISYCL_DUMP_T	478
11.31.1.3 TRISYCL_INTERNAL_DUMP	478
11.32debug.hpp	479
11.33include/CL/sycl/detail/default_classes.hpp File Reference	481
11.34default_classes.hpp	483
11.35include/CL/sycl/detail/global_config.hpp File Reference	484
11.36global_config.hpp	485
11.37include/CL/sycl/detail/linear_id.hpp File Reference	486
11.38linear_id.hpp	488
11.39include/CL/sycl/detail/shared_ptr_implementation.hpp File Reference	488
11.40shared_ptr_implementation.hpp	490
11.41include/CL/sycl/detail/singleton.hpp File Reference	491
11.42singleton.hpp	492
11.43include/CL/sycl/detail/small_array.hpp File Reference	493
11.44small_array.hpp	495
11.45include/CL/sycl/detail/unimplemented.hpp File Reference	498
11.46unimplemented.hpp	499
11.47include/CL/sycl/device/detail/device.hpp File Reference	500
11.48device.hpp	501
11.49include/CL/sycl/device.hpp File Reference	502
11.50device.hpp	504
11.51include/CL/sycl/info/device.hpp File Reference	507
11.52device.hpp	510

11.53	include/CL/sycl/device/detail/device_tail.hpp File Reference	512
11.54	device_tail.hpp	513
11.55	include/CL/sycl/device/detail/host_device.hpp File Reference	513
11.56	host_device.hpp	515
11.57	include/CL/sycl/device/detail/ocl_device.hpp File Reference	516
11.58	ocl_device.hpp	518
11.59	include/CL/sycl/device_selector.hpp File Reference	519
11.60	device_selector.hpp	521
11.61	include/CL/sycl/device_selector/detail/device_selector_tail.hpp File Reference	522
11.62	device_selector_tail.hpp	523
11.63	include/CL/sycl/error_handler.hpp File Reference	524
11.64	error_handler.hpp	526
11.65	include/CL/sycl/event.hpp File Reference	527
11.66	event.hpp	527
11.67	include/CL/sycl/exception.hpp File Reference	528
11.68	exception.hpp	530
11.69	include/CL/sycl/group.hpp File Reference	533
11.70	group.hpp	535
11.71	include/CL/sycl/handler.hpp File Reference	537
11.71.1	Macro Definition Documentation	539
11.71.1.1	TRISYCL_parallel_for_functor_GLOBAL	539
11.71.1.2	TRISYCL_ParallelForFunctor_GLOBAL_OFFSET	539
11.71.1.3	TRISYCL_ParallelForKernel_RANGE	540
11.72	handler.hpp	540
11.73	include/CL/sycl/handler_event.hpp File Reference	545
11.74	handler_event.hpp	546
11.75	include/CL/sycl/id.hpp File Reference	546
11.76	id.hpp	548
11.77	include/CL/sycl/image.hpp File Reference	549
11.77.1	Detailed Description	549

11.78image.hpp	550
11.79include/CL/sycl/info/param_traits.hpp File Reference	551
11.79.1 Macro Definition Documentation	552
11.79.1.1 TRISYCL_INFO_PARAM_TRAITS	552
11.79.1.2 TRISYCL_INFO_PARAM_TRAITS_ANY_T	552
11.80param_traits.hpp	552
11.81include/CL/sycl/info/platform.hpp File Reference	553
11.82platform.hpp	555
11.83include/CL/sycl/platform/detail/platform.hpp File Reference	556
11.84platform.hpp	557
11.85include/CL/sycl/platform.hpp File Reference	558
11.86platform.hpp	560
11.87include/CL/sycl/item.hpp File Reference	562
11.88item.hpp	564
11.89include/CL/sycl/kernel/detail/kernel.hpp File Reference	565
11.89.1 Macro Definition Documentation	567
11.89.1.1 TRISYCL_ParallelForKernel_RANGE	567
11.90kernel.hpp	567
11.91include/CL/sycl/kernel.hpp File Reference	568
11.92kernel.hpp	569
11.93include/CL/sycl/kernel/detail/opencl_kernel.hpp File Reference	571
11.93.1 Macro Definition Documentation	573
11.93.1.1 TRISYCL_ParallelForKernel_RANGE	573
11.94opencl_kernel.hpp	573
11.95include/CL/sycl/nd_item.hpp File Reference	575
11.96nd_item.hpp	576
11.97include/CL/sycl/nd_range.hpp File Reference	579
11.98nd_range.hpp	580
11.99include/CL/sycl/parallelism/detail/parallelism.hpp File Reference	581
11.99.1 Detailed Description	583

11.10parallelism.hpp	584
11.10include/CL/sycl/parallelism.hpp File Reference	588
11.101.Detailed Description	589
11.10parallelism.hpp	590
11.10include/CL/sycl/pipe/detail/pipe.hpp File Reference	590
11.10pipe.hpp	592
11.10include/CL/sycl/pipe.hpp File Reference	597
11.10pipe.hpp	598
11.10include/CL/sycl/pipe/detail/pipe_accessor.hpp File Reference	600
11.10pipe_accessor.hpp	601
11.10include/CL/sycl/pipe_reservation/detail/pipe_reservation.hpp File Reference	604
11.10pipe_reservation.hpp	606
11.11include/CL/sycl/pipe_reservation.hpp File Reference	608
11.11pipe_reservation.hpp	609
11.11include/CL/sycl/platform/detail/host_platform.hpp File Reference	611
11.11host_platform.hpp	613
11.11include/CL/sycl/platform/detail/opencl_platform.hpp File Reference	614
11.11opencl_platform.hpp	616
11.11include/CL/sycl/queue/detail/host_queue.hpp File Reference	618
11.11host_queue.hpp	619
11.11include/CL/sycl/queue/detail/opencl_queue.hpp File Reference	620
11.12opencl_queue.hpp	621
11.12include/CL/sycl/queue/detail/queue.hpp File Reference	622
11.12queue.hpp	623
11.12include/CL/sycl/queue.hpp File Reference	625
11.12queue.hpp	627
11.12include/CL/sycl/range.hpp File Reference	631
11.12range.hpp	633
11.12include/CL/sycl/static_pipe.hpp File Reference	634
11.12static_pipe.hpp	635
11.12include/CL/sycl/vec.hpp File Reference	636
11.129.Detailed Description	638
11.13vec.hpp	638

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/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.

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 multidimensional 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 inferred 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 multidimensional 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 >::opengl_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 >::opengl_type`

Add to the specification

Member `cl::sycl::detail::address_space_variable< T, AS >::opengl_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::opengl_device::get_platform () const override`

To be implemented

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

To be implemented

Member `cl::sycl::detail::opengl_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::opengl_queue::get_context () const override`

Finish context

Member `cl::sycl::detail::parallel_for` (`nd_range< Dimensions > r, ParallelForFunc` 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, ParallelForFunc` 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 committed 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_type_name_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 operations

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:

Data access and storage in SYCL	27
Dealing with OpenCL address spaces	121
Platforms, contexts, devices and queues	144
Helpers to do array and tuple conversion	217
Debugging and tracing support	221
Manage default configuration and types	224
Some helpers for the implementation	226
Error handling	240
Expressing parallelism through kernels	266
Vector types in SYCL	317

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

cl	The vector type to be used as SYCL vector	323
cl::sycl	323
cl::sycl::access		
	Describe the type of access by kernels	328
cl::sycl::detail	330
cl::sycl::info	334
cl::sycl::trisycl	337
std	337

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

cl::sycl::detail::address_space_base< T, AS >	121
cl::sycl::detail::address_space_object< T, AS >	121
cl::sycl::detail::address_space_variable< T, AS >	121
cl::sycl::detail::address_space_array< T, AS >	121
cl::sycl::detail::address_space_fundamental< T, AS >	121
cl::sycl::detail::address_space_ptr< T, AS >	121
array	
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >	226
cl::sycl::detail::small_array< BasicType, FinalType, 1 >	226
cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >	226
cl::sycl::detail::small_array< BasicType, FinalType, 2 >	226
cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >	226
cl::sycl::detail::small_array< BasicType, FinalType, 3 >	226
cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >	226
cl::sycl::detail::small_array< BasicType, FinalType, Dims >	226
cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >	226
cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements >	226
cl::sycl::vec< DataType, NumElements >	317
cl::sycl::detail::small_array< std::size_t, id< dims >, Dims >	226
cl::sycl::detail::small_array_123< std::size_t, id< dims >, dims >	226
cl::sycl::id< dims >	266
cl::sycl::id< dimensionality >	266
cl::sycl::detail::small_array< std::size_t, range< dims >, Dims >	226
cl::sycl::detail::small_array_123< std::size_t, range< dims >, dims >	226
cl::sycl::range< dims >	266
cl::sycl::range< dimensionality >	266
bitwise	
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >	226
cl::sycl::detail::small_array< BasicType, FinalType, 1 >	226
cl::sycl::detail::small_array< BasicType, FinalType, 2 >	226
cl::sycl::detail::small_array< BasicType, FinalType, 3 >	226
cl::sycl::detail::small_array< BasicType, FinalType, Dims >	226
cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements >	226
cl::sycl::detail::small_array< std::size_t, id< dims >, Dims >	226

cl::sycl::detail::small_array< std::size_t, range< dims >, Dims >	226
cl::sycl::detail::cache< Key, Value >	358
cl::sycl::detail::cache< cl_command_queue, detail::cl::sycl::detail::opencl_queue >	358
cl::sycl::detail::cache< cl_device_id, detail::cl::sycl::detail::opencl_device >	358
cl::sycl::detail::cache< cl_kernel, detail::cl::sycl::detail::opencl_kernel >	358
cl::sycl::detail::cache< cl_platform_id, detail::cl::sycl::detail::opencl_platform >	358
cl::sycl::context	144
cl::sycl::detail::debug< T >	221
cl::sycl::detail::debug< accessor< T, Dimensions, Mode, Target > >	221
cl::sycl::detail::accessor< T, Dimensions, Mode, Target >	27
cl::sycl::detail::debug< buffer< T, Dimensions > >	221
cl::sycl::detail::buffer< T, Dimensions >	27
cl::sycl::detail::debug< buffer< T, Dimensions, Allocator > >	221
cl::sycl::buffer< T, Dimensions, Allocator >	339
cl::sycl::detail::debug< buffer_waiter< T, Dimensions, Allocator > >	221
cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >	27
cl::sycl::detail::debug< detail::kernel >	221
cl::sycl::detail::kernel	144
cl::sycl::detail::opencl_kernel	382
cl::sycl::detail::debug< detail::pipe_accessor< DataType, AccessMode, Target > >	221
cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >	27
cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >	27
cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::pipe >	27
cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >	27
cl::sycl::detail::debug< detail::pipe_accessor< T, AccessMode, Target > >	221
cl::sycl::detail::pipe_accessor< T, AccessMode, Target >	27
cl::sycl::detail::debug< detail::pipe_reservation< PipeAccessor > >	221
cl::sycl::detail::pipe_reservation< PipeAccessor >	27
cl::sycl::detail::debug< detail::queue >	221
cl::sycl::detail::queue	392
cl::sycl::detail::host_queue	373
cl::sycl::detail::opencl_queue	386
cl::sycl::detail::debug< host_queue >	221
cl::sycl::detail::host_queue	373
cl::sycl::detail::debug< opencl_kernel >	221
cl::sycl::detail::opencl_kernel	382
cl::sycl::detail::debug< opencl_queue >	221
cl::sycl::detail::opencl_queue	386
cl::sycl::detail::debug< pipe< T > >	221
cl::sycl::detail::pipe< T >	27
cl::sycl::pipe< T >	27
cl::sycl::detail::debug< pipe< value_type > >	221
cl::sycl::detail::pipe< value_type >	27
cl::sycl::detail::debug< queue >	221
cl::sycl::queue	144
cl::sycl::detail::debug< static_pipe< T, Capacity > >	221
cl::sycl::static_pipe< T, Capacity >	27
cl::sycl::detail::debug< task >	221
cl::sycl::detail::task	404
cl::sycl::detail::device	144
cl::sycl::detail::host_device	369
cl::sycl::detail::opencl_device	376

cl::sycl::device_selector	144
cl::sycl::device_type_selector	144
cl::sycl::device_type_name_selector< DeviceType >	144
cl::sycl::detail::display_vector< T >	221
cl::sycl::detail::display_vector< FinalType >	221
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >	226
cl::sycl::detail::small_array< BasicType, FinalType, 1 >	226
cl::sycl::detail::small_array< BasicType, FinalType, 2 >	226
cl::sycl::detail::small_array< BasicType, FinalType, 3 >	226
cl::sycl::detail::small_array< BasicType, FinalType, Dims >	226
cl::sycl::detail::display_vector< id< dims > >	221
cl::sycl::detail::small_array< std::size_t, id< dims >, Dims >	226
cl::sycl::detail::display_vector< range< dims > >	221
cl::sycl::detail::small_array< std::size_t, range< dims >, Dims >	226
cl::sycl::detail::display_vector< vec< DataType, NumElements > >	221
cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements >	226
enable_shared_from_this	
cl::sycl::detail::buffer_base	353
cl::sycl::detail::buffer< T, Dimensions >	27
cl::sycl::detail::task	404
cl::sycl::error_handler	240
cl::sycl::trisycl::default_error_handler	362
euclidean_ring_operators	
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >	226
cl::sycl::detail::small_array< BasicType, FinalType, 1 >	226
cl::sycl::detail::small_array< BasicType, FinalType, 2 >	226
cl::sycl::detail::small_array< BasicType, FinalType, 3 >	226
cl::sycl::detail::small_array< BasicType, FinalType, Dims >	226
cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements >	226
cl::sycl::detail::small_array< std::size_t, id< dims >, Dims >	226
cl::sycl::detail::small_array< std::size_t, range< dims >, Dims >	226
cl::sycl::event	363
cl::sycl::exception	240
cl::sycl::async_exception	240
cl::sycl::cl_exception	240
cl::sycl::device_error	240
cl::sycl::compile_program_error	240
cl::sycl::feature_not_supported	240
cl::sycl::invalid_object_error	240
cl::sycl::link_program_error	240
cl::sycl::memory_allocation_error	240
cl::sycl::platform_error	240
cl::sycl::profiling_error	240
cl::sycl::runtime_error	240
cl::sycl::accessor_error	240
cl::sycl::event_error	240
cl::sycl::invalid_parameter_error	240
cl::sycl::kernel_error	240
cl::sycl::nd_range_error	240
cl::sycl::non_cl_error	240
cl::sycl::pipe_error	240
cl::sycl::detail::expand_to_vector< V, Tuple, expansion >	217
cl::sycl::detail::expand_to_vector< V, Tuple, true >	217
cl::sycl::group< dims >	266
cl::sycl::handler	144
handler_event	363

std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >	364
std::hash< cl::sycl::device >	365
std::hash< cl::sycl::kernel >	366
std::hash< cl::sycl::platform >	367
std::hash< cl::sycl::queue >	368
cl::sycl::image< dimensions >	27
cl::sycl::item< dims >	266
cl::sycl::nd_item< dims >	266
cl::sycl::nd_range< dims >	266
cl::sycl::detail::opencil_type< T, AS >	121
cl::sycl::detail::opencil_type< T, constant_address_space >	121
cl::sycl::detail::opencil_type< T, generic_address_space >	121
cl::sycl::detail::opencil_type< T, global_address_space >	121
cl::sycl::detail::opencil_type< T, local_address_space >	121
cl::sycl::detail::opencil_type< T, private_address_space >	121
cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id >	266
cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >	266
cl::sycl::detail::parallel_OpenMP_for_iterate< level, Range, ParallelForFunctor, Id >	266
cl::sycl::info::param_traits< T, Param >	391
cl::sycl::pipe_reservation< PipeAccessor >	27
cl::sycl::detail::platform	144
cl::sycl::detail::host_platform	144
cl::sycl::detail::opencil_platform	144
cl::sycl::detail::reserve_id< T >	27
shiftable	
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >	226
cl::sycl::detail::small_array< BasicType, FinalType, 1 >	226
cl::sycl::detail::small_array< BasicType, FinalType, 2 >	226
cl::sycl::detail::small_array< BasicType, FinalType, 3 >	226
cl::sycl::detail::small_array< BasicType, FinalType, Dims >	226
cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements >	226
cl::sycl::detail::small_array< std::size_t, id< dims >, Dims >	226
cl::sycl::detail::small_array< std::size_t, range< dims >, Dims >	226
cl::sycl::detail::singleton< T >	403
cl::sycl::detail::singleton< host_device >	403
cl::sycl::detail::host_device	369
cl::sycl::detail::singleton< host_platform >	403
cl::sycl::detail::host_platform	144
totally_ordered	
cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >	399
cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >	399
cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >	27
cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer<← waiter< T, Dimensions, Allocator > >	399
cl::sycl::buffer< T, Dimensions, Allocator >	339
cl::sycl::detail::shared_ptr_implementation< buffer_waiter< T, Dimensions, Allocator >, detail::← buffer< T, Dimensions > >	399
cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >	27
cl::sycl::detail::shared_ptr_implementation< device, detail::device >	399
cl::sycl::device	144
cl::sycl::detail::shared_ptr_implementation< kernel, detail::kernel >	399
cl::sycl::kernel	144
cl::sycl::detail::shared_ptr_implementation< pipe< T >, detail::pipe< T > >	399
cl::sycl::pipe< T >	27
cl::sycl::detail::shared_ptr_implementation< platform, detail::platform >	399
cl::sycl::platform	144

cl::sycl::detail::shared_ptr_implementation< queue, detail::queue >	399
cl::sycl::queue	144
cl::sycl::detail::shared_ptr_implementation< static_pipe< T, Capacity >, detail::pipe< T > >	399
cl::sycl::static_pipe< T, Capacity >	27
type	
cl::sycl::detail::address_space_object< T, AS >	121
vector	
cl::sycl::exception_list	240
bool	??
shared_ptr< detail::accessor< DataType, Dimensions, AccessMode, Target > >	??
shared_ptr< detail::buffer< T, Dimensions > >	??
shared_ptr< detail::buffer_waiter< T, Dimensions, Allocator > >	??
shared_ptr< detail::cl::sycl::detail::pipe< DataType > >	??
shared_ptr< detail::device >	??
shared_ptr< detail::kernel >	??
shared_ptr< detail::pipe< T > >	??
shared_ptr< detail::platform >	??
shared_ptr< detail::queue >	??
static const size_t	??
static constexpr bool	??

Chapter 6

Class Index

6.1 Class List

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

cl::sycl::buffer< T, Dimensions, Allocator >	
<T, Dimensions, Mode, Target> up data Data access and storage in SYCL	339
cl::sycl::detail::buffer_base	
Factorize some template independent buffer aspects in a base class	353
cl::sycl::detail::cache< Key, Value >	
A simple thread safe cache mechanism to cache std::shared_ptr of values indexed by keys . .	358
cl::sycl::trisycl::default_error_handler	362
cl::sycl::event	363
handler_event	
Handler event	363
std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >	364
std::hash< cl::sycl::device >	365
std::hash< cl::sycl::kernel >	366
std::hash< cl::sycl::platform >	367
std::hash< cl::sycl::queue >	368
cl::sycl::detail::host_device	
SYCL host device	369
cl::sycl::detail::host_queue	
Some implementation details about the SYCL queue	373
cl::sycl::detail::opencl_device	
SYCL OpenCL device	376
cl::sycl::detail::opencl_kernel	
An abstraction of the OpenCL kernel	382
cl::sycl::detail::opencl_queue	
Some implementation details about the SYCL queue	386
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)	391
cl::sycl::detail::queue	
Some implementation details about the SYCL queue	392
cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >	
Provide an implementation as shared_ptr with total ordering and hashing to be used with algo- rithms and in (un)ordered containers	399
cl::sycl::detail::singleton< T >	
Provide a singleton factory	403
cl::sycl::detail::task	
The abstraction to represent SYCL tasks executing inside command_group	404

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

include/CL/sycl.hpp	415
include/CL/sycl/access.hpp	417
include/CL/sycl/accessor.hpp	419
include/CL/sycl/address_space.hpp	
Implement OpenCL address spaces in SYCL with C++-style	438
include/CL/sycl/buffer.hpp	446
include/CL/sycl/buffer_allocator.hpp	460
include/CL/sycl/context.hpp	466
include/CL/sycl/device.hpp	502
include/CL/sycl/device_selector.hpp	519
include/CL/sycl/error_handler.hpp	524
include/CL/sycl/event.hpp	527
include/CL/sycl/exception.hpp	528
include/CL/sycl/group.hpp	533
include/CL/sycl/handler.hpp	537
include/CL/sycl/handler_event.hpp	545
include/CL/sycl/id.hpp	546
include/CL/sycl/image.hpp	
OpenCL SYCL image class	549
include/CL/sycl/item.hpp	562
include/CL/sycl/kernel.hpp	568
include/CL/sycl/nd_item.hpp	575
include/CL/sycl/nd_range.hpp	579
include/CL/sycl/parallelism.hpp	
Implement parallel constructions to launch kernels	588
include/CL/sycl/pipe.hpp	597
include/CL/sycl/pipe_reservation.hpp	608
include/CL/sycl/platform.hpp	558
include/CL/sycl/queue.hpp	625
include/CL/sycl/range.hpp	631
include/CL/sycl/static_pipe.hpp	634
include/CL/sycl/vec.hpp	
Implement the small OpenCL vector class	636
include/CL/sycl/address_space/detail/address_space.hpp	
Implement OpenCL address spaces in SYCL with C++-style	431

include/CL/sycl/buffer/detail/ accessor.hpp	425
include/CL/sycl/buffer/detail/ buffer.hpp	442
include/CL/sycl/buffer/detail/ buffer_base.hpp	454
include/CL/sycl/buffer/detail/ buffer_waiter.hpp	457
include/CL/sycl/command_group/detail/ task.hpp	462
include/CL/sycl/detail/ array_tuple_helpers.hpp	
Some helpers to do array-tuple conversions	470
include/CL/sycl/detail/ cache.hpp	474
include/CL/sycl/detail/ debug.hpp	477
include/CL/sycl/detail/ default_classes.hpp	481
include/CL/sycl/detail/ global_config.hpp	484
include/CL/sycl/detail/ linear_id.hpp	486
include/CL/sycl/detail/ shared_ptr_implementation.hpp	488
include/CL/sycl/detail/ singleton.hpp	491
include/CL/sycl/detail/ small_array.hpp	493
include/CL/sycl/detail/ unimplemented.hpp	498
include/CL/sycl/device/detail/ device.hpp	500
include/CL/sycl/device/detail/ device_tail.hpp	512
include/CL/sycl/device/detail/ host_device.hpp	513
include/CL/sycl/device/detail/ opencl_device.hpp	516
include/CL/sycl/device_selector/detail/ device_selector_tail.hpp	522
include/CL/sycl/info/ device.hpp	507
include/CL/sycl/info/ param_traits.hpp	551
include/CL/sycl/info/ platform.hpp	553
include/CL/sycl/kernel/detail/ kernel.hpp	565
include/CL/sycl/kernel/detail/ opencl_kernel.hpp	571
include/CL/sycl/parallelism/detail/ parallelism.hpp	
Implement the detail of the parallel constructions to launch kernels	581
include/CL/sycl/pipe/detail/ pipe.hpp	590
include/CL/sycl/pipe/detail/ pipe_accessor.hpp	600
include/CL/sycl/pipe_reservation/detail/ pipe_reservation.hpp	604
include/CL/sycl/platform/detail/ host_platform.hpp	611
include/CL/sycl/platform/detail/ opencl_platform.hpp	614
include/CL/sycl/platform/detail/ platform.hpp	556
include/CL/sycl/queue/detail/ host_queue.hpp	618
include/CL/sycl/queue/detail/ opencl_queue.hpp	620
include/CL/sycl/queue/detail/ queue.hpp	622

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>`

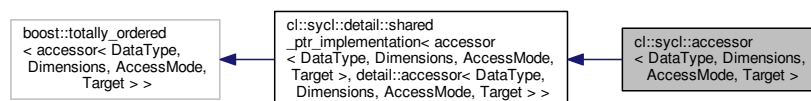
`class cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >`

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

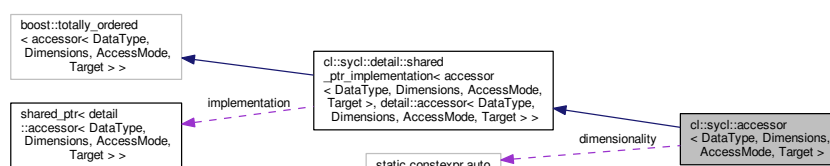
Todo Implement it for images according so section 3.3.4.5

Definition at line 45 of file [accessor.hpp](#).

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



Collaboration diagram for `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, AccessMode, 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`.

```
00099         : implementation_t {
00100     new detail::accessor<DataType, Dimensions, AccessMode, Target> {
00101         target_buffer.implementation->implementation, command_group_handler }
00102     } {
00103     static_assert(Target == access::target::global_buffer
00104         || Target == access::target::constant_buffer,
00105         "access target should be global_buffer or constant_buffer "
00106         "when a handler is used");
00107     }
```

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`.

```
00121     : implementation_t {
00122     new detail::accessor<DataType, Dimensions, AccessMode, Target> {
00123         target_buffer.implementation->implementation }
00124     } {
00125     static_assert(Target == access::target::host_buffer,
00126         "without a handler, access target should be host_buffer");
00127     }
```

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).

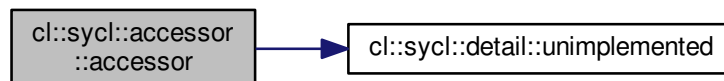
This accessor is recommended for discard-write and discard read write access modes, when the unaffected parts of the processing should be retained.

Definition at line 148 of file [accessor.hpp](#).

References [cl::sycl::detail::unimplemented\(\)](#).

```
00151                                     {
00152     detail::unimplemented();
00153 }
```

Here is the call graph for this function:



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.

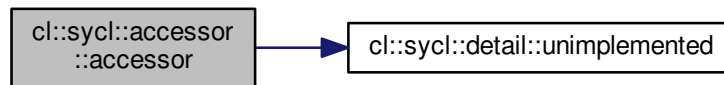
It needs as a parameter a command group handler object from the command group scope. Constructor only available if AccessMode is local, see Table 3.25.

Definition at line 164 of file [accessor.hpp](#).

References [cl::sycl::detail::unimplemented\(\)](#).

```
00165                                     {
00166     detail::unimplemented();
00167 }
```

Here is the call graph for this function:



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 multidimensional 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`.

```

00279                                     {
00280     return implementation->begin();
00281 }
  
```

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`.

```

00296                                     {
00297     return implementation->cbegin();
00298 }
  
```

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](#).

```
00301                                     {
00302     return implementation->cend();
00303 }
```

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](#).

```
00322                                     {
00323     return implementation->rbegin();
00324 }
```

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](#).

```
00327                                     {
00328     return implementation->rend();
00329 }
```

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](#).

```
00285                                     {
00286     return implementation->end();
00287 }
```



```
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](#).

```
00237                                     {
00238     return **implementation;
00239 }
```

```
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](#).

```
00252                                     {
00253     return **implementation;
00254 }
```

```
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](#).

```
00175                                     {
00176     return (*implementation)[index];
00177 }
```

```
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](#).

```
00185                                     {
00186     return (*implementation) [index];
00187 }
```

```
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](#).

```
00191                                     {
00192     return (*implementation) [index];
00193 }
```

```
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](#).

```
00197                                     {
00198     return (*implementation) [index];
00199 }
```

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]`

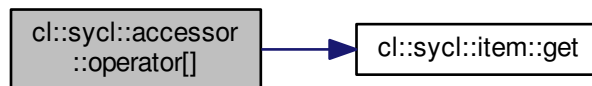
To use an accessor with [item<>].

Definition at line 203 of file [accessor.hpp](#).

References [cl::sycl::item< dims >::get\(\)](#).

```
00203                                     {
00204     return (*this)[index.get()];
00205 }
```

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]`

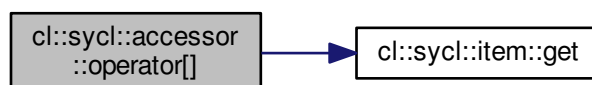
To use an accessor with [item<>].

Definition at line 209 of file [accessor.hpp](#).

References [cl::sycl::item< dims >::get\(\)](#).

```
00209                                     {
00210     return (*this)[index.get()];
00211 }
```

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<>]`.

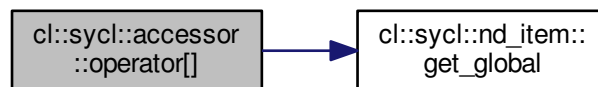
Todo Add in the specification because used by HPC-GPU slide 22

Definition at line 218 of file [accessor.hpp](#).

References [cl::sycl::nd_item< dims >::get_global\(\)](#).

```
00218                                     {
00219     return (*this)[index.get_global()];
00220 }
```

Here is the call graph for this function:



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<>]`.

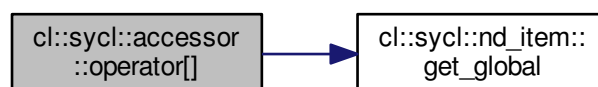
Todo Add in the specification because used by HPC-GPU slide 22

Definition at line 226 of file [accessor.hpp](#).

References [cl::sycl::nd_item< dims >::get_global\(\)](#).

```
00226                                     {
00227     return (*this)[index.get_global()];
00228 }
```

Here is the call graph for this function:



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](#).

```
00306                                     {
00307     return implementation->rbegin();
00308 };
```

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](#).

```
00311                                     {
00312     return implementation->rend();
00313 };
```

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 >`

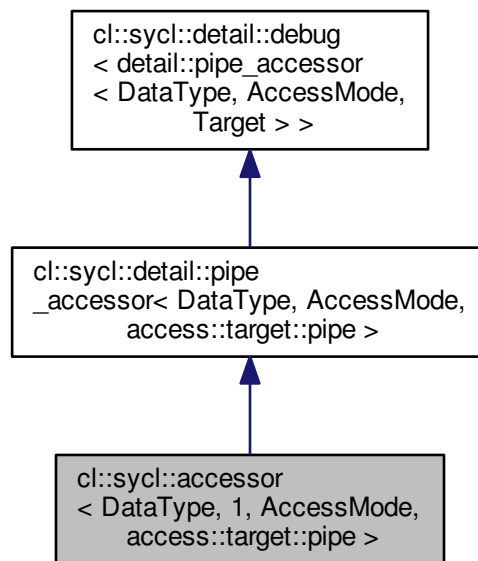
```
template<typename DataType, access::mode AccessMode>
class cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >
```

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

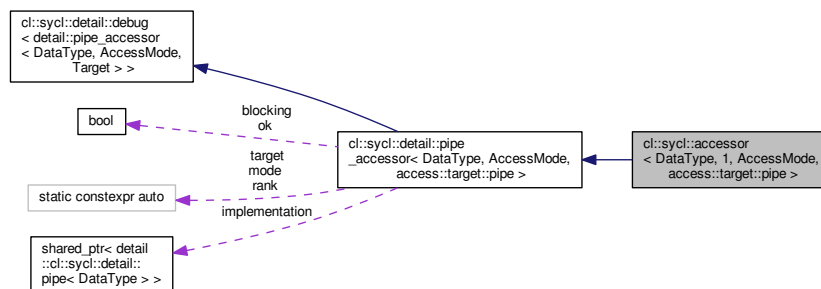
A specialization for an non-blocking pipe

Definition at line 341 of file [accessor.hpp](#).

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



Collaboration diagram for `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]`

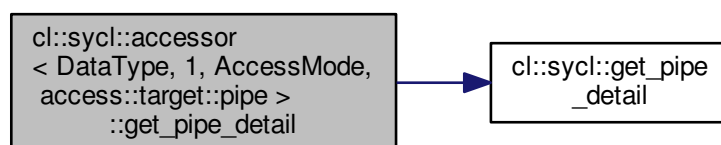
Get the underlying pipe implementation.

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:



```
8.1.2.3.2 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 >
```

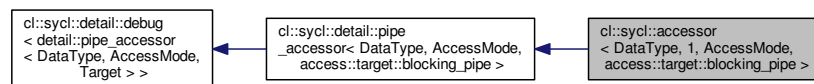
```
template<typename DataType, access::mode AccessMode>
class cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >
```

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

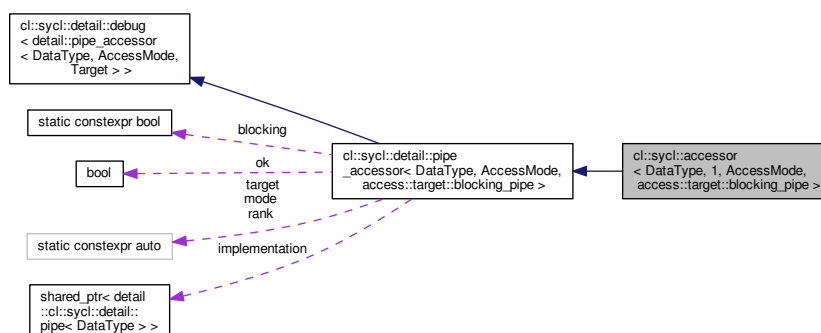
A specialization for a blocking pipe

Definition at line 379 of file [accessor.hpp](#).

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



Collaboration diagram for `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](#).

```
00394      : 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]`

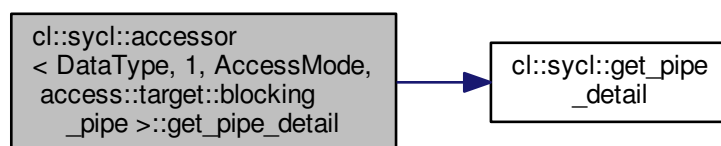
Get the underlying pipe implementation.

Definition at line 404 of file [accessor.hpp](#).

References [cl::sycl::get_pipe_detail\(\)](#).

```
00404      {
00405      return accessor_detail::get_pipe_detail();
00406      }
```

Here is the call graph for this function:



```
8.1.2.3.2 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.

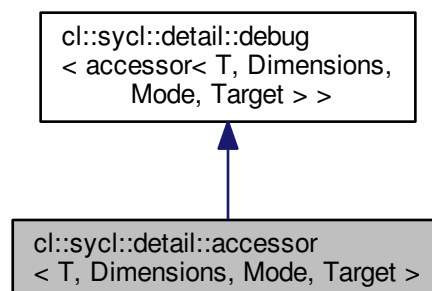
This implementation relies on `boost::multi_array` to provide this nice syntax and behaviour.

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).

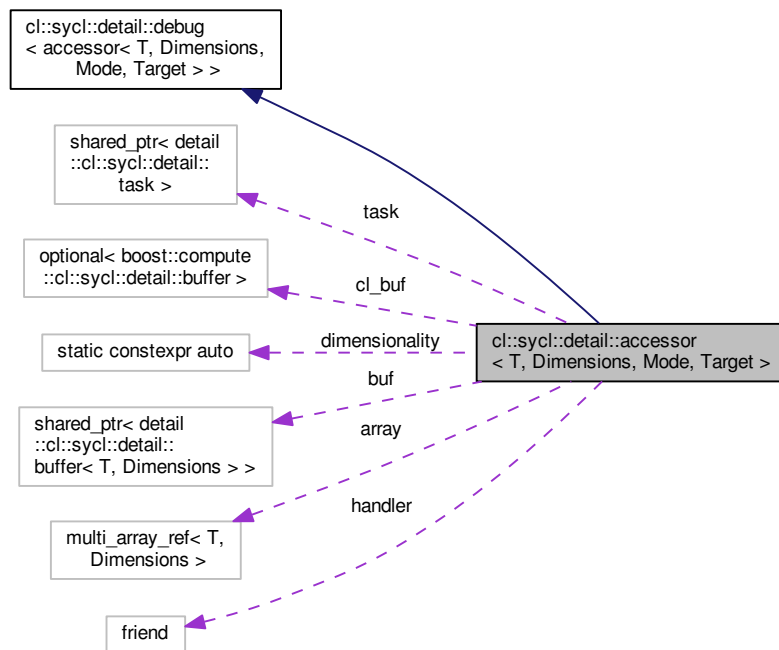
Todo Use the [access::mode](#)

Definition at line 58 of file [accessor.hpp](#).

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



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



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

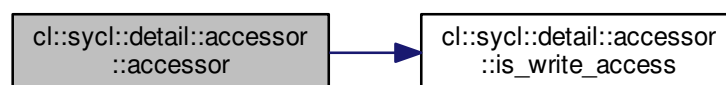
Definition at line 125 of file [accessor.hpp](#).

References [cl::sycl::access::host_buffer](#), [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write_access\(\)](#), and [TRISYCL_DUMP_T](#).

```

00125                                     :
00126     buf { target_buffer }, array { target_buffer->access } {
00127     TRISYCL_DUMP_T("Create a host accessor write = " <<
is_write_access());
00128     static_assert(Target == access::target::host_buffer,
00129         "without a handler, access target should be host_buffer");
00130     /* The host needs to wait for all the producers of the buffer to
00131         have finished */
00132     buf->wait();
00133 }
```

Here is the call graph for this function:



```
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.

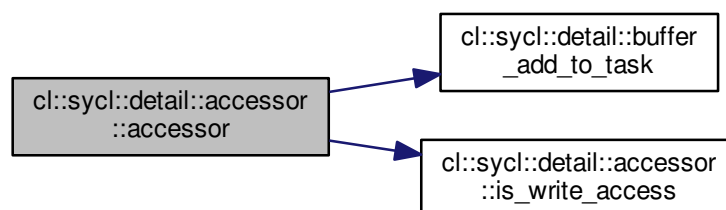
Todo fix the specification to rename target that shadows template parm

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](#).

```
00142                                     :
00143     buf { target_buffer }, array { target_buffer->access } {
00144     TRISYCL_DUMP_T("Create a kernel accessor write = " <<
00145     is_write_access());
00146     static_assert(Target == access::target::global_buffer
00147     || Target == access::target::constant_buffer,
00147     "access target should be global_buffer or constant_buffer "
00148     "when a handler is used");
00149     // Register the buffer to the task dependencies
00150     task = buffer_add_to_task(buf, &command_group_handler,
00151     is_write_access());
00151 }
```

Here is the call graph for this function:



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 multidimensional 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`.

```
00312         {
00313         return const_cast<writable_array_view_type &>(array) .
begin();
00314     }
```

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`.

```
00329 { 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`.

```
00332 { 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`.

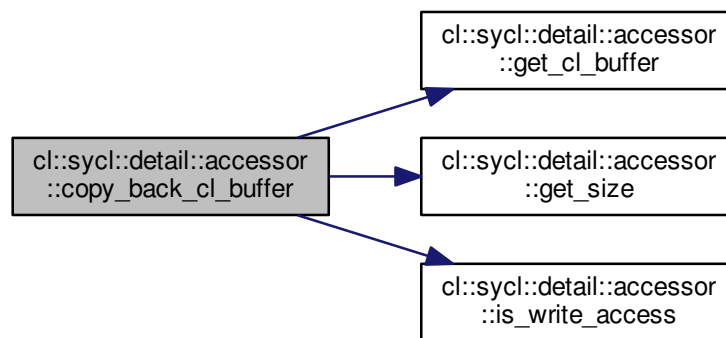
References `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_cl_buffer()`, `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_size()`, and `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write_access()`.


```

00396         {
00397         // \todo Use if constexpr in C++17
00398         if (is_write_access())
00399             task->get_queue()->get_boost_compute()
00400                 .enqueue_read_buffer(get_cl_buffer(),
00401                                     0 /*< Offset */,
00402                                     get_size()*sizeof(value_type),
00403                                     array.data());
00404         }

```

Here is the call graph for this function:



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](#).

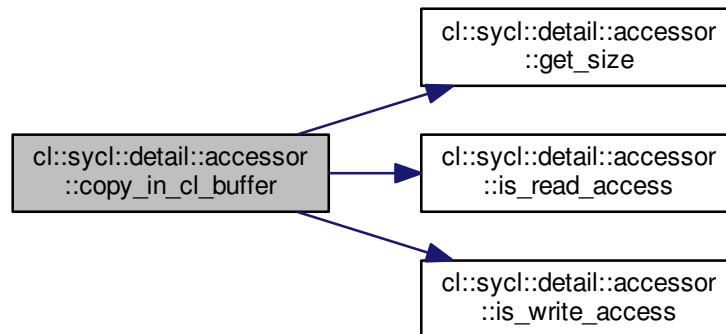
References `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_size()`, `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_read_access()`, and `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write_access()`.

```

00376         {
00377         // This should be a constexpr
00378         cl_mem_flags flags = is_read_access() && is_write_access() ?
00379             CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR
00380             : is_read_access() ? CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR
00381             : CL_MEM_WRITE_ONLY;
00382
00383         /* Create the OpenCL buffer and copy in data from the host if in
00384            read mode */
00385         cl_buf = { task->get_queue()->get_boost_compute().get_context(),
00386                   get_size()*sizeof(value_type),
00387                   flags,
00388                   is_read_access() ? array.data() : 0 };
00389     }

```

Here is the call graph for this function:



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 >::rbegin () const`
`[inline]`

Definition at line 353 of file [accessor.hpp](#).

```
00353 { 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 >::rend () const`
`[inline]`

Definition at line 356 of file [accessor.hpp](#).

```
00356 { 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](#).

```
00318 {
00319     return const_cast<writable_array_view_type &>(array) .
    end();
00320 }
```

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](#).

```
00251                                     {
00252     return *buf;
00253 }
```

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]`

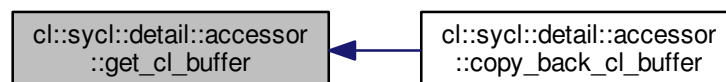
Get the boost::compute::buffer or throw if unset.

Definition at line 365 of file [accessor.hpp](#).

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

```
00365                                     {
00366     // This throws if not set
00367     return cl_buf.value();
00368 }
```

Here is the caller graph for this function:



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](#).

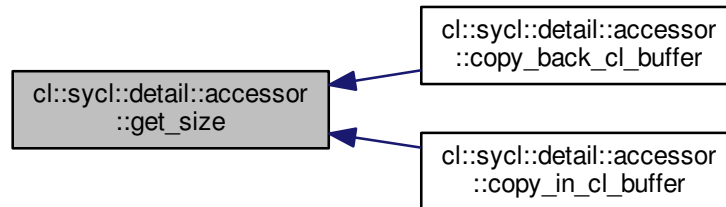
Referenced by [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_back_cl_buffer\(\)](#), and [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_in_cl_buffer\(\)](#).

```

00158         {
00159     return array.num_elements();
00160     }

```

Here is the caller graph for this function:



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`.

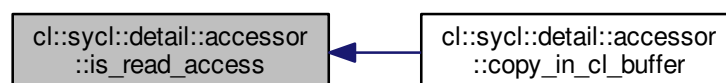
Referenced by `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_in_cl_buffer()`.

```

00264     {
00265     return Mode == access::mode::read
00266         || Mode == access::mode::read_write
00267         || Mode == access::mode::discard_read_write;
00268     }

```

Here is the caller graph for this function:



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 ?

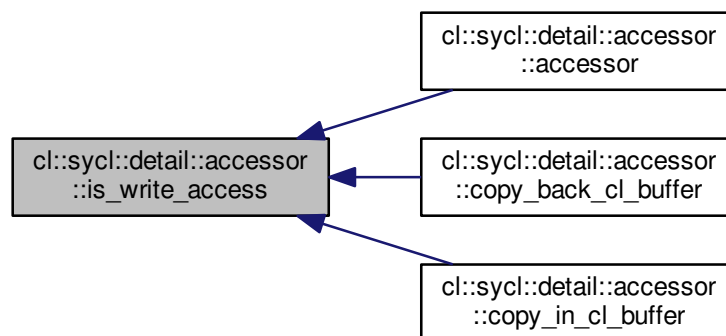
Definition at line 279 of file `accessor.hpp`.

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()`.

```
00279 {
00280     return Mode == access::mode::write
00281     || Mode == access::mode::read_write
00282     || Mode == access::mode::discard_write
00283     || Mode == access::mode::discard_read_write;
00284 }
```

Here is the caller graph for this function:



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`.

```
00230 {
00231     return *array.data();
00232 }
```

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](#).

```
00245                                     {
00246     return *array.data();
00247 }
```

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](#).

```
00168                                     {
00169     return array[index];
00170 }
```

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](#).

```
00178                                     {
00179     return array[index];
00180 }
```

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](#).

```
00184                                     {
00185     return array(index);
00186 }
```

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](#).

```
00190                                     {
00191     return array(index);
00192 }
```

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]`

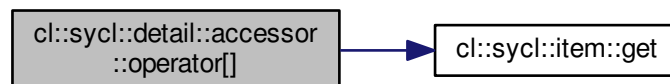
To use an accessor with `[item<>]`.

Definition at line 196 of file [accessor.hpp](#).

References [cl::sycl::item< dims >::get\(\)](#).

```
00196                                     {
00197     return (*this)[index.get()];
00198 }
```

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\(\)](#).

```
00202                                     {
00203     return (*this)[index.get()];
00204 }
```

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

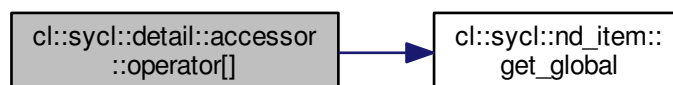
Definition at line 211 of file [accessor.hpp](#).

References [cl::sycl::nd_item< dims >::get_global\(\)](#).

```

00211                                     {
00212     return (*this)[index.get_global()];
00213 }
  
```

Here is the call graph for this function:



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<>]`.

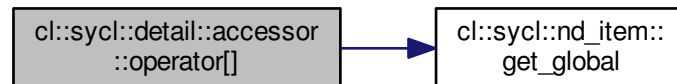
Todo Add in the specification because used by HPC-GPU slide 22

Definition at line 219 of file [accessor.hpp](#).

References [cl::sycl::nd_item< dims >::get_global\(\)](#).

```
00219                                     {
00220     return (*this)[index.get_global()];
00221 }
```

Here is the call graph for this function:



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](#).

```
00336                                     {
00337     return const_cast<writable_array_view_type &>(array) .
00338     rbegin();
00338 }
```

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](#).

```
00342                                     {
00343     return const_cast<writable_array_view_type &>(array) .
00344     rend();
00344 }
```

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`

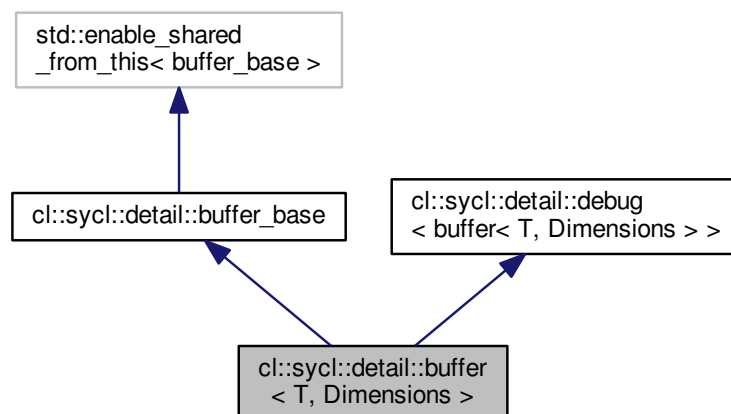
```
template<typename T, std::size_t Dimensions = 1>  
class cl::sycl::detail::buffer< T, Dimensions >
```

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 >`:



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

<i>r</i>	
----------	--

Definition at line 82 of file [buffer.hpp](#).

```
00082         : buffer_base { false },
00083           allocation { r },
00084           access { allocation }
00085         {}
```

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

<i>host_data</i>	of size
<i>r</i>	without further allocation

Definition at line 90 of file [buffer.hpp](#).

```
00090         : buffer_base { false },
00091           access { host_data, r },
00092           host_write_back { true }
00093         {}
```

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

<i>host_data</i>	of size
<i>r</i>	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](#).

```
00102         :
00103         /* \todo Need to solve this const buffer issue in a clean way
00104
00105         Just allocate memory? */
00106         buffer_base { true },
00107         access { const_cast<T *>(host_data), r }
00108         {}
```

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

<i>M</i>	
----------	--

Todo Remove if not used

Definition at line 177 of file [buffer.hpp](#).

```
00177                                     {
00178     return { *this };
00179 }
```

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\(\)](#).

```
00204                                     {
00205     return allocation.num_elements();
00206 }
```

Here is the caller graph for this function:



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`.

```

00240                                     {
00241     boost::optional<std::future<void>> f;
00242     /* If there is only 1 shared_ptr user of the buffer, this is the
00243        caller of this function, the \c buffer_waiter, so there is no
00244        need to get a \ future otherwise there will be a dead-lock if
00245        there is only 1 thread waiting for itself.
00246
00247        Since \c use_count() is applied to a \c shared_ptr just created
00248        for this purpose, it actually increase locally the count by 1,
00249        so check for 1 + 1 use count instead...
00250     */
00251     if (shared_from_this().use_count() > 2)
00252         // \todo Double check the specification and add unit tests
00253         if (host_write_back || !final_data.expired() ||
shared_data) {
00254         // Create a promise to wait for
00255         notify_buffer_destructor = std::promise<void> {};
00256         // And return the future to wait for it
00257         f = notify_buffer_destructor->get_future();
00258     }
00259     return f;
00260 }
```

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`.

```

00186                                     {
00187     /* Interpret the shape which is a pointer to the first element as an
00188        array of Dimensions elements so that the range<Dimensions>
00189        constructor is happy with this collection
00190
00191        \todo Add also a constructor in range<> to accept a const
00192        std::size_t *?
00193     */
00194     return range<Dimensions> {
00195         *(const std::size_t (*) [Dimensions]) (allocation.shape())
00196     };
00197 }
```

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)`.

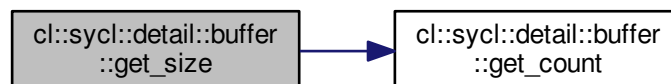
Todo rename to something else. In <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf> it is named `bytes()` for example

Definition at line 217 of file [buffer.hpp](#).

References [cl::sycl::detail::buffer< T, Dimensions >::get_count\(\)](#).

```
00217         {
00218     return get_count()*sizeof(T);
00219     }
```

Here is the call graph for this function:



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](#).

```
00228                                     {
00229     final_data = finalData;
00230 }
```

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](#).

```
8.1.2.5.2  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](#).

```
8.1.2.5.3  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](#).

```
8.1.2.5.4  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  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\(\)](#).

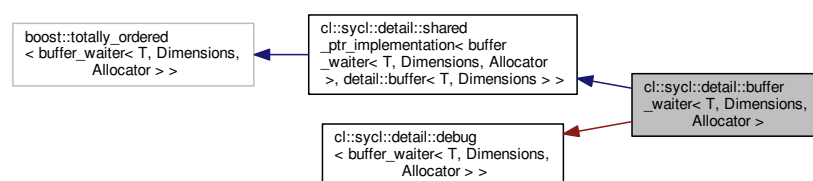
8.1.2.6 class `cl::sycl::detail::buffer_waiter`

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

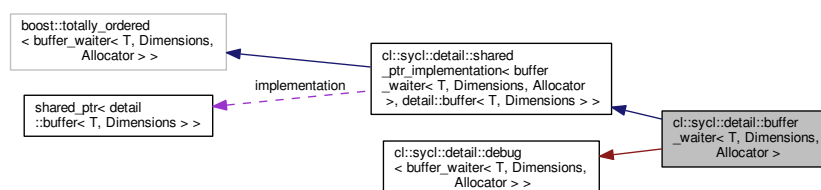
A helper class to wait for the final buffer destruction if the conditions for blocking are met.

Definition at line 36 of file [buffer_waiter.hpp](#).

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



Collaboration diagram for `cl::sycl::detail::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](#).

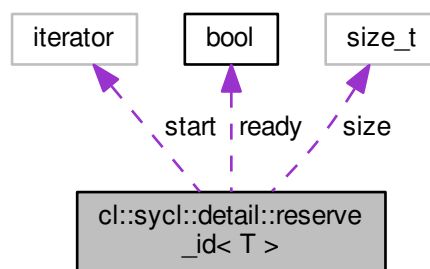
8.1.2.8 struct cl::sycl::detail::reserve_id

```
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	<i>start</i>	point to the start of the reservation in the pipe storage
in	<i>size</i>	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`

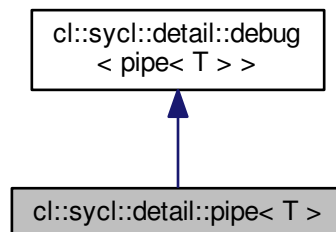
```
template<typename T>
class cl::sycl::detail::pipe<T>
```

Implement a pipe object.

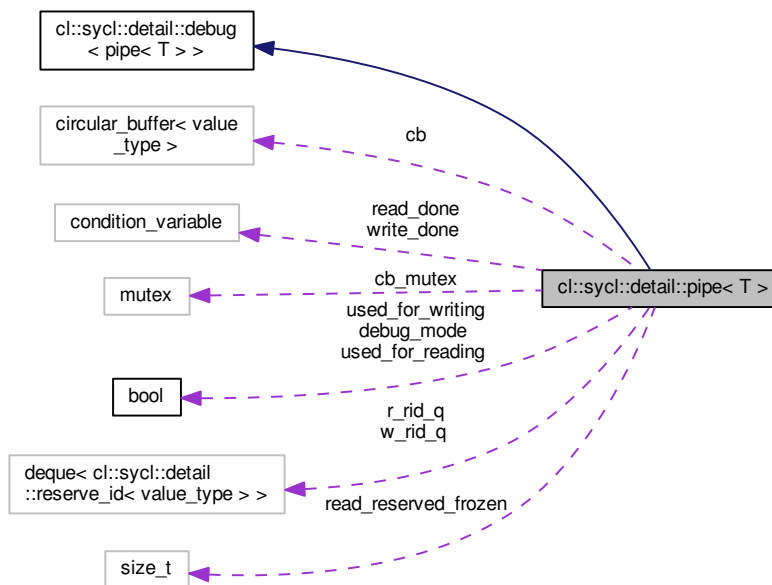
Use some mutable members so that the pipe object can be changed even when the accessors are captured in a lambda.

Definition at line 70 of file [pipe.hpp](#).

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



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



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](#).

```
00123 : 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](#).

```
00128                                     {
00129     // No lock required since it is fixed and set at construction time
00130     return cb.capacity();
00131 }
```

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](#).

```
00163                                     {
00164     TRISYCL_DUMP_T("empty() cb.size() = " << cb.size()
00165     << " size() = " << size());
00166     // It is empty when the size is zero, taking into account reservations
00167     return size() == 0;
00168 }
```

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](#).

```
00194         {
00195     std::lock_guard<std::mutex> lg { cb_mutex };
00196     return empty();
00197 }
```

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](#).

```
00179         {
00180     return cb.full();
00181 }
```

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](#).

```
00201         {
00202     std::lock_guard<std::mutex> lg { cb_mutex };
00203     return full();
00204 }
```

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

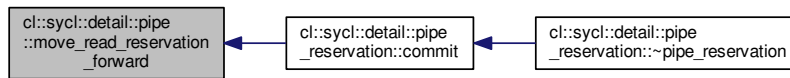
Process the read reservations that are ready to be released in the reservation queue.

Definition at line 422 of file [pipe.hpp](#).

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

```
00422         {
00423     // Lock the pipe to avoid nuisance
00424     std::lock_guard<std::mutex> lg { cb_mutex };
00425
00426     for (;;) {
00427         if (r_rid_q.empty())
00428             // No pending reservation, so nothing to do
00429             break;
00430         if (!r_rid_q.front().ready)
00431             /* If the first reservation is not ready to be released, stop
00432              because it is blocking all the following in the queue
00433              anyway */
00434             break;
00435         // Remove the reservation to be released from the queue
00436         r_rid_q.pop_front();
00437         std::size_t n_to_pop;
00438         if (r_rid_q.empty())
00439             // If it was the last one, remove all the reservation
00440             n_to_pop = read_reserved_frozen;
00441         else
00442             // Else remove everything up to the next reservation
00443             n_to_pop = r_rid_q.front().start - cb.begin();
00444         // No longer take into account these reserved slots
00445         read_reserved_frozen -= n_to_pop;
00446         // Release the elements from the FIFO
00447         while (n_to_pop--)
00448             cb.pop_front();
00449         // Notify the clients waiting for some room to write in the pipe
00450         read_done.notify_all();
00451         /* ...and process the next reservation to see if it is ready to
00452          be released too */
00453     }
00454 }
```

Here is the caller graph for this function:



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\(\)](#).

```

00460         {
00461         // Lock the pipe to avoid nuisance
00462         std::lock_guard<std::mutex> lg { cb_mutex };
00463
00464         for (;;) {
00465             if (w_rid_q.empty())
00466                 // No pending reservation, so nothing to do
00467                 break;
00468             // Get the first reservation
00469             const auto &rid = w_rid_q.front();
00470             if (!rid.ready)
00471                 /* If the reservation is not ready to be released, stop
00472                  because it is blocking all the following in the queue
00473                  anyway */
00474                 break;
00475             // Remove the reservation to be released from the queue
00476             w_rid_q.pop_front();
00477             // Notify the clients waiting to read something from the pipe
00478             write_done.notify_all();
00479             /* ...and process the next reservation to see if it is ready to
00480              be released too */
00481         }
00482     }
  
```

Here is the caller graph for this function:



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	<i>value</i>	is the reference to where to store what is read
in	<i>blocking</i>	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](#).

```

00255                                     {
00256     // Lock the pipe to avoid being disturbed
00257     std::unique_lock<std::mutex> ul { cb_mutex };
00258     TRISYCL_DUMP_T("Read pipe empty = " << empty());
00259
00260     if (blocking)
00261         /* If in blocking mode, wait for the not empty condition, that
00262            may be changed when a write is done */
00263         write_done.wait(ul, [&] { return !empty(); });
00264     else if (empty())
00265         return false;
00266
00267     TRISYCL_DUMP_T("Read pipe front = " << cb.front()
00268                   << " back = " << cb.back()
00269                   << " reserved_for_reading() = " << reserved_for_reading());
00270     if (read_reserved_frozen)
00271         /** If there is a pending reservation, read the next element to
00272            be read and update the number of reserved elements */
00273         value = cb.begin()[read_reserved_frozen++];
00274     else {
00275         /* There is no pending read reservation, so pop the read value
00276            from the pipe */
00277         value = cb.front();
00278         cb.pop_front();
00279     }
00280
00281     TRISYCL_DUMP_T("Read pipe value = " << value);
00282     // Notify the clients waiting for some room to write in the pipe
00283     read_done.notify_all();
00284     return true;
00285 }
```

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	<i>s</i>	is the number of element to reserve
out	<i>rid</i>	is an iterator to a description of the reservation that has been done if successful
in	<i>blocking</i>	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\(\)](#).

```

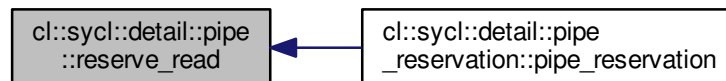
00334                                     {
00335     // Lock the pipe to avoid being disturbed
00336     std::unique_lock<std::mutex> ul { cb_mutex };
00337
00338     TRISYCL_DUMP_T("Before read reservation cb.size() = " << cb.size()
00339                   << " size() = " << size());
```

```

00340     if (s == 0)
00341         // Empty reservation requested, so nothing to do
00342         return false;
00343
00344     if (blocking)
00345         /* If in blocking mode, wait for enough elements to read in the
00346            pipe for the reservation. This condition can change when a
00347            write is done */
00348         write_done.wait(ul, [&] { return s <= size(); });
00349     else if (s > size())
00350         // Not enough elements to read in the pipe for the reservation
00351         return false;
00352
00353     // Compute the location of the first element of the reservation
00354     auto first = cb.begin() + read_reserved_frozen;
00355     // Increment the number of frozen elements
00356     read_reserved_frozen += s;
00357     /* Add a description of the reservation at the end of the
00358        reservation queue */
00359     r_rid_q.emplace_back(first, s);
00360     // Return the iterator to the last reservation descriptor
00361     rid = r_rid_q.end() - 1;
00362     TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
00363                   << " size() = " << size());
00364     return true;
00365 }

```

Here is the caller graph for this function:



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	<i>s</i>	is the number of element to reserve
out	<i>rid</i>	is an iterator to a description of the reservation that has been done if successful
in	<i>blocking</i>	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\(\)](#).

```

00382     {
00383         // Lock the pipe to avoid being disturbed
00384         std::unique_lock<std::mutex> ul { cb_mutex };

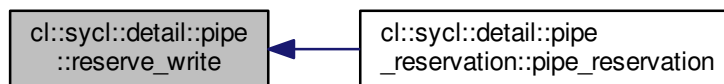
```

```

00385
00386     TRISYCL_DUMP_T("Before write reservation cb.size() = " << cb.size()
00387                   << " size() = " << size());
00388     if (s == 0)
00389         // Empty reservation requested, so nothing to do
00390         return false;
00391
00392     if (blocking)
00393         /* If in blocking mode, wait for enough room in the pipe, that
00394            may be changed when a read is done. Do not use a difference
00395            here because it is only about unsigned values */
00396         read_done.wait(ul, [&] { return cb.size() + s <= capacity(); });
00397     else if (cb.size() + s > capacity())
00398         // Not enough room in the pipe for the reservation
00399         return false;
00400
00401     /* If there is enough room in the pipe, just create default values
00402        in it to do the reservation */
00403     for (std::size_t i = 0; i != s; ++i)
00404         cb.push_back();
00405     /* Compute the location of the first element a posteriori since it
00406        may not exist a priori if cb was empty before */
00407     auto first = cb.end() - s;
00408     /* Add a description of the reservation at the end of the
00409        reservation queue */
00410     w_rid_q.emplace_back(first, s);
00411     // Return the iterator to the last reservation descriptor
00412     rid = w_rid_q.end() - 1;
00413     TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
00414                   << " size() = " << size());
00415     return true;
00416 }

```

Here is the caller graph for this function:



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](#).

```

00296     {
00297         return read_reserved_frozen;
00298     }

```

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](#).

```
00309                                     {
00310     if (w_rid_q.empty())
00311         // No on-going reservation
00312         return 0;
00313     else
00314         /* The reserved size is from the first element of the first
00315            on-going reservation up to the end of the pipe content */
00316         return cb.end() - w_rid_q.front().start;
00317 }
```

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.

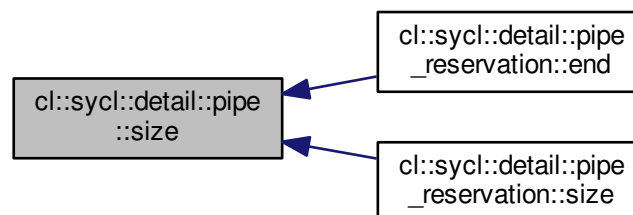
Note that on some devices it may be costly to implement (for example on FPGA).

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\(\)](#).

```
00143     {
00144         TRISYCL_DUMP_T("size() cb.size() = " << cb.size()
00145             << " cb.end() = " << (void *)&cb.end()
00146             << " reserved_for_reading() = " << reserved_for_reading()
00147             << " reserved_for_writing() = " << reserved_for_writing());
00148         /* The actual number of available elements depends from the
00149            elements blocked by some reservations.
00150            This prevents a consumer to read into reserved area. */
00151         return cb.size() - reserved_for_reading() -
00152            reserved_for_writing();
00152     }
```

Here is the caller graph for this function:



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`.

```
00187     {
00188         std::lock_guard<std::mutex> lg { cb_mutex };
00189         return size();
00190     }
```

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	<i>value</i>	is what we want to write
in	<i>blocking</i>	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`.

```
00218     {
00219         // Lock the pipe to avoid being disturbed
00220         std::unique_lock<std::mutex> ul { cb_mutex };
00221         TRISYCL_DUMP_T("Write pipe full = " << full()
00222             << " value = " << value);
00223
00224         if (blocking)
00225             /* If in blocking mode, wait for the not full condition, that
00226              * may be changed when a read is done */
00227             read_done.wait(ul, [&] { return !full(); });
00228         else if (full())
00229             return false;
00230
00231         cb.push_back(value);
00232         TRISYCL_DUMP_T("Write pipe front = " << cb.front()
00233             << " back = " << cb.back()
00234             << " cb.begin() = " << (void *)&cb.begin()
00235             << " cb.size() = " << cb.size()
00236             << " cb.end() = " << (void *)&cb.end()
00237             << " reserved_for_reading() = " << reserved_for_reading()
00238             << " reserved_for_writing() = " << reserved_for_writing());
00239         // Notify the clients waiting to read something from the pipe
00240         write_done.notify_all();
00241         return true;
00242     }
```

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](#).

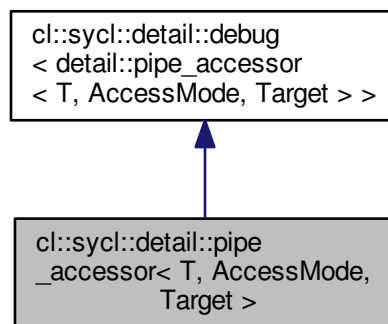
8.1.2.10 `class cl::sycl::detail::pipe_accessor`

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

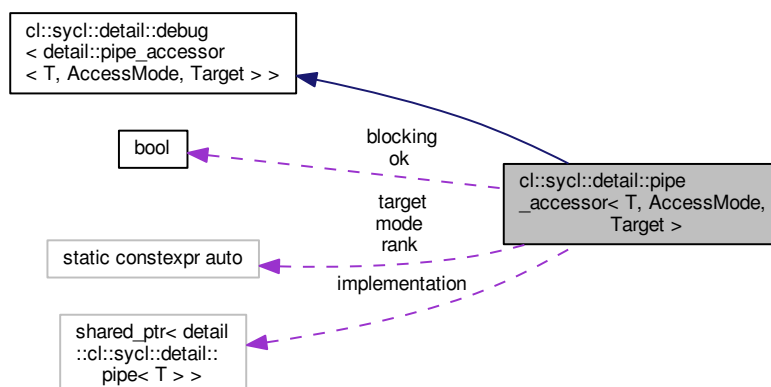
The accessor abstracts the way pipe data are accessed inside a kernel.

Definition at line 44 of file [pipe_accessor.hpp](#).

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



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



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

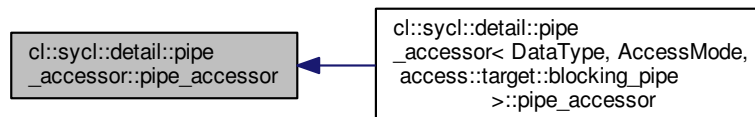
Definition at line 83 of file [pipe_accessor.hpp](#).

```
00084                                     :
00085     implementation { p } {
00086         // TRISYCL_DUMP_T("Create a kernel pipe accessor write = "
00087         //                 << is_write_access());
00088         // Verify that the pipe is not already used in the requested mode
00089         if (mode == access::mode::write)
00090             if (implementation->used_for_writing)
00091                 /// \todo Use pipe_exception instead
00092                 throw std::logic_error { "The pipe is already used for writing." };
00093         else
00094             implementation->used_for_writing = true;
00095     else
00096         if (implementation->used_for_reading)
00097             throw std::logic_error { "The pipe is already used for reading." };
00098         else
00099             implementation->used_for_reading = true;
00100     }
```

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:



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`.

```

00272     {
00273     /// Free the pipe for a future usage for the current mode
00274     if (mode == access::mode::write)
00275         implementation->used_for_writing = false;
00276     else
00277         implementation->used_for_reading = false;
00278     }
  
```

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`.

```

00107     {
00108     return implementation->capacity();
00109     }
  
```

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`.

```

00132     {
00133     return implementation->empty_with_lock();
00134     }
  
```

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](#).

```
00145         {
00146     return implementation->full_with_lock();
00147     }
```

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](#).

```
00267     {
00268     return implementation;
00269     }
```

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](#).

```
00162         {
00163     return ok;
00164     }
```

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](#).

```
00192                                     {
00193     static_assert(mode == access::mode::write,
00194                   "'<<' operator on a pipe accessor is only possible"
00195                   " with write access mode");
00196     // Return a reference to *this so we can apply a sequence of >>
00197     return write(value);
00198 }
```

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](#).

```
00247                                     {
00248     static_assert(mode == access::mode::read,
00249                   "'>>' operator on a pipe accessor is only possible"
00250                   " with read access mode");
00251     // Return a reference to *this so we can apply a sequence of >>
00252     return read(value);
00253 }
```

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](#).

```

00213     {
00214         static_assert(mode == access::mode::read,
00215             "''.read(value_type &value)' method on a pipe accessor"
00216             " is only possible with read access mode");
00217         ok = implementation->read(value, blocking);
00218         // Return a reference to *this so we can apply a sequence of read
00219         return *this;
00220     }

```

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](#).

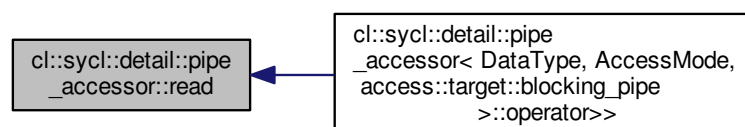
Referenced by [cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >::operator>>\(\)](#).

```

00232     {
00233         static_assert(mode == access::mode::read,
00234             "''.read()' method on a pipe accessor is only possible"
00235             " with read access mode");
00236         static_assert(blocking,
00237             "''.read()' method on a pipe accessor is only possible"
00238             " with a blocking pipe");
00239         value_type value;
00240         implementation->read(value, blocking);
00241         return value;
00242     }

```

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](#).

```
00256                                     {
00257     return { *implementation, size };
00258 }
```

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](#).

```
00262                                     {
00263     implementation->debug_mode = enable;
00264 }
```

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](#).

```
00119                                     {
00120     return implementation->size_with_lock();
00121 }
```

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	<i>value</i>	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](#).

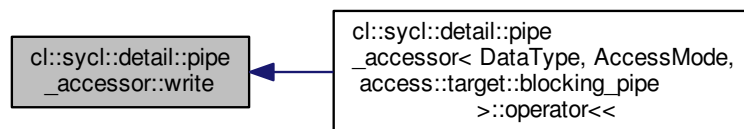
Referenced by [cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >::operator<<\(\)](#).

```

00180                                     {
00181     static_assert(mode == access::mode::write,
00182                   "'.write(const value_type &value)' method on a pipe accessor"
00183                   " is only possible with write access mode");
00184     ok = implementation->write(value, blocking);
00185     // Return a reference to *this so we can apply a sequence of write
00186     return *this;
00187 }

```

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

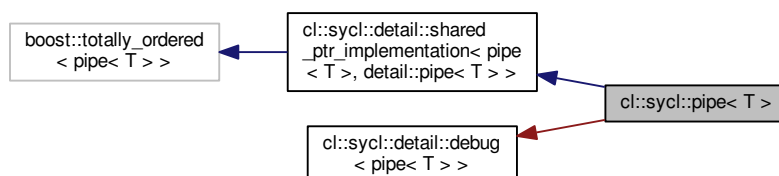
```
template<typename T>
class cl::sycl::pipe< T >
```

A SYCL pipe.

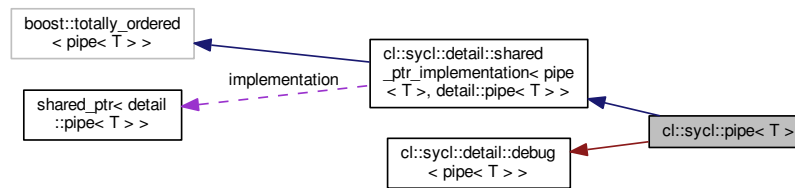
Implement a FIFO-style object that can be used through accessors to send some objects T from the input to the output

Definition at line 29 of file [accessor.hpp](#).

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



Collaboration diagram for `cl::sycl::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](#).

```
00056      : 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](#).

```
00081      {
00082      return implementation->capacity();
00083      }
```

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

	<i>Mode</i>	is the requested access mode
	<i>Target</i>	is the type of pipe access required
in	<i>command_group_handler</i>	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](#).

```
00071      {
00072      static_assert(Target == access::target::pipe
00073                    || Target == access::target::blocking_pipe,
00074                    "get_access(handler) with pipes can only deal with "
00075                    "access::pipe or access::blocking_pipe");
00076      return { implementation, command_group_handler };
00077      }
```

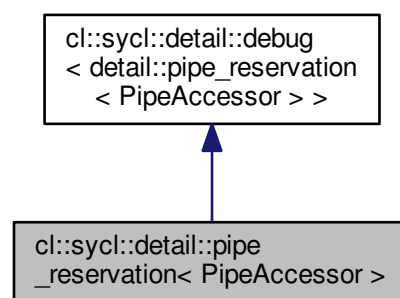
8.1.2.12 class `cl::sycl::detail::pipe_reservation`

```
template<typename PipeAccessor>
class cl::sycl::detail::pipe_reservation< PipeAccessor >
```

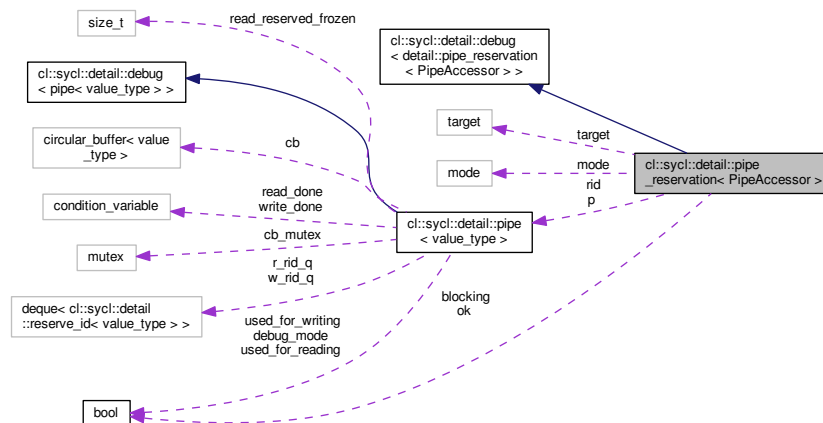
The implementation of the pipe reservation station.

Definition at line 33 of file [pipe_reservation.hpp](#).

Inheritance diagram for `cl::sycl::detail::pipe_reservation< PipeAccessor >`:



Collaboration diagram for `cl::sycl::detail::pipe_reservation< PipeAccessor >`:



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]`

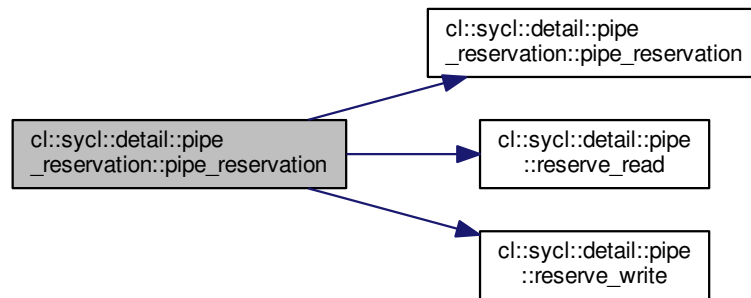
Create a pipe reservation station that reserves the pipe itself.

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](#).

```
00078                                     : p { p } {
00079     static_assert(mode == access::mode::write
00080                   || mode == access::mode::read,
00081                   "A pipe can only be accessed in read or write mode,"
00082                   " exclusively");
00083
00084     /* Since this test is constexpr and dependent of a template
00085        parameter, it should be equivalent to a specialization of the
00086        method but in a clearer way */
00087     if (mode == access::mode::write)
00088         ok = p.reserve_write(s, rid, blocking);
00089     else
00090         ok = p.reserve_read(s, rid, blocking);
00091 }
```


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.

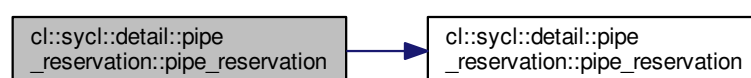
Definition at line 101 of file [pipe_reservation.hpp](#).

References [cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation\(\)](#).

```

00101                                     :
00102     ok {orig.ok },
00103     rid {orig.rid },
00104     p { orig.p } {
00105         /* Even when an object is moved, the destructor of the old
00106            object is eventually called, so leave the old object in a
00107            destructable state but without any commit capability */
00108         orig.ok = false;
00109     }
  
```

Here is the call graph for this function:



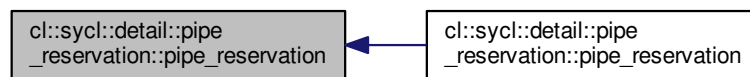
8.1.2.12.2.4 `template<typename PipeAccessor> cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation() [default]`

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:



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.

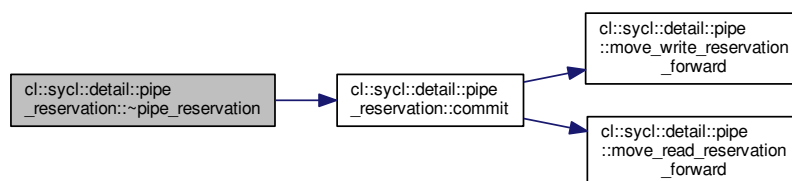
Definition at line 185 of file [pipe_reservation.hpp](#).

References [cl::sycl::detail::pipe_reservation< PipeAccessor >::commit\(\)](#).

```

00185         {
00186     commit();
00187     }
  
```

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.

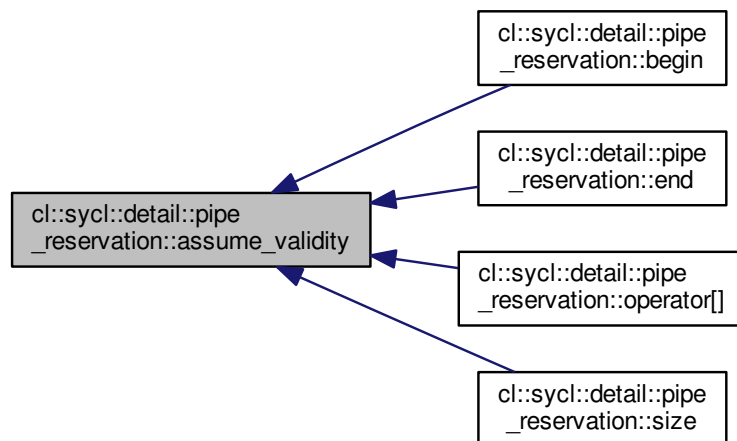
Todo Throw exception instead

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\(\)](#).

```
00071         {
00072     assert(ok);
00073     }
```

Here is the caller graph for this function:



8.1.2.12.3.2 `template<typename PipeAccessor> iterator cl::sycl::detail::pipe_reservation< PipeAccessor >::begin () [inline]`

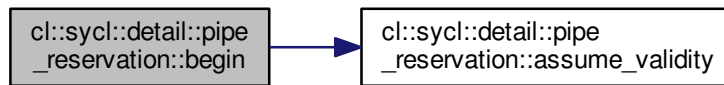
Start of the reservation area.

Definition at line 134 of file [pipe_reservation.hpp](#).

References [cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity\(\)](#).

```
00134     {
00135     assume_validity();
00136     return rid->start;
00137 }
```

Here is the call graph for this function:



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 committed several times but only the first one is taken into account

Definition at line 170 of file [pipe_reservation.hpp](#).

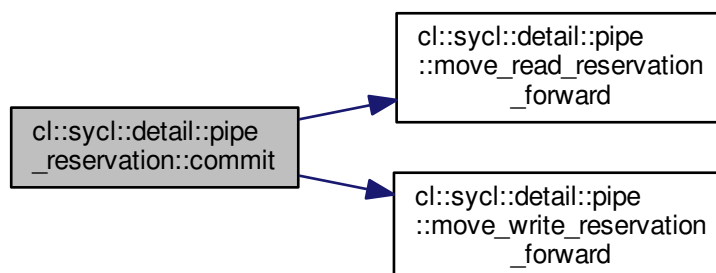
References [cl::sycl::detail::pipe< T >::move_read_reservation_forward\(\)](#), [cl::sycl::detail::pipe< T >::move_write_reservation_forward\(\)](#), [TRISYCL_DUMP_T](#), and [cl::sycl::access::write](#).

Referenced by [cl::sycl::detail::pipe_reservation< PipeAccessor >::~~pipe_reservation\(\)](#).

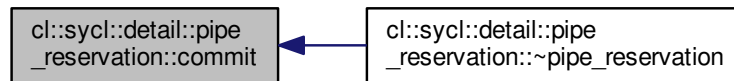
```

00170         {
00171     if (ok) {
00172         // If the reservation is in a committable state, commit
00173         TRISYCL_DUMP_T("Commit");
00174         rid->ready = true;
00175         if (mode == access::mode::write)
00176             p.move_write_reservation_forward();
00177         else
00178             p.move_read_reservation_forward();
00179         ok = false;
00180     }
00181 }
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



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.

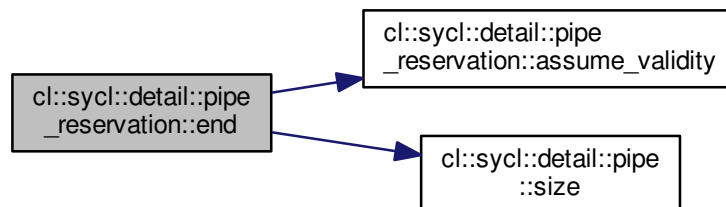
Definition at line 141 of file [pipe_reservation.hpp](#).

References [cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity\(\)](#), and [cl::sycl::detail::pipe< T >::size\(\)](#).

```

00141     {
00142         assume_validity();
00143         return rid->start + rid->size;
00144     }
  
```

Here is the call graph for this function:



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](#).

```

00128     {
00129         return ok;
00130     }
  
```

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](#).

```
00155                                     {
00156     assume_validity();
00157     TRISYCL_DUMP_T("[[] index = " << index
00158                   << " Reservation write address = " << &(rid->start[index]));
00159
00160     return rid->start[index];
00161 }
```

Here is the call graph for this function:



8.1.2.12.3.7 `template<typename PipeAccessor> std::size_t cl::sycl::detail::pipe_reservation< PipeAccessor >::size () [inline]`

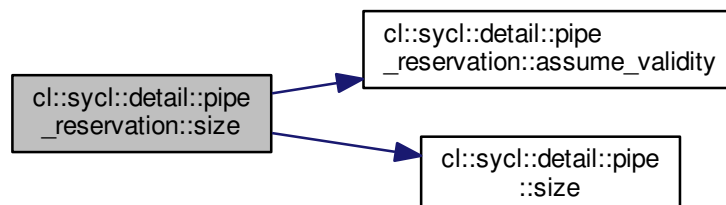
Get the number of elements in the reservation station.

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\(\)](#).

```
00148     {
00149     assume_validity();
00150     return rid->size;
00151 }
```

Here is the call graph for this function:



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.

By 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](#).

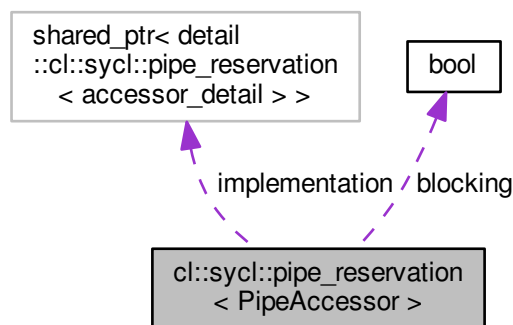
8.1.2.13 struct cl::sycl::pipe_reservation

```
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 >`:



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]`

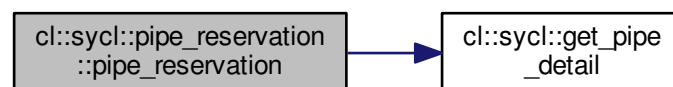
Create a [pipe_reservation](#) for an accessor and a number of elements.

Definition at line 66 of file [pipe_reservation.hpp](#).

References [cl::sycl::get_pipe_detail\(\)](#).

```
00067     : implementation {
00068     new detail::pipe_reservation<accessor_detail> {
00069         get_pipe_detail(accessor), s }
00070     } {}
```

Here is the call graph for this function:



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.

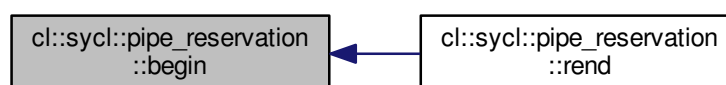
Definition at line 119 of file [pipe_reservation.hpp](#).

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:



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.

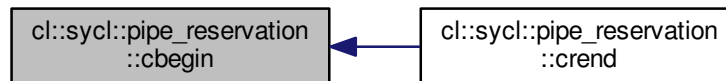
Definition at line 131 of file [pipe_reservation.hpp](#).

References [cl::sycl::pipe_reservation< PipeAccessor >::implementation](#).

Referenced by [cl::sycl::pipe_reservation< PipeAccessor >::crend\(\)](#).

```
00131     {
00132     return implementation->begin();
00133 }
```

Here is the caller graph for this function:



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.

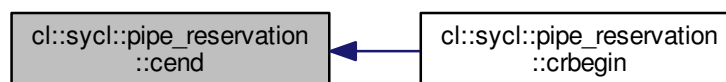
Definition at line 137 of file [pipe_reservation.hpp](#).

References [cl::sycl::pipe_reservation< PipeAccessor >::implementation](#).

Referenced by [cl::sycl::pipe_reservation< PipeAccessor >::crbegin\(\)](#).

```
00137     {
00138     return implementation->end();
00139 }
```

Here is the caller graph for this function:



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](#).

```
00113         {
00114     return implementation->commit();
00115 }
```

8.1.2.13.3.5 `template<typename PipeAccessor > const_reverse_iterator cl::sycl::pipe_reservation< PipeAccessor >::crbegin () const [inline]`

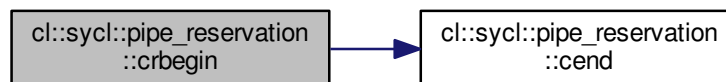
Get a constant reverse iterator on the last element of the reservation station.

Definition at line 157 of file [pipe_reservation.hpp](#).

References [cl::sycl::pipe_reservation< PipeAccessor >::cend\(\)](#).

```
00157         {
00158     return std::make_reverse_iterator(cend());
00159 }
```

Here is the call graph for this function:



8.1.2.13.3.6 `template<typename PipeAccessor > const_reverse_iterator cl::sycl::pipe_reservation< PipeAccessor >::crend () const [inline]`

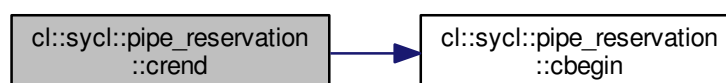
Get a constant reverse iterator on the first element past the end of the reservation station.

Definition at line 164 of file [pipe_reservation.hpp](#).

References [cl::sycl::pipe_reservation< PipeAccessor >::cbegin\(\)](#).

```
00164         {
00165     return std::make_reverse_iterator(cbegin());
00166 }
```

Here is the call graph for this function:



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.

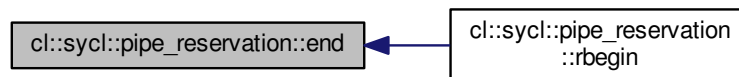
Definition at line 125 of file [pipe_reservation.hpp](#).

References [cl::sycl::pipe_reservation< PipeAccessor >::implementation](#).

Referenced by [cl::sycl::pipe_reservation< PipeAccessor >::rbegin\(\)](#).

```
00125         {
00126     return implementation->end();
00127     }
```

Here is the caller graph for this function:



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](#).

```
00091         {
00092     return *implementation;
00093     }
```

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](#).

```
00103         {
00104     return (*implementation)[index];
00105     }
```

8.1.2.13.3.10 `template<typename PipeAccessor > reverse_iterator cl::sycl::pipe_reservation< PipeAccessor >::rbegin () const [inline]`

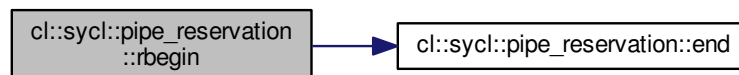
Get a reverse iterator on the last element of the reservation station.

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:



8.1.2.13.3.11 `template<typename PipeAccessor > reverse_iterator cl::sycl::pipe_reservation< PipeAccessor >::rend () const [inline]`

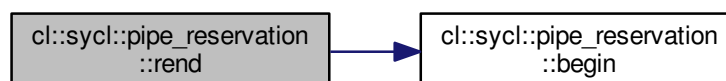
Get a reverse iterator on the first element past the end of the reservation station.

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:



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](#).

```
00097         {
00098     return implementation->size();
00099     }
```

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< PipeAccessor >::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.

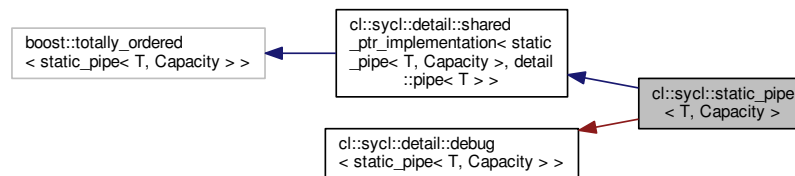
Compared to a normal pipe, a [static_pipe](#) takes a constexpr size and is expected to be declared in a compile-unit static context so the compiler can generate everything at compile time.

This is useful to generate a fixed and optimized hardware implementation on FPGA for example, where the inter-connection graph can be also inferred at compile time.

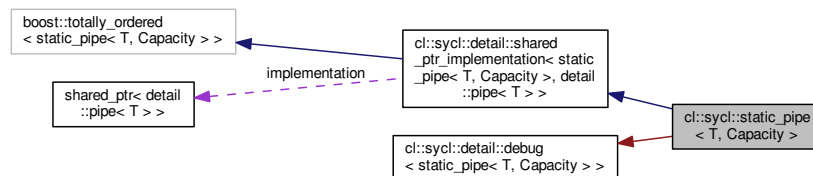
It is not directly mapped to the OpenCL program-scoped pipe because in SYCL there is not this concept of separated program. But the SYCL device compiler is expected to generate some OpenCL program(s) with program-scoped pipes when a SYCL static-scoped pipe is used. These details are implementation defined.

Definition at line 50 of file [static_pipe.hpp](#).

Inheritance diagram for `cl::sycl::static_pipe< T, Capacity >`:



Collaboration diagram for `cl::sycl::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

	<i>Mode</i>	is the requested access mode
	<i>Target</i>	is the type of pipe access required
in	<i>command_group_handler</i>	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](#).

```

00088         {
00089     static_assert(Target == access::target::pipe
00090                   || Target == access::target::blocking_pipe,
00091                   "get_access(handler) with pipes can only deal with "
00092                   "access::pipe or access::blocking_pipe");
00093     return { implementation, command_group_handler };
00094 }

```

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](#).

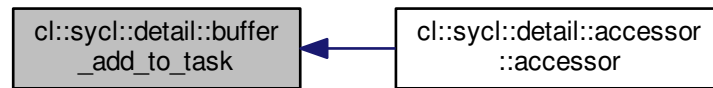
Referenced by [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor\(\)](#).

```

00283     {
00284     return buf->add_to_task(command_group_handler, is_write_mode);
00285     }

```

Here is the caller graph for this function:



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](#).

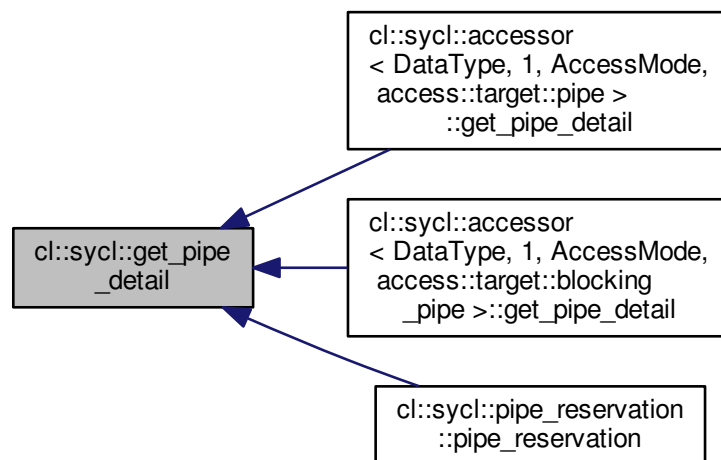
Referenced by `cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >::get_pipe_detail()`, `cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >::get_pipe_detail()`, and `cl::sycl::pipe_reservation< PipeAccessor >::pipe_reservation()`.

```

00414     {
00415     return a.get_pipe_detail();
00416     }

```

Here is the caller graph for this function:



```
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()`.

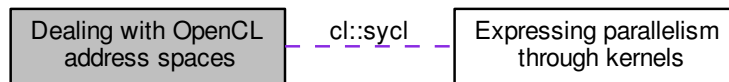
```
00080  
00081     return new buffer_waiter<T, Dimensions> { b };  
00082 }
```

Here is the caller graph for this function:



8.2 Dealing with OpenCL address spaces

Collaboration diagram for Dealing with OpenCL address spaces:



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::opcnl_type`

```
template<typename T, address_space AS>
struct cl::sycl::detail::opcnl_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::openccl_type< T, AS >::type = T`

Definition at line 28 of file [address_space.hpp](#).

8.2.2.2 `struct cl::sycl::detail::openccl_type< T, constant_address_space >`

```
template<typename T>
struct cl::sycl::detail::openccl_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::openccl_type< T, constant_address_space >::type = T`

Definition at line 40 of file [address_space.hpp](#).

8.2.2.3 `struct cl::sycl::detail::openccl_type< T, generic_address_space >`

```
template<typename T>
struct cl::sycl::detail::openccl_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::openccl_type< T, generic_address_space >::type = T`

Definition at line 52 of file [address_space.hpp](#).

8.2.2.4 struct cl::sycl::detail::opencil_type< T, global_address_space >

```
template<typename T>
struct cl::sycl::detail::opencil_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::opencil_type< T, global_address_space >::type = T

Definition at line 64 of file [address_space.hpp](#).

8.2.2.5 struct cl::sycl::detail::opencil_type< T, local_address_space >

```
template<typename T>
struct cl::sycl::detail::opencil_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::opencil_type< T, local_address_space >::type = T

Definition at line 76 of file [address_space.hpp](#).

8.2.2.6 struct cl::sycl::detail::opencil_type< T, private_address_space >

```
template<typename T>
struct cl::sycl::detail::opencil_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::opencil_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 >
```

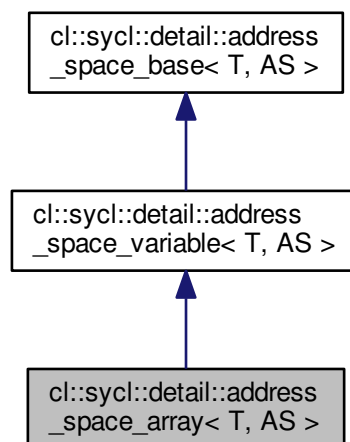
Implementation of an array variable with an OpenCL address space.

Parameters

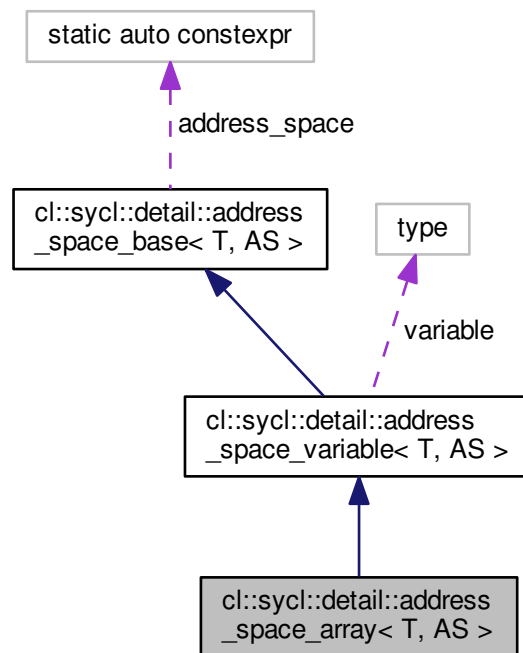
<i>T</i>	is the type of the basic object to be created
<i>AS</i>	is the address space to place the object into

Definition at line 95 of file [address_space.hpp](#).

Inheritance diagram for `cl::sycl::detail::address_space_array< T, AS >`:



Collaboration diagram for `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](#).

```
00316         {
00317     std::copy(std::begin(array), std::end(array), std::begin(super::variable));
00318     };
```

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](#).

```
00325         {
00326     std::copy(std::begin(list), std::end(list), std::begin(super::variable));
00327     };
```

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.

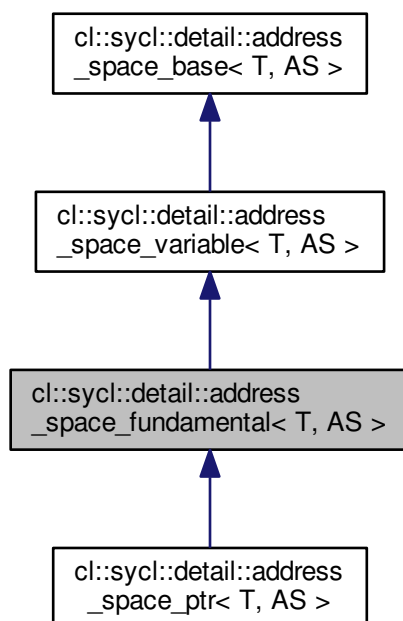
Parameters

<i>T</i>	is the type of the basic object to be created
<i>AS</i>	is the address space to place the object into

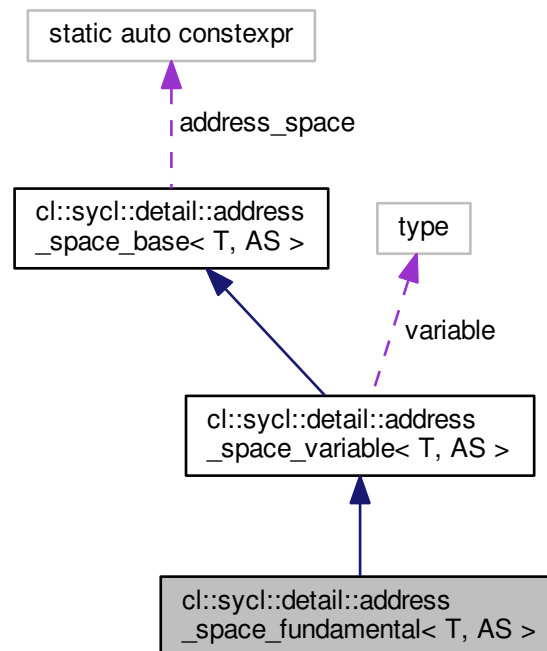
Todo Verify/improve to deal with const/volatile?

Definition at line 98 of file [address_space.hpp](#).

Inheritance diagram for `cl::sycl::detail::address_space_fundamental< T, AS >`:



Collaboration diagram for `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](#).

```
00258 {
00259     /* Strangely I cannot have it working in the initializer instead, for
00260        some cases */
00261     super::variable = SomeType(v);
00262 }
```

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

<i>T</i>	is the type of the basic object to be created
<i>AS</i>	is the address space to place the object into

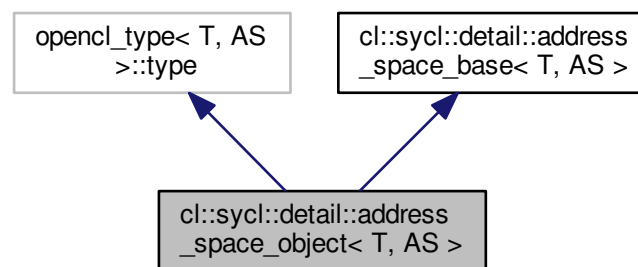
The class implementation is just inheriting of T so that all methods and non-member operators on T work also on `address_space_object<T>`

Todo Verify/improve to deal with const/volatile?

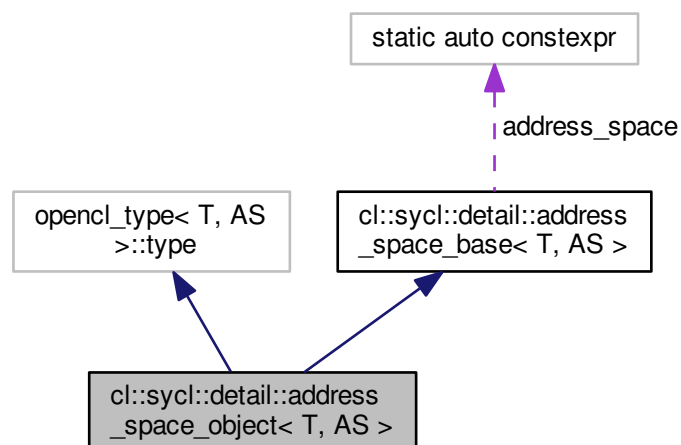
Todo what about T having some final methods?

Definition at line 101 of file [address_space.hpp](#).

Inheritance diagram for `cl::sycl::detail::address_space_object< T, AS >`:



Collaboration diagram for `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 opengl_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 >::opengl_type = typename opengl_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 : opengl_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 opengl_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 [opengl_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

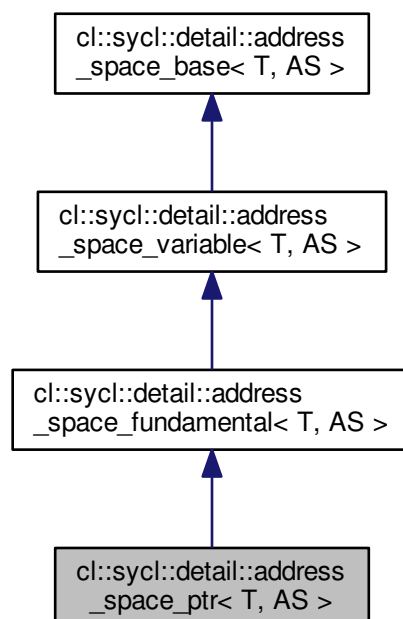
T	is the pointer type
-----	---------------------

Note that if T is not a pointer type, it is an error.

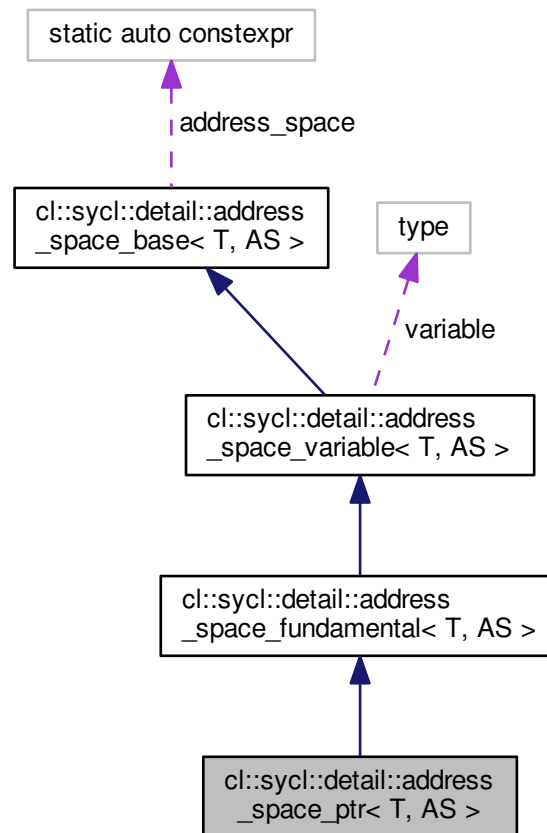
All the address space pointers inherit from it, which makes trivial the implementation of `cl::sycl::multi_ptr<T, AS>`

Definition at line 104 of file [address_space.hpp](#).

Inheritance diagram for `cl::sycl::detail::address_space_ptr< T, AS >`:



Collaboration diagram for `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]`

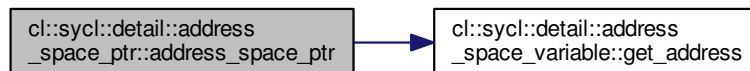
Allow initialization of a pointer type from the address of an element with the same type and address space.

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:



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.

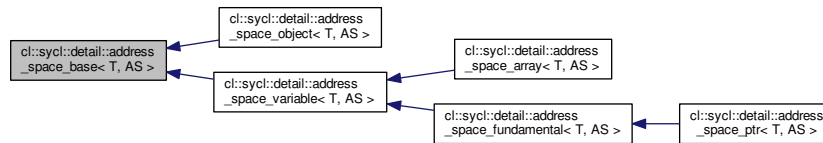
Parameters

<i>T</i>	is the type of the basic stuff to be created
<i>AS</i>	is the address space to place the object into

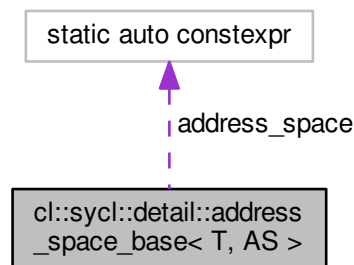
Todo Verify/improve to deal with const/volatile?

Definition at line 135 of file [address_space.hpp](#).

Inheritance diagram for `cl::sycl::detail::address_space_base< T, AS >`:



Collaboration diagram for `cl::sycl::detail::address_space_base< T, AS >`:



Public Types

- using `type` = `T`
Store the base type of the object.
- using `opengl_type` = `typename opengl_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 >::opengl_type = typename opengl_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`.

8.2.2.11.2 `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 >
```

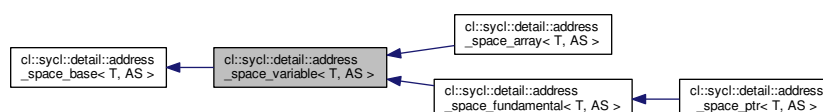
Implementation of a variable with an OpenCL address space.

Parameters

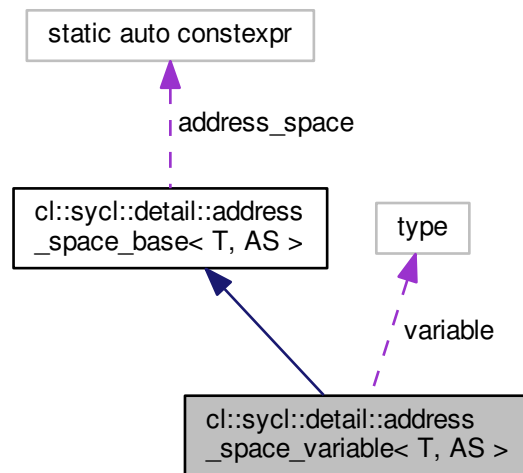
<i>T</i>	is the type of the basic object to be created
<i>AS</i>	is the address space to place the object into

Definition at line 162 of file [address_space.hpp](#).

Inheritance diagram for `cl::sycl::detail::address_space_variable< T, AS >`:



Collaboration diagram for `cl::sycl::detail::address_space_variable< T, AS >`:



Public Types

- using `opengl_type` = `typename opengl_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 opengl_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.
- `opengl_type * get_address ()`
Return the address of the value to implement pointers.

Protected Attributes

- `opengl_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 >::opengl_type = typename opengl_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> opengl_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:



8.2.2.12.3.2 `template<typename T, address_space AS> cl::sycl::detail::address_space_variable< T, AS >::operator opcnl_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 [opcnl_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> opcnl_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

<i>T</i>	is the type of the object to be created
<i>AS</i>	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

<i>T</i>	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

<i>T</i>	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

<i>T</i>	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

<i>T</i>	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

<i>Pointer</i>	is the pointer type
<i>AS</i>	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

<code>T</code>	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

<i>pointer</i>	is the address with its address space to point to
----------------	---

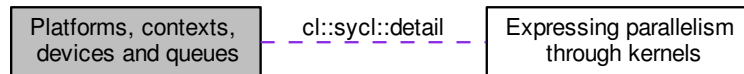
Todo Implement the case with a plain pointer

Definition at line 109 of file [address_space.hpp](#).

```
00109                                     {  
00110     return pointer;  
00111 }
```

8.3 Platforms, contexts, devices and queues

Collaboration diagram for Platforms, contexts, devices and queues:



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_type_name_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...](#)

- using `cl::sycl::default_selector` = device_type_name_selector< info::device_type::defaults >
Devices selected by heuristics of the system.
- using `cl::sycl::gpu_selector` = device_type_name_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_type_name_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_type_name_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

- `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_width_char, cl::sycl::info::device::native_vector_width_short,`
`cl::sycl::info::device::native_vector_width_int,`
`cl::sycl::info::device::native_vector_width_long_long, cl::sycl::info::device::native_vector_width_float, cl::sycl::info::device::native_vector_width_double,`
`cl::sycl::info::device::native_vector_width_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_width, cl::sycl::info::device::image3d_max_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::preferred_interop_device_sync,`

```

::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_↵`
`::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::opcnl_kernel > opcnl_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::opcnl_device > opcnl_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 asyncHandler=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.

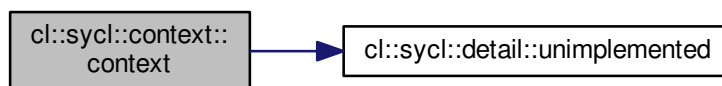
Note that the default case `asyncHandler = nullptr` is handled by the default constructor.

Definition at line 76 of file [context.hpp](#).

References [cl::sycl::detail::unimplemented\(\)](#).

```
00076                                     {
00077     detail::unimplemented();
00078 }
```

Here is the call graph for this function:



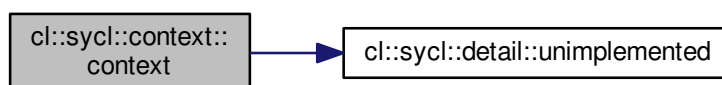
8.3.2.1.1.2 `cl::sycl::context::context (cl_context clContext, async_handler asyncHandler = nullptr) [inline]`

Definition at line 90 of file [context.hpp](#).

References [cl::sycl::detail::unimplemented\(\)](#).

```
00090                                     {
00091     detail::unimplemented();
00092 }
```

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.

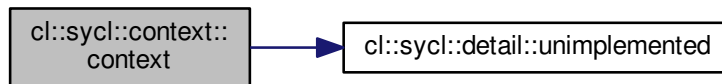
Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the `async_handler`, if provided.

Definition at line 103 of file [context.hpp](#).

References [cl::sycl::detail::unimplemented\(\)](#).

```
00105                                     {
00106     detail::unimplemented();
00107 }
```

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.

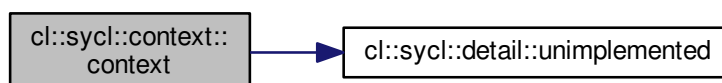
Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the `async_handler`, if provided.

Definition at line 115 of file [context.hpp](#).

References [cl::sycl::detail::unimplemented\(\)](#).

```
00117                                     {
00118     detail::unimplemented();
00119 }
```

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.

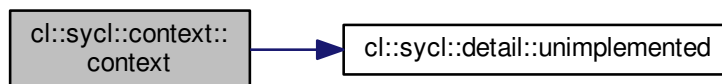
Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the `async_handler`, if provided.

Definition at line 127 of file [context.hpp](#).

References [cl::sycl::detail::unimplemented\(\)](#).

```
00129                                     {
00130     detail::unimplemented();
00131 }
```

Here is the call graph for this function:



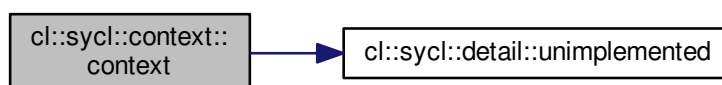
8.3.2.1.1.6 `cl::sycl::context::context (const vector_class< device > & deviceList, info::gl_context_interop interopFlag, async_handler asyncHandler = nullptr) [inline]`

Definition at line 142 of file [context.hpp](#).

References [cl::sycl::info::context](#), and [cl::sycl::detail::unimplemented\(\)](#).

```
00144                                     {
00145     detail::unimplemented();
00146 }
```

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\(\)](#).

```
00165         {  
00166     detail::unimplemented();  
00167     return {};  
00168 }
```

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.

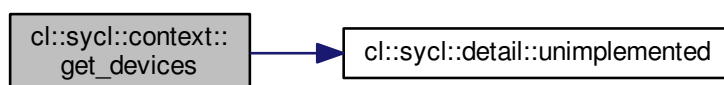
Todo To be implemented

Definition at line 189 of file [context.hpp](#).

References [cl::sycl::detail::unimplemented\(\)](#).

```
00189         {  
00190     detail::unimplemented();  
00191     return {};  
00192 }
```

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.

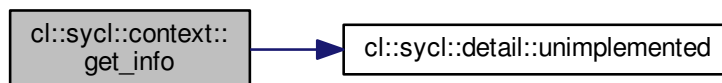
Todo To be implemented

Definition at line 200 of file [context.hpp](#).

References [cl::sycl::detail::unimplemented\(\)](#).

```
00200                                     {
00201     detail::unimplemented();
00202     return {};
00203 }
```

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](#).

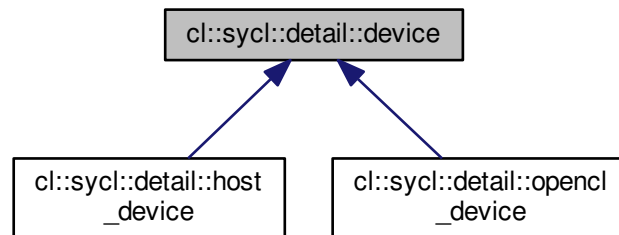
```
00173                                     {
00174     return true;
00175 }
```

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`:



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.

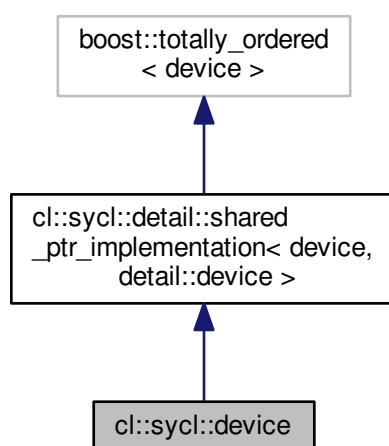
Implemented in [cl::sycl::detail::opencl_device](#), and [cl::sycl::detail::host_device](#).

8.3.2.3 class `cl::sycl::device`

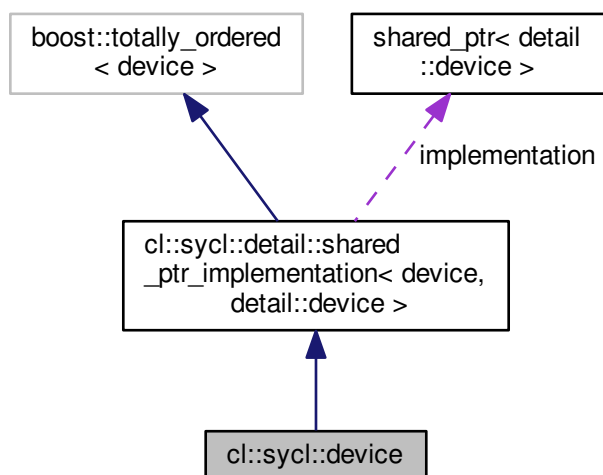
SYCL device.

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::openccl_device::instance\(\)](#).

```
00080 : implementation_t { detail::openccl_device::instance(d)
    } {}
```

Here is the call graph for this function:



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?

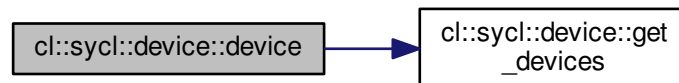
Definition at line 91 of file [device.hpp](#).

References [get_devices\(\)](#), and [cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation](#).

```

00091                                     {
00092     auto devices = device::get_devices();
00093     if (devices.empty())
00094         // \todo Put a SYCL exception
00095         throw std::domain_error("No device at all! Internal error...");
00096
00097     /* Find the device with the best score according to the given
00098        device_selector */
00099     auto max = std::max_element(devices.cbegin(), devices.cend(),
00100                                [&] (const device &d1, const device &d2) {
00101                                    return ds(d1) < ds(d2);
00102                                });
00103     if (ds(*max) < 0)
00104         // \todo Put a SYCL exception
00105         throw std::domain_error("No device selected because no positive "
00106                                   "device_selector score found");
00107
00108     // Create the current device as a shared copy of the selected one
00109     implementation = max->implementation;
00110 }
```

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](#).

```

00124     {
00125     return implementation->get();
00126 }
```

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](#).

```
00199         {
00200         //return implementation->get_info<Param>(param);
00201     }
```

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](#).

```
00211         {
00212         // Forward to the version where the info parameter is not a template
00213         //return get_info<typename info::param_traits_t<info::device, Param>>(Param);
00214     }
```

8.3.2.3.3.4 `platform cl::sycl::device::get_platform () const [inline]`

Return the platform of device.

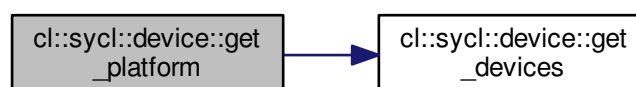
Return synchronous errors via the SYCL exception class.

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](#).

```
00178         {
00179         return implementation->get_platform();
00180     }
```

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](#).

```
00218                                     {
00219     return implementation->has_extension(extension);
00220 }
```

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\(\)](#).

```
00149                                     {
00150     return implementation->is_accelerator();
00151 }
```

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.

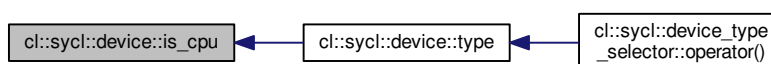
Definition at line 137 of file [device.hpp](#).

References [cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation](#).

Referenced by [type\(\)](#).

```
00137                                     {
00138     return implementation->is_cpu();
00139 }
```

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.

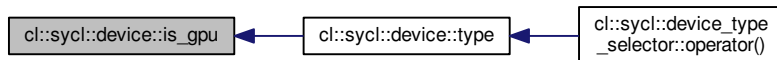
Definition at line 143 of file [device.hpp](#).

References [cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation](#).

Referenced by [type\(\)](#).

```
00143     {
00144     return implementation->is_gpu();
00145 }
```

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.

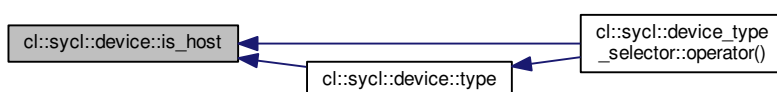
Definition at line 131 of file [device.hpp](#).

References [cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation](#).

Referenced by [cl::sycl::device_type_selector::operator\(\)](#), and [type\(\)](#).

```
00131     {
00132     return implementation->is_host();
00133 }
```

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()`.

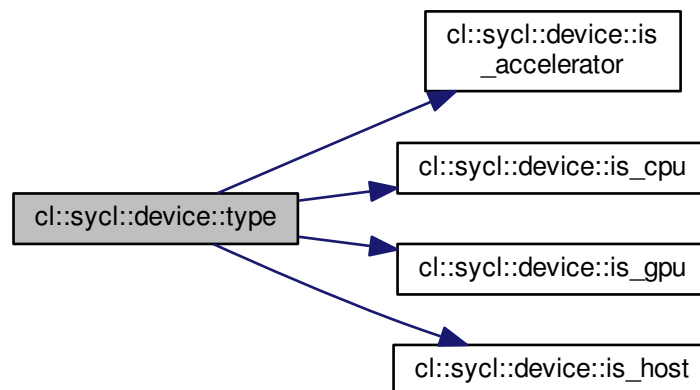
Referenced by `cl::sycl::device_type_selector::operator()()`.

```

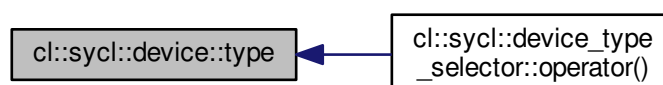
00159         {
00160     if (is_host())
00161         return info::device_type::host;
00162     else if (is_cpu())
00163         return info::device_type::cpu;
00164     else if (is_gpu())
00165         return info::device_type::gpu;
00166     else if (is_accelerator())
00167         return info::device_type::accelerator;
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:



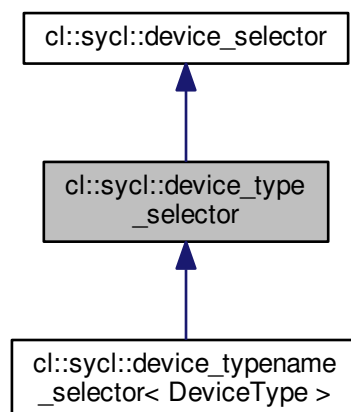
8.3.2.4 class `cl::sycl::device_type_selector`

A device selector by device_type.

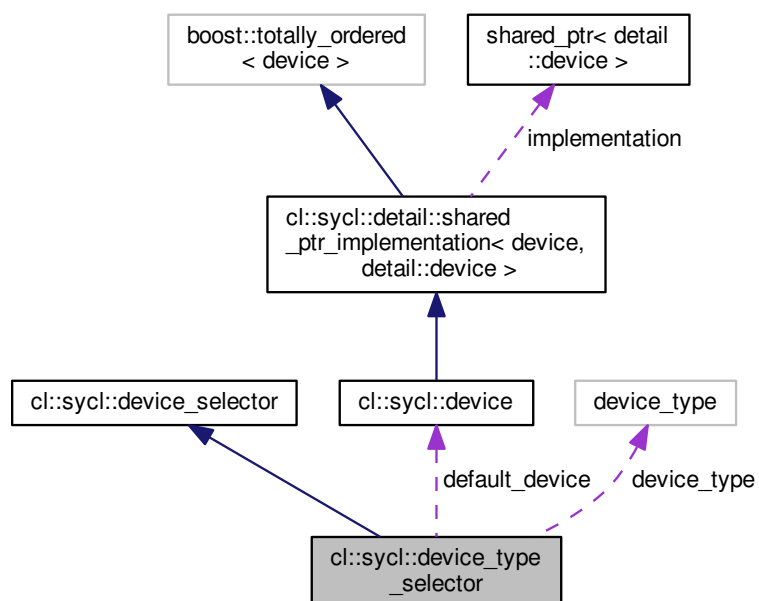
Todo To be added to the specification

Definition at line 28 of file [device_selector_tail.hpp](#).

Inheritance diagram for `cl::sycl::device_type_selector`:



Collaboration diagram for `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]

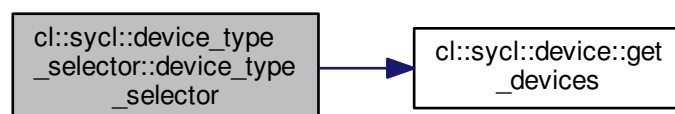
Definition at line 44 of file [device_selector_tail.hpp](#).

References [cl::sycl::info::defaults](#), [cl::sycl::device::get_devices\(\)](#), and [cl::sycl::info::opencl](#).

```

00045     : device_type { device_type } {
00046     // The default device selection heuristic
00047     if (device_type == info::device_type::defaults) {
00048         auto devices = device::get_devices(
00049             info::device_type::opencl);
00049         /* If there is an OpenCL device, pick the first one as the
00050            default device, other wise it is the host device */
00051         if (!devices.empty())
00052             default_device = devices[0];
00053     }
00054 }
```

Here is the call graph for this function:



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](#).

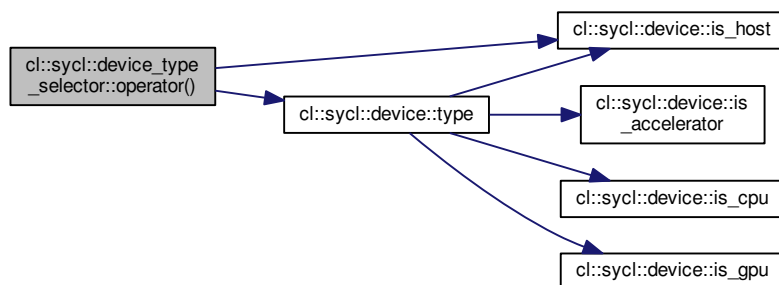
References [cl::sycl::info::all](#), [cl::sycl::info::defaults](#), [cl::sycl::device::is_host\(\)](#), [cl::sycl::info::opencl](#), and [cl::sycl::device::type\(\)](#).

```

00057
00058     if (device_type == info::device_type::all)
00059         // All devices fit all
00060         return 1;
00061
00062     if (device_type == info::device_type::defaults)
00063         // Only select the default device
00064         return dev == default_device ? 1 : -1;
00065
00066     if (device_type == info::device_type::opencl)
00067         // For now, any non host device is an OpenCL device
00068         return dev.is_host() ? -1 : 1;
00069
00070     return dev.type() == device_type ? 1 : -1;
00071 }

```

Here is the call graph for this function:



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`

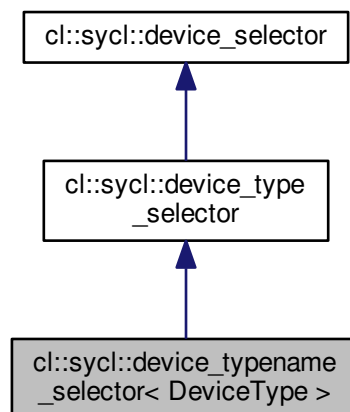
```
template<info::device_type DeviceType>  
class cl::sycl::device_typename_selector< DeviceType >
```

Select a device by template `device_type` parameter.

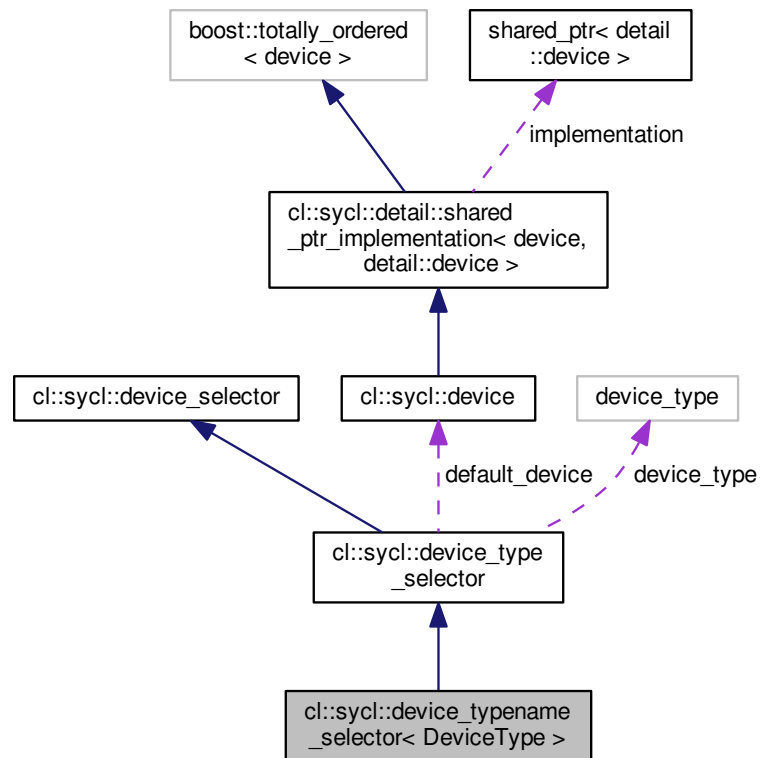
Todo To be added to the specification

Definition at line 81 of file [device_selector_tail.hpp](#).

Inheritance diagram for `cl::sycl::device_typename_selector< DeviceType >`:



Collaboration diagram for `cl::sycl::device_type_name_selector< DeviceType >`:



Public Member Functions

- [device_type_name_selector\(\)](#)

8.3.2.5.1 Constructor & Destructor Documentation

8.3.2.5.1.1 `template<info::device_type DeviceType> cl::sycl::device_type_name_selector< DeviceType >::device_type_name_selector() [inline]`

Definition at line 85 of file [device_selector_tail.hpp](#).

```
00085 : device_type_selector { DeviceType } {}
```

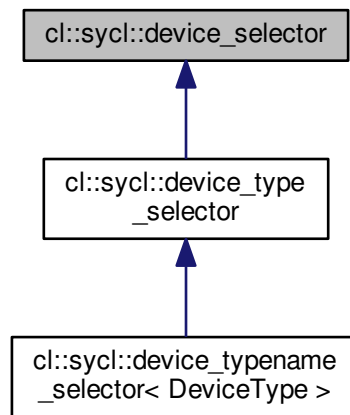
8.3.2.6 class `cl::sycl::device_selector`

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`:



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](#).

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]

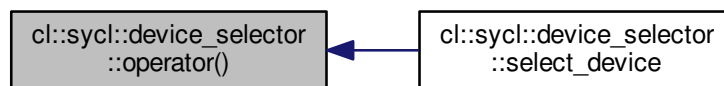
This pure virtual operator allows the customization of device selection.

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:

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

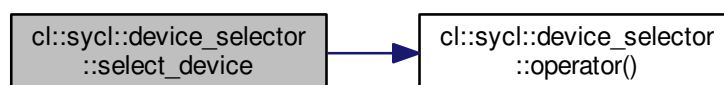
Definition at line 35 of file `device_selector.hpp`.

References `operator()()`.

```

00035                                     {
00036     //     return {};
00037 }
  
```

Here is the call graph for this function:



8.3.2.7 class `cl::sycl::handler`

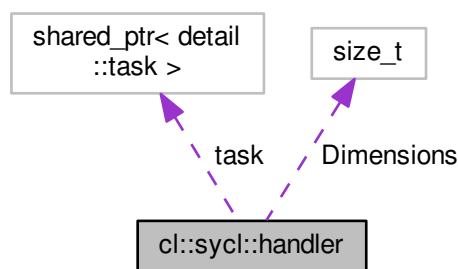
Command group handler class.

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_ParallelForFunctor_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_ParallelForKernel_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](#).

```
00061         {
00062         // Create a new task for this command_group
00063         task = std::make_shared<detail::task>(q);
00064     }
```

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.

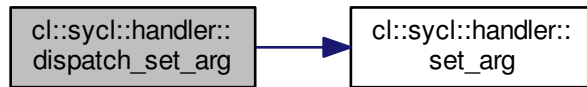
Definition at line 129 of file [handler.hpp](#).

References [set_arg\(\)](#).

Referenced by [set_args\(\)](#).

```
00129         {
00130         // Use an intermediate tuple to ease individual argument access
00131         auto &t = std::make_tuple(std::forward<Ts>(args)...);
00132         // Dispatch individual set_arg() for each argument
00133         auto just_to_evaluate = {
00134             0 /*< At least 1 element to deal with empty set_args() */,
00135             ( set_arg(Is, std::forward<Ts>(std::get<Is>(t))), 0)...
00136         };
00137         // Remove the warning about unused variable
00138         static_cast<void>(just_to_evaluate);
00139     }
```

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]`

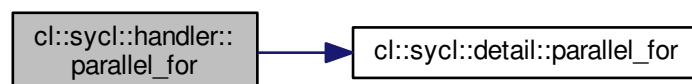
Definition at line 281 of file [handler.hpp](#).

References [cl::sycl::detail::parallel_for\(\)](#).

```

00281                                     {
00282     task->schedule(detail::trace_kernel<KernelName>([=] {
00283         detail::parallel_for(r, f);
00284     }));
00285 }
  
```

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

<i>r</i>	defines the iteration space with the work-group layout and offset
<i>Dimensions</i>	dimensionality of the iteration space
<i>f</i>	is the kernel functor to execute
<i>ParallelForFunctor</i>	is the kernel functor type
<i>KernelName</i>	is a class type that defines the name to be used for the underlying kernel

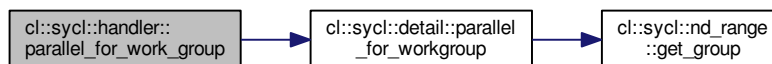
Definition at line 312 of file [handler.hpp](#).

References [cl::sycl::detail::parallel_for_workgroup\(\)](#).

```

00313
00314     task->schedule(detail::trace_kernel<KernelName>({ [=] {
00315         detail::parallel_for_workgroup(r, f); } }));
00316 }
```

Here is the call graph for this function:



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](#).

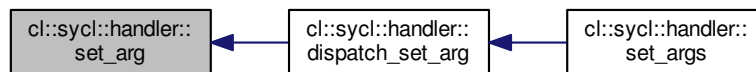
Referenced by [dispatch_set_arg\(\)](#).

```

00087                                     {
00088     /* Before running the kernel, make sure the cl_mem behind this
00089     accessor is up-to-date on the device if needed and pass it to
00090     the kernel.
00091
00092     Explicitly capture task by copy instead of having this captured
00093     by reference and task by reference by side effect */
00094     task->add_prelude([=, task = task] {
00095         acc_obj.implementation->copy_in_cl_buffer();
00096         task->get_kernel().get_boost_compute()
00097             .set_arg(arg_index, acc_obj.implementation->get_cl_buffer());
00098     });
00099     /* After running the kernel, make sure the cl_mem behind this
00100     accessor is up-to-date on the host if needed */
00101     task->add_postlude([=] {
00102         acc_obj.implementation->copy_back_cl_buffer();
00103     });
00104 }

```

Here is the caller graph for this function:



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`.

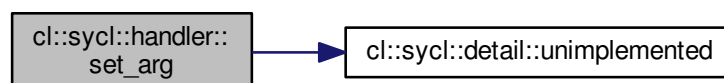
References `cl::sycl::detail::unimplemented()`.

```

00120                                     {
00121     detail::unimplemented();
00122 }

```

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.

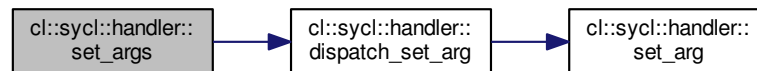
Todo Update the specification to add this function according to https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15978 proposal

Definition at line 150 of file [handler.hpp](#).

References [dispatch_set_arg\(\)](#).

```
00150         {
00151         /* Construct a set of increasing argument index to be able to call
00152            the real set_arg */
00153         dispatch_set_arg (std::make_index_sequence<sizeof...(Ts)>{}),
00154                          std::forward<Ts>(args) ...);
00155     }
```

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

<i>F</i>	specify the kernel to be launched as a single_task
<i>KernelName</i>	is a class type that defines the name to be used for the underlying kernel

Definition at line 173 of file [handler.hpp](#).

```
00173         {
00174         task->schedule (detail::trace_kernel<KernelName>(F));
00175     }
```

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

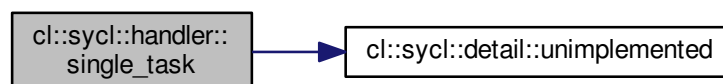
Todo To be implemented

Definition at line 327 of file [handler.hpp](#).

References [cl::sycl::detail::unimplemented\(\)](#).

```
00327                                     {
00328     detail::unimplemented();
00329 }
```

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

<i>global_size</i>	is the global size of the range<>
<i>offset</i>	is the offset to be add to the id<> during iteration
<i>f</i>	is the kernel functor to execute
<i>ParallelForFunctor</i>	is the kernel functor type
<i>KernelName</i>	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

<i>r</i>	defines the iteration space with the work-group layout and offset
<i>Dimensions</i>	dimensionality of the iteration space
<i>f</i>	is the kernel functor to execute
<i>ParallelForFunctor</i>	is the kernel functor type
<i>KernelName</i>	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

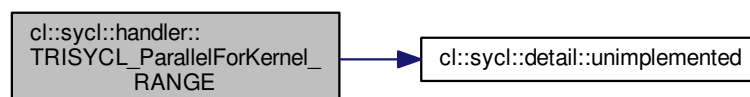
Definition at line 365 of file [handler.hpp](#).

References [cl::sycl::detail::unimplemented\(\)](#).

```

00381                                     {
00382     detail::unimplemented();
00383 }
```

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](#).

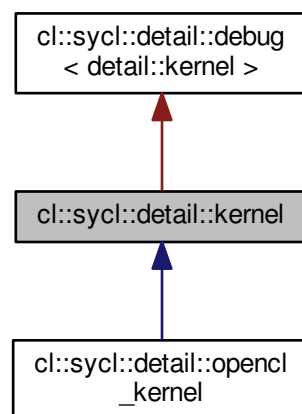
Referenced by [cl::sycl::detail::add_buffer_to_task\(\)](#).

8.3.2.8 class `cl::sycl::detail::kernel`

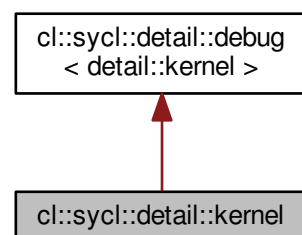
Abstract SYCL kernel.

Definition at line 31 of file [kernel.hpp](#).

Inheritance diagram for `cl::sycl::detail::kernel`:



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_ParallelForKernel_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`

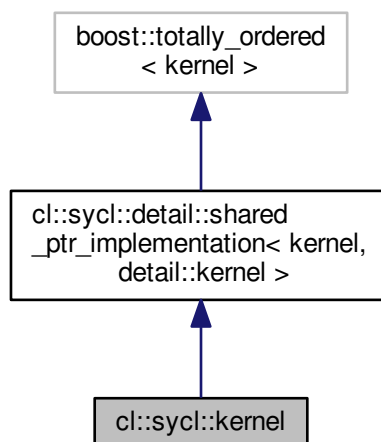
SYCL kernel.

Todo To be implemented

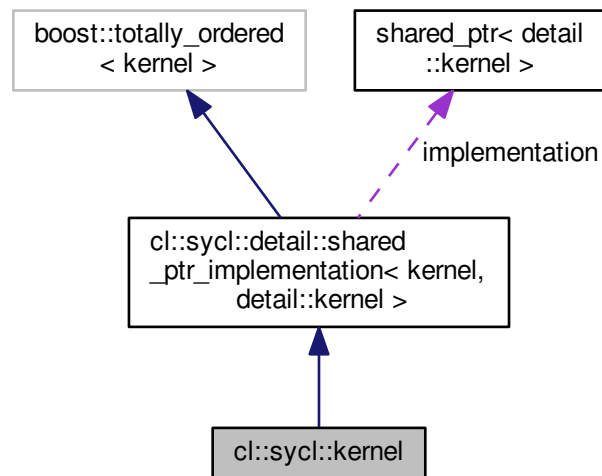
Todo Check specification

Definition at line 38 of file [kernel.hpp](#).

Inheritance diagram for `cl::sycl::kernel`:



Collaboration diagram for `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 enqueueing.
- `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 enqueueing.

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.

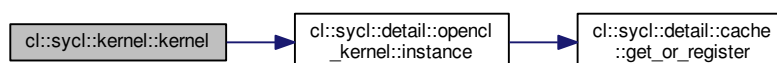
Return synchronous errors via the SYCL exception class.

Definition at line 76 of file [kernel.hpp](#).

References [cl::sycl::detail::opengl_kernel::instance\(\)](#).

```
00077 : implementation_t { detail::opengl_kernel::instance(k)
    } {}
```

Here is the call graph for this function:



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\(\)](#).

```
00085         {
00086     return implementation->get();
00087     }
```

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]`

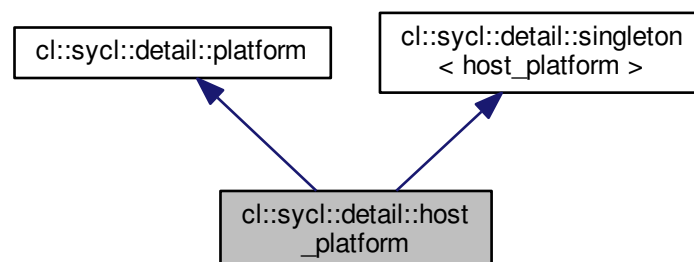
Definition at line 51 of file [kernel.hpp](#).

8.3.2.10 `class cl::sycl::detail::host_platform`

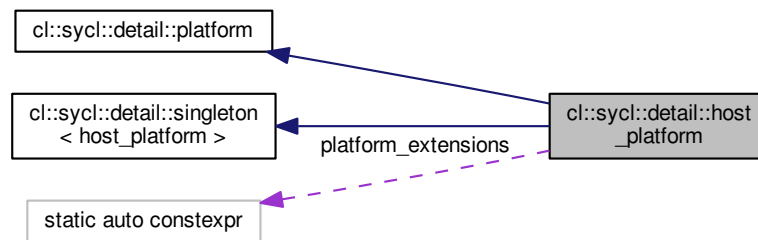
SYCL host platform.

Definition at line 31 of file [host_platform.hpp](#).

Inheritance diagram for `cl::sycl::detail::host_platform`:



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](#).

```

00045                                     {
00046     throw non_cl_error("The host platform has no OpenCL platform");
00047 }
  
```

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`.

```

00079                                     {
00080     switch (param) {
00081     case info::platform::profile:
00082         /* Well... Is the host platform really a full profile whereas it
00083          * is not really OpenCL? */
00084         return "FULL_PROFILE";
00085     case info::platform::version:
00086         // \todo I guess it should include the software version too...
00087         return "2.2";
00088     case info::platform::name:
00089         return "triSYCL host platform";
00090     case info::platform::vendor:
00091         return "triSYCL Open Source project";
00092     case info::platform::extensions:
00093         return platform_extensions;
00094     default:
00095         // \todo Define some SYCL exception type for this type of errors
00096         throw std::invalid_argument {
00097             "Unknown parameter value for SYCL platform information" };
00098     }
00099 }
00100
00101
00102
00103
00104

```

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`.

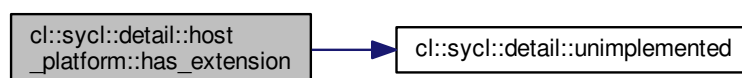
References `cl::sycl::detail::unimplemented()`.

```

00111                                     {
00112     detail::unimplemented();
00113     return {};
00114 }

```

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.

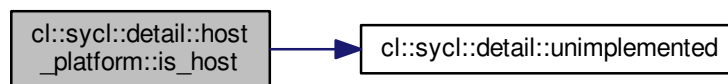
Implements [cl::sycl::detail::platform](#).

Definition at line 52 of file [host_platform.hpp](#).

References [cl::sycl::info::all](#), and [cl::sycl::detail::unimplemented\(\)](#).

```
00052                                     {
00053     return true;
00054 }
```

Here is the call graph for this function:



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](#).

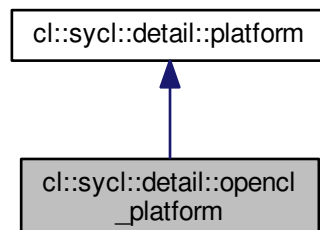
Referenced by [get_info_string\(\)](#).

8.3.2.11 `class cl::sycl::detail::opencl_platform`

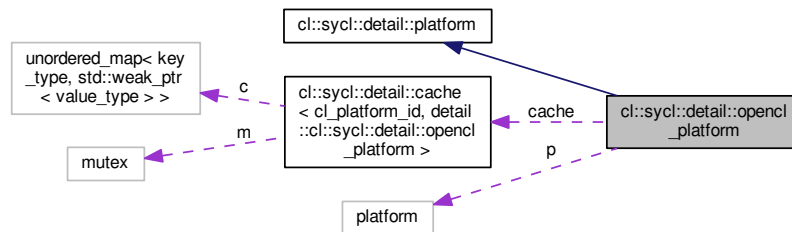
SYCL OpenCL platform.

Definition at line 36 of file [opencl_platform.hpp](#).

Inheritance diagram for `cl::sycl::detail::opencl_platform`:



Collaboration diagram for `cl::sycl::detail::opencl_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](#).

```
00106 : p { p } {}
```

8.3.2.11.1.2 `cl::sycl::detail::opencl_platform::~~opencl_platform () [inline],[override]`

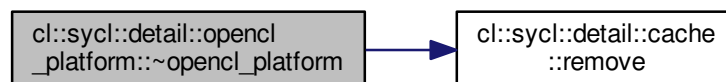
Unregister from the cache on destruction.

Definition at line 111 of file [opencl_platform.hpp](#).

References [cl::sycl::detail::__attribute__](#), [cache](#), and [cl::sycl::detail::cache< Key, Value >::remove\(\)](#).

```
00111                                     {
00112     cache.remove(p.id());
00113 }
```

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](#).

```
00051                                     {
00052     return p.id();
00053 }
```

8.3.2.11.2.2 `string_class cl::sycl::detail::openccl_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 [openccl_platform.hpp](#).

```
00082                                     {
00083     /* Use the fact that the triSYCL info values are the same as the
00084        OpenCL ones used in Boost.Compute to just cast the enum class
00085        to the int value */
00086     return p.get_info<std::string>(static_cast<cl_platform_info>(param));
00087 }
```

8.3.2.11.2.3 `bool cl::sycl::detail::openccl_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 [openccl_platform.hpp](#).

```
00091                                     {
00092     return p.supports_extension(extension);
00093 }
```

8.3.2.11.2.4 `static std::shared_ptr<openccl_platform> cl::sycl::detail::openccl_platform::instance (const boost::compute::platform & p)` `[inline], [static]`

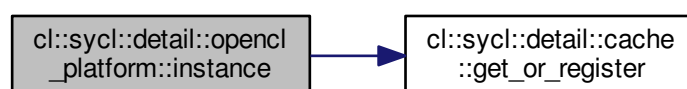
Definition at line 98 of file [openccl_platform.hpp](#).

References [cl::sycl::detail::cache< Key, Value >::get_or_register\(\)](#).

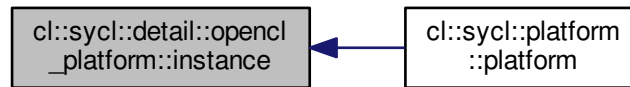
Referenced by [cl::sycl::platform::platform\(\)](#).

```
00098                                     {
00099     return cache.get_or_register(p.id(),
00100                                [&] { return new openccl_platform {
00101     p }; });
00101 }
```

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::opengl_platform::is_host () const` `[inline],[override],[virtual]`

Return false since an OpenGL platform is not the SYCL host platform.

Implements [cl::sycl::detail::platform](#).

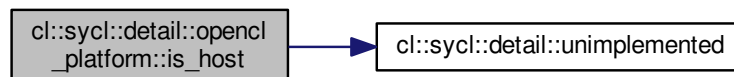
Definition at line 57 of file [opengl_platform.hpp](#).

References [cl::sycl::info::all](#), and [cl::sycl::detail::unimplemented\(\)](#).

```

00057                                     {
00058     return false;
00059 }
  
```

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::opengl_platform> cl::sycl::detail::opengl_platform::cache` `[static],[private]`

A cache to always return the same live platform for a given OpenGL platform.

C++11 guaranties the static construction is thread-safe

Definition at line 46 of file [opengl_platform.hpp](#).

Referenced by [~opengl_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.

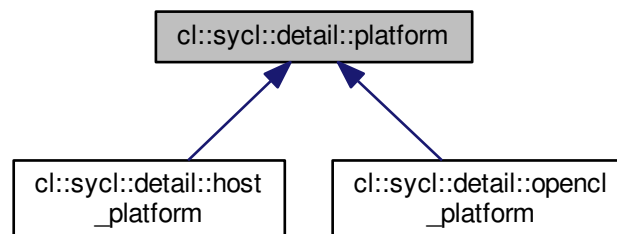
Definition at line 39 of file [opencl_platform.hpp](#).

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`:



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](#).

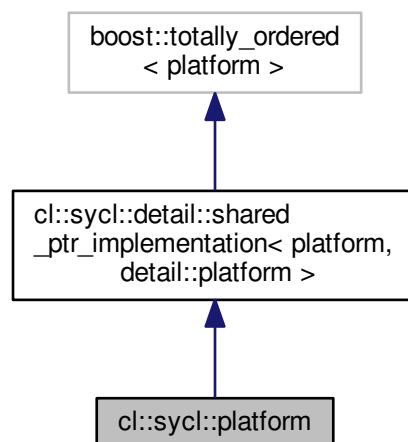
8.3.2.13 class cl::sycl::platform

Abstract the OpenCL platform.

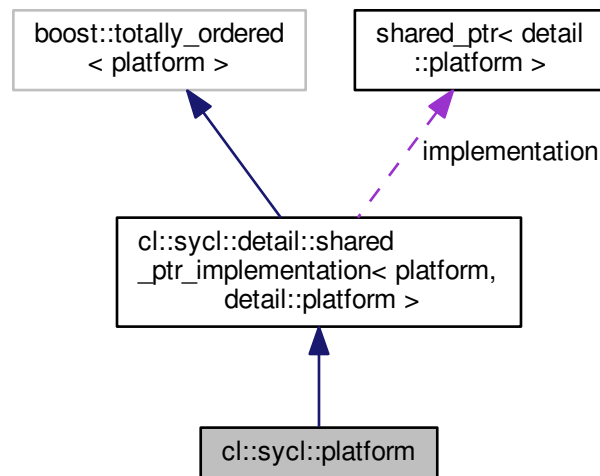
Todo triSYCL Implementation

Definition at line 43 of file [platform.hpp](#).

Inheritance diagram for `cl::sycl::platform`:



Collaboration diagram for `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.

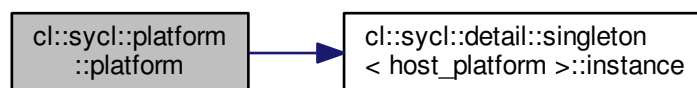
Returns errors via the SYCL exception class.

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:



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::opengl_platform::instance\(\)](#).

```
00084      : implementation_t { detail::opengl_platform::instance
      (p) } {}
```

Here is the call graph for this function:



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.

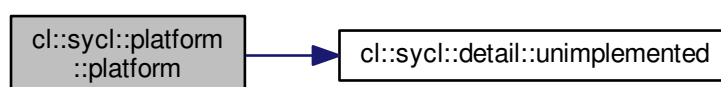
Returns errors via the SYCL exception class.

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:



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.

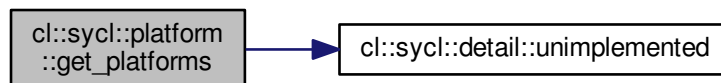
Definition at line 113 of file [platform.hpp](#).

References [cl::sycl::info::all](#), and [cl::sycl::detail::unimplemented\(\)](#).

```

00113                                     {
00114     // Start with the default platform
00115     vector_class<platform> platforms { {} };
00116
00117     #ifdef TRISYCL_OPENCL
00118     // Then add all the OpenCL platforms
00119     for (const auto &d : boost::compute::system::platforms())
00120         platforms.emplace_back(d);
00121     #endif
00122
00123     return platforms;
00124 }
```

Here is the call graph for this function:



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](#).

```

00167                                     {
00168     return implementation->has_extension(extension);
00169 }
```

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](#).

```

00173                                     {
00174     return implementation->is_host();
00175 }
```

8.3.2.14 class `cl::sycl::queue`

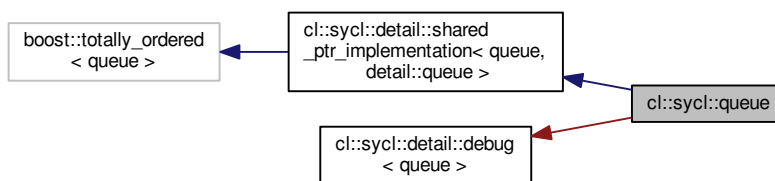
SYCL queue, similar to the OpenCL queue concept.

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

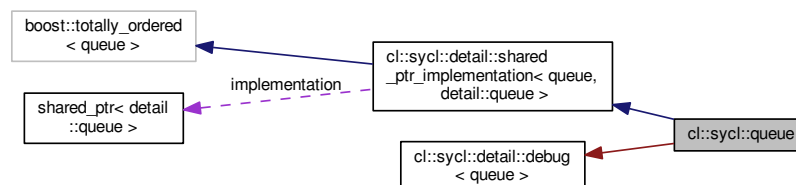
Todo All the queue methods should return a `queue&` instead of `void` to it is possible to chain operations

Definition at line 80 of file [queue.hpp](#).

Inheritance diagram for `cl::sycl::queue`:



Collaboration diagram for `cl::sycl::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](#) asyncHandler=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\(\)](#).

```
00117                                     : queue { } {
00118     detail::unimplemented();
00119 }
```

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\(\)](#).

```
00131                                     : queue { } {
00132     detail::unimplemented();
00133 }
```

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\(\)](#).

```
00141                                     : queue { } {
00142     detail::unimplemented();
00143 };
```

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\(\)](#).

```
00159                                     : queue { } {
00160     detail::unimplemented();
00161 };
```

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\(\)](#).

```
00175                                     : queue { } {
00176     detail::unimplemented();
00177 }
```

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\(\)](#).

```
00194                                     : queue { } {
00195     detail::unimplemented();
00196 }
```

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.

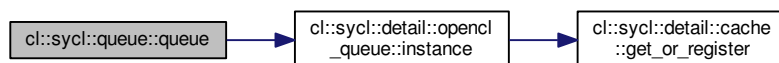
Todo Deal with handler

Definition at line 221 of file [queue.hpp](#).

References [cl::sycl::detail::openccl_queue::instance\(\)](#).

```
00222      : implementation_t { detail::openccl_queue::instance (q) }
      {}
```

Here is the call graph for this function:



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](#).

```
00247     {
00248     return implementation->get_context();
00249 }
```

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](#).

```
00256     {
00257     return implementation->get_device();
00258 }
```

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\(\)](#).

```
00306     {
00307     detail::unimplemented();
00308     return {};
00309 }
```

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](#).

```
00262         {
00263     return implementation->is_host();
00264 }
```

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](#).

```
00324         {
00325     handler command_group_handler { implementation };
00326     cgf(command_group_handler);
00327     return {};
00328 }
```

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\(\)](#).

```
00340         {
00341     detail::unimplemented();
00342     // Since it is not implemented, always submit on the main queue
00343     return submit(cgf);
00344 }
```

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.

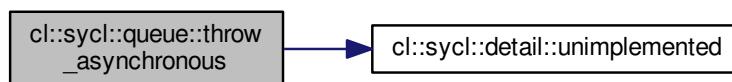
If no `async_handler` was provided then asynchronous exceptions will be lost.

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:



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.

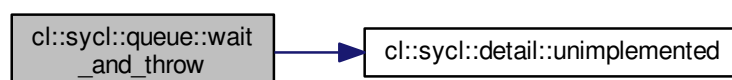
If no `async_handler` was provided then asynchronous exceptions will be lost.

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:



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_type_name_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_width_char*
- native_vector_width_short*
- native_vector_width_int*
- native_vector_width_long_long*
- native_vector_width_float*
- native_vector_width_double*
- native_vector_width_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](#).

```

00052         : int {
00053     device_type,
00054     vendor_id,
00055     max_compute_units,
00056     max_work_item_dimensions,
00057     max_work_item_sizes,
00058     max_work_group_size,
00059     preferred_vector_width_char,
00060     preferred_vector_width_short,
00061     preferred_vector_width_int,
00062     preferred_vector_width_long_long,
00063     preferred_vector_width_float,
00064     preferred_vector_width_double,
00065     preferred_vector_width_half,
00066     native_vector_witdth_char,
00067     native_vector_witdth_short,
00068     native_vector_witdth_int,
00069     native_vector_witdth_long_long,
00070     native_vector_witdth_float,
00071     native_vector_witdth_double,
00072     native_vector_witdth_half,
00073     max_clock_frequency,
00074     address_bits,
00075     max_mem_alloc_size,
00076     image_support,
00077     max_read_image_args,
00078     max_write_image_args,
00079     image2d_max_height,
00080     image2d_max_width,
00081     image3d_max_height,
00082     image3d_max_widht,
00083     image3d_mas_depth,
00084     image_max_buffer_size,
00085     image_max_array_size,
00086     max_samplers,
00087     max_parameter_size,
00088     mem_base_addr_align,
00089     single_fp_config,
00090     double_fp_config,
00091     global_mem_cache_type,
00092     global_mem_cache_line_size,
00093     global_mem_cache_size,
00094     global_mem_size,
00095     max_constant_buffer_size,
00096     max_constant_args,
00097     local_mem_type,
00098     local_mem_size,
00099     error_correction_support,
00100     host_unified_memory,
00101     profiling_timer_resolution,
00102     endian_little,
00103     is_available,
00104     is_compiler_available,
00105     is_linker_available,
00106     execution_capabilities,
00107     queue_properties,
00108     built_in_kernels,
00109     platform,
00110     name,
00111     vendor,
00112     driver_version,
00113     profile,
00114     device_version,
00115     opencl_version,
00116     extensions,
00117     printf_buffer_size,
00118     preferred_interop_user_sync,
00119     parent_device,
00120     partition_max_sub_devices,
00121     partition_properties,
00122     partition_affinity_domain,
00123     partition_type,
00124     reference_count
00125 };

```


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](#).

```
00135                                     : int {
00136     unsupported,
00137     numa,
00138     L4_cache,
00139     L3_cache,
00140     L2_cache,
00141     next_partitionable
00142 };
```

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](#).

```
00176                                     : unsigned int {
00177     exec_kernel,
00178     exec_native_kernel
00179 };
```

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](#).

```
00127                                     : int {
00128     unsupported,
00129     partition_equally,
00130     partition_by_counts,
00131     partition_by_affinity_domain,
00132     partition_affinity_domain_next_partitionable
00133 };
```

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](#).

```
00144                                     : int {
00145     no_partition,
00146     numa,
00147     L4_cache,
00148     L3_cache,
00149     L2_cache,
00150     L1_cache
00151 };
```

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 opcnl to the specification

Todo there is no accelerator_selector and custom_accelerator

Enumerator

cpu
gpu
accelerator
custom
defaults
host
opcnl
all

Definition at line 34 of file [device.hpp](#).

```
00034                                     : unsigned int {
00035     cpu,
00036     gpu,
00037     accelerator,
00038     custom,
00039     defaults,
00040     host,
00041     opcnl,
00042     all
00043 };
```

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](#).

```
00159                                     : int {  
00160     denorm,  
00161     inf\_nan,  
00162     round\_to\_nearest,  
00163     round\_to\_zero,  
00164     round\_to\_inf,  
00165     fma,  
00166     correctly\_rounded\_divide\_sqrt,  
00167     soft\_float  
00168 };
```

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](#).

```
00170                                     : int {  
00171     none,  
00172     read\_only,  
00173     write\_only  
00174 };
```

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](#).

```
00153             : int {
00154     none,
00155     local,
00156     global
00157 };
```

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](#).

```
00031             : unsigned int {
00032     /** Returns the profile name (as a string_class) supported by the
00033         implementation.
00034
00035         Can be either FULL PROFILE or EMBEDDED PROFILE.
00036     */
00037     profile TRISYCL_SKIP_OPENCL(= CL_PLATFORM_PROFILE),
00038
00039     /** Returns the OpenCL software driver version string in the form major
00040         number.minor number (as a string_class)
00041     */
00042     version TRISYCL_SKIP_OPENCL(= CL_PLATFORM_VERSION),
00043
00044     /** Returns the name of the platform (as a string_class)
00045     */
00046     name TRISYCL_SKIP_OPENCL(= CL_PLATFORM_NAME),
```

```

00047
00048 /** Returns the string provided by the platform vendor (as a string_class)
00049 */
00050 vendor TRISYCL_SKIP_OPENCL(= CL_PLATFORM_VENDOR),
00051
00052 /** Returns a space-separated list of extension names supported by the
00053     platform (as a string_class)
00054 */
00055 extensions TRISYCL_SKIP_OPENCL(= CL_PLATFORM_EXTENSIONS),
00056
00057 #if CL_SYCL_LANGUAGE_VERSION >= 220 && defined(CL_VERSION_2_1)
00058 /** Returns the resolution of the host timer in nanoseconds as used by
00059     clGetDeviceAndHostTimer
00060 */
00061 host_timer_resolution
00062     TRISYCL_SKIP_OPENCL(= CL_PLATFORM_HOST_TIMER_RESOLUTION)
00063 #endif
00064 };

```

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](#).

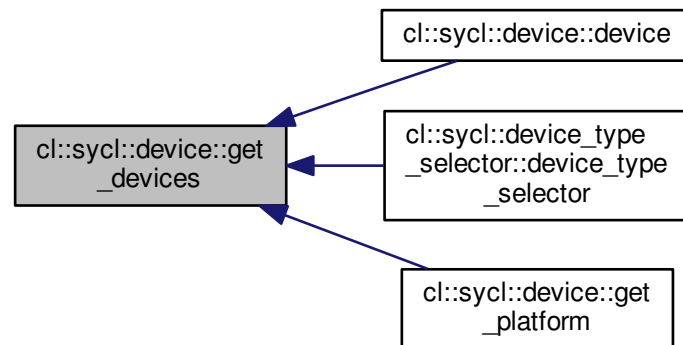
Referenced by [cl::sycl::device::device\(\)](#), [cl::sycl::device_type_selector::device_type_selector\(\)](#), and [cl::sycl::device::get_platform\(\)](#).

```

00026
00027 // Start with the default device
00028 vector_class<device> devices = { {} };
00029
00030 #ifndef TRISYCL_OPENCL
00031 // Then add all the OpenCL devices
00032 for (const auto &d : boost::compute::system::devices())
00033     devices.emplace_back(d);
00034 #endif
00035
00036 // The selected devices
00037 vector_class<device> sd;
00038 device_type_selector s { device_type };
00039
00040 // Return the devices with the good criterion according to the selector
00041 std::copy_if(devices.begin(), devices.end(), std::back_inserter(sd),
00042     [&](const device &e) { return s(e) >= 0; });
00043 return sd;
00044 }

```

Here is the caller graph for this function:



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_RANGE()`, `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]

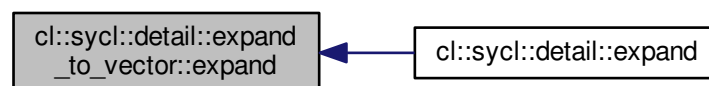
```
#include <include/CL/sycl/detail/array_tuple_helpers.hpp>
```

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.

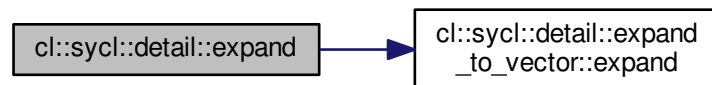
If there is only 1 initializer, this is a scalar initialization of a vector and the value is expanded to all the vector elements first.

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

<i>value</i>	is used to initialize each tuple element
<i>size</i>	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](#).

```

00093                                     {
00094     /* The effect is like a static for-loop with Is counting from 0 to
00095        dimension-1 and thus replicating the pattern to have
00096        make_tuple( (0, e), (1, e), ... (n - 1, e) )
00097
00098        Since the "," operator is just here to throw away the Is value
00099        (which is needed for the pack expansion...), at the end this is
00100        equivalent to:
00101        make_tuple( e, e, ..., e )
00102    */
00103    return std::make_tuple(((void)Is, e)...);
00104 }

```

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](#).

```

00053                                     {
00054     /* Construct an index_sequence with 0, 1, ..., (size of the tuple-1)
00055        so that tuple_to_array_iterate can statically iterate on it */
00056     return tuple_to_array_iterate<V>(t,
00057                                     std::make_index_sequence<std::tuple_size<Tuple>::value>{});
00058 }

```

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](#).

```

00037                                     {
00038     /* The effect is like a static for-loop with Is counting from 0 to
00039        dimension-1 and thus constructing a uniform initialization { }
00040        construction from each tuple element:
00041        { std::get<0>(t), std::get<1>(t), ..., std::get<dimension-1>(t) }
00042
00043        The static cast is here to avoid the warning when there is a loss
00044        of precision, for example when initializing an int from a float.
00045    */
00046     return { { static_cast<typename V::element_type>(std::get<Is>(t))... } };
00047 }

```

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

<i>T</i>	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

<i>T</i>	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]`

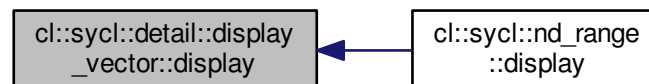
To debug and test.

Definition at line 163 of file [debug.hpp](#).

Referenced by [cl::sycl::nd_range< dims >::display\(\)](#).

```
00163     {
00164     #ifdef TRISYCL_DEBUG
00165         std::cout << boost::typeindex::type_id<T>().pretty_name() << ":";
00166     #endif
00167         // Get a pointer to the real object
00168         for (auto e : *static_cast<const T *>(this))
00169             std::cout << " " << e;
00170         std::cout << std::endl;
00171     }
```

Here is the caller graph for this function:



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.

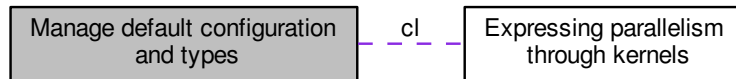
Definition at line 130 of file [debug.hpp](#).

References [TRISYCL_INTERNAL_DUMP](#).

```
00130                                     {
00131 #ifdef TRISYCL_TRACE_KERNEL
00132     // Inject tracing message around the kernel
00133     return [=] {
00134         /* Since the class KernelName may just be declared and not really
00135            defined, just use it through a class pointer to have
00136            typeid().name() not complaining */
00137         TRISYCL_INTERNAL_DUMP(
00138             "Kernel started "
00139             << boost::typeid::type_id<KernelName *>().pretty_name());
00140         f();
00141         TRISYCL_INTERNAL_DUMP(
00142             "Kernel stopped "
00143             << boost::typeid::type_id<KernelName *>().pretty_name());
00144     };
00145 #else
00146     // Identity by default
00147     return f;
00148 #endif
00149 }
```

8.6 Manage default configuration and types

Collaboration diagram for Manage default configuration and types:



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.

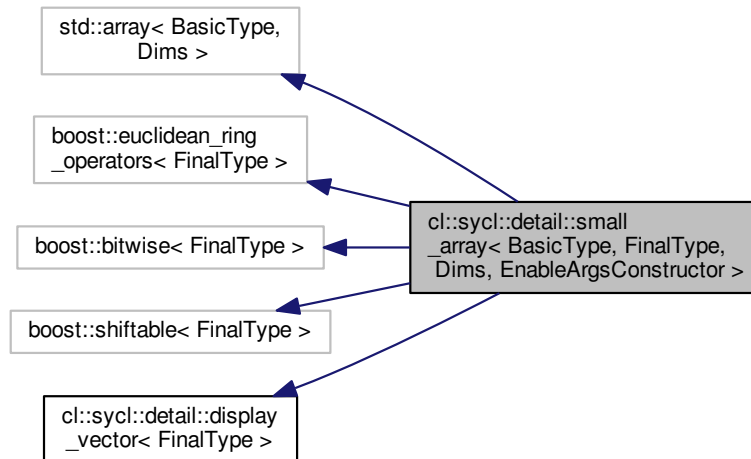
Parameters

<i>BasicType</i>	is the type element, such as <code>int</code>
<i>Dims</i>	is the dimension number, typically between 1 and 3
<i>FinalType</i>	is the final type, such as <code>range<></code> or <code>id<></code> , so that <code>boost::operator</code> can return the right type
<i>EnableArgsConstructor</i>	adds a constructors from Dims variadic elements when true. It is false by default.

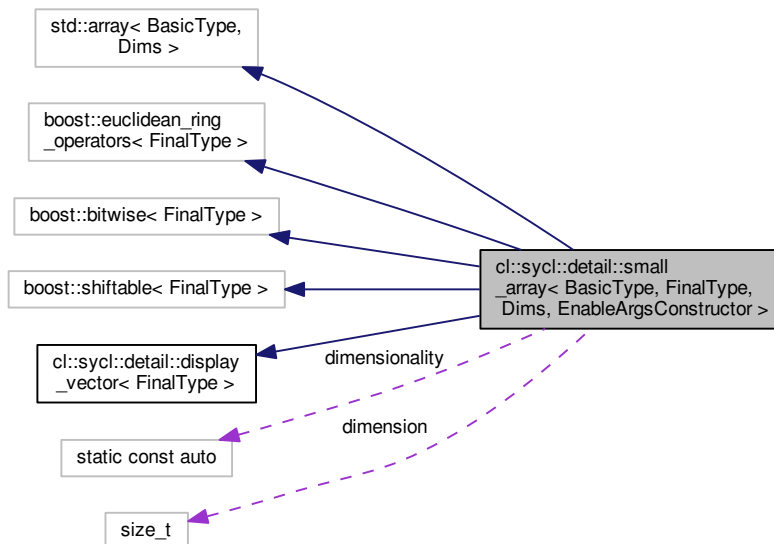
`std::array<>` provides the collection concept, with `.size()`, `==` and `!=` too.

Definition at line 65 of file [small_array.hpp](#).

Inheritance diagram for `cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >`:



Collaboration diagram for `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`.

```
00094                                     {
00095     // (*this)[0] is the first element of the underlying array
00096     std::copy_n(src, Dims, &(*this)[0]);
00097 }
```

```

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) {}

```

```

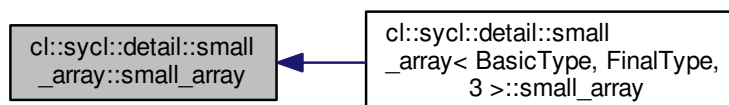
8.7.2.1.2.5 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false>
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::small_array ( )
[default]

```

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](#).

```
00152                                     {
00153     return (*this)[index];
00154 }
```

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](#).

```
00191                                     {
00192     return *static_cast<FinalType *>(this);
00193 }
```

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](#).

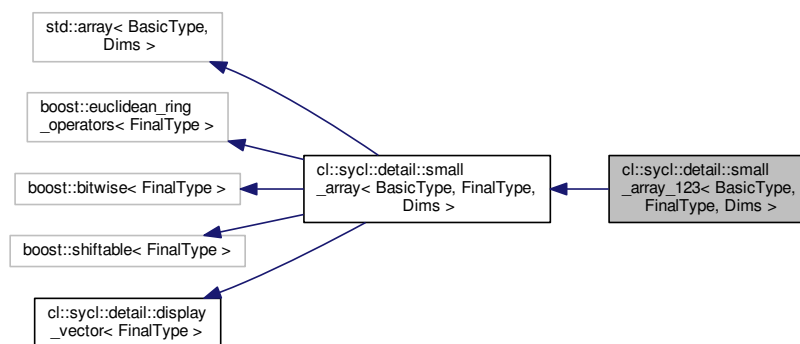
8.7.2.2 struct `cl::sycl::detail::small_array_123`

```
template<typename BasicType, typename FinalType, std::size_t Dims>
struct cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >
```

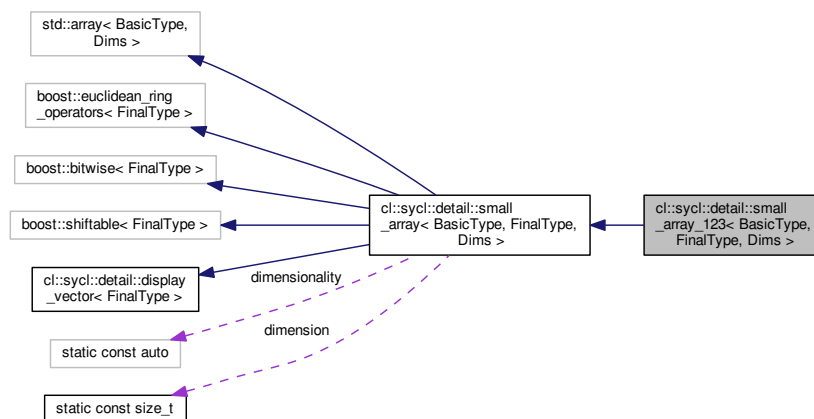
A small array of 1, 2 or 3 elements with the implicit constructors.

Definition at line 200 of file [small_array.hpp](#).

Inheritance diagram for `cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >`:



Collaboration diagram for `cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >`:



Additional Inherited Members

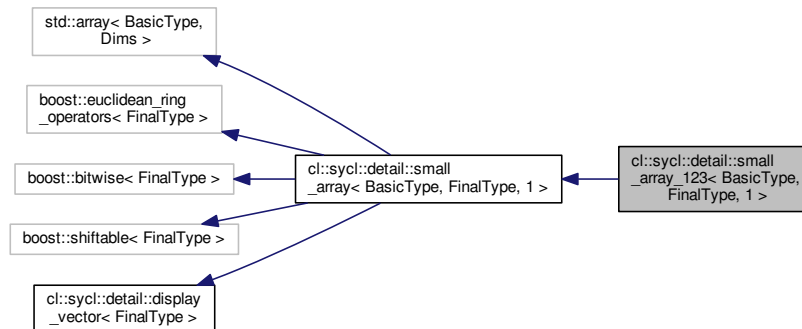
8.7.2.3 struct `cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >`

```
template<typename BasicType, typename FinalType>
struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >
```

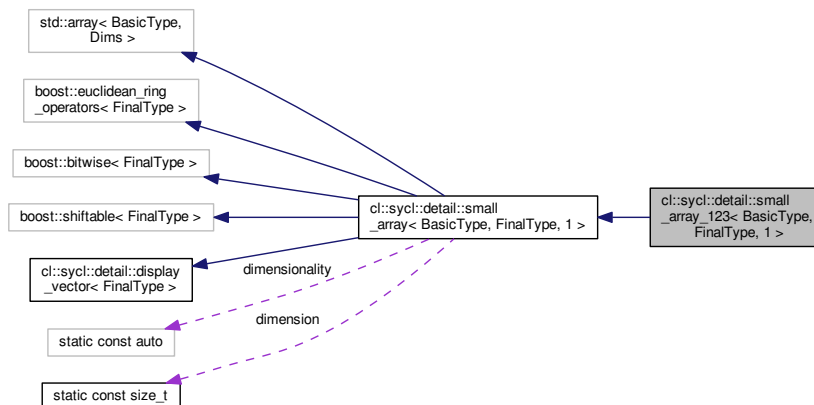
Use some specializations so that some function overloads can be determined according to some implicit constructors and to have an implicit conversion from/to BasicType (such as an int typically) if dims = 1.

Definition at line 212 of file [small_array.hpp](#).

Inheritance diagram for `cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >`:



Collaboration diagram for `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](#).

```
00216                                     {
00217     (*this)[0] = x;
00218 }
```

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](#).

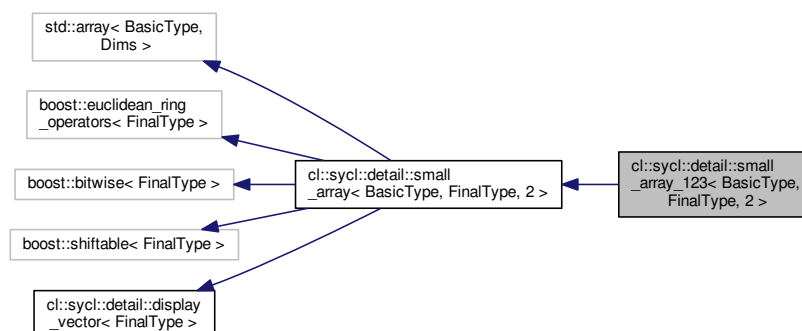
```
00228                                     {
00229     return (*this)[0];
00230 }
```

8.7.2.4 `struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >`

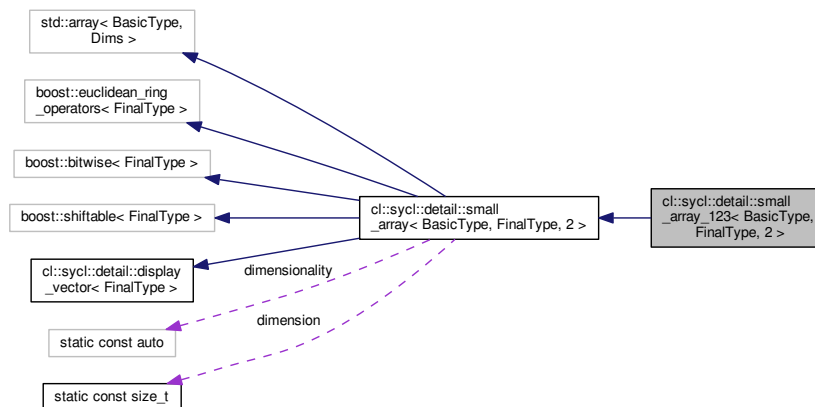
```
template<typename BasicType, typename FinalType>
struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >
```

Definition at line 235 of file [small_array.hpp](#).

Inheritance diagram for `cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >`:



Collaboration diagram for `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]
```

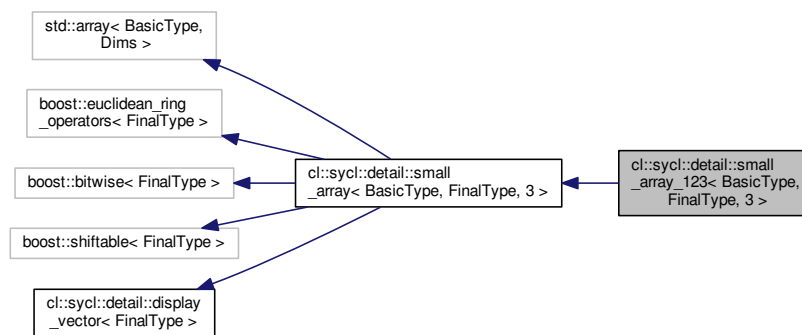
Keep other constructors.

```
8.7.2.5 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >
```

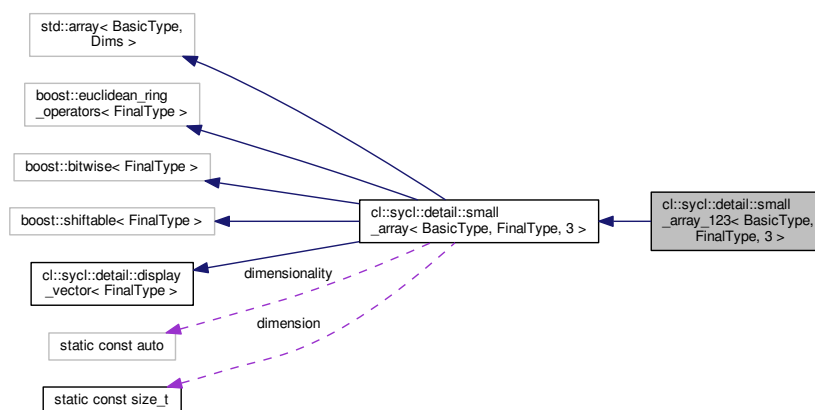
```
template<typename BasicType, typename FinalType>
struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >
```

Definition at line 261 of file [small_array.hpp](#).

Inheritance diagram for `cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >`:



Collaboration diagram for `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.

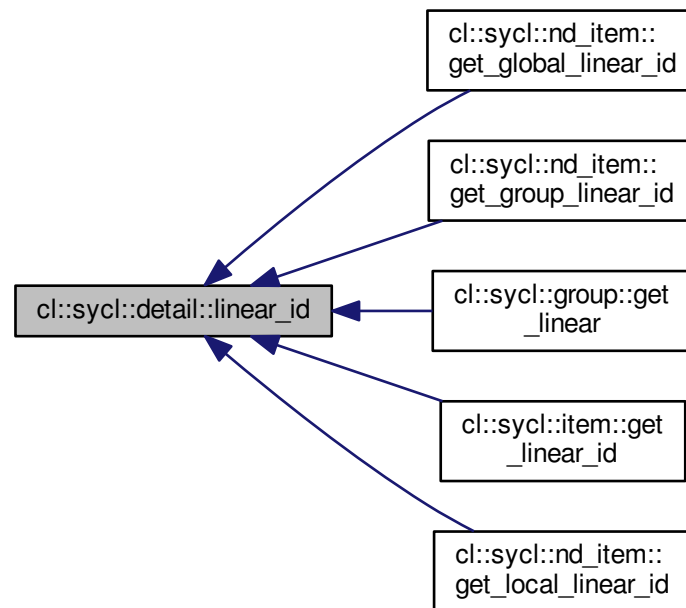
Typically for the [get_global_linear_id\(\)](#) and [get_local_linear_id\(\)](#) functions.

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\(\)](#).

```
00028                                     {} ) {
00029     auto dims = std::distance(std::begin(range), std::end(range));
00030
00031     size_t linear_id = 0;
00032     /* A good compiler should unroll this and do partial evaluation to
00033        remove the first multiplication by 0 of this Horner evaluation and
00034        remove the 0 offset evaluation */
00035     for (int i = dims - 1; i >= 0; --i)
00036         linear_id = linear_id*range[i] + id[i] - offset[i];
00037
00038     return linear_id;
00039 }
```

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.

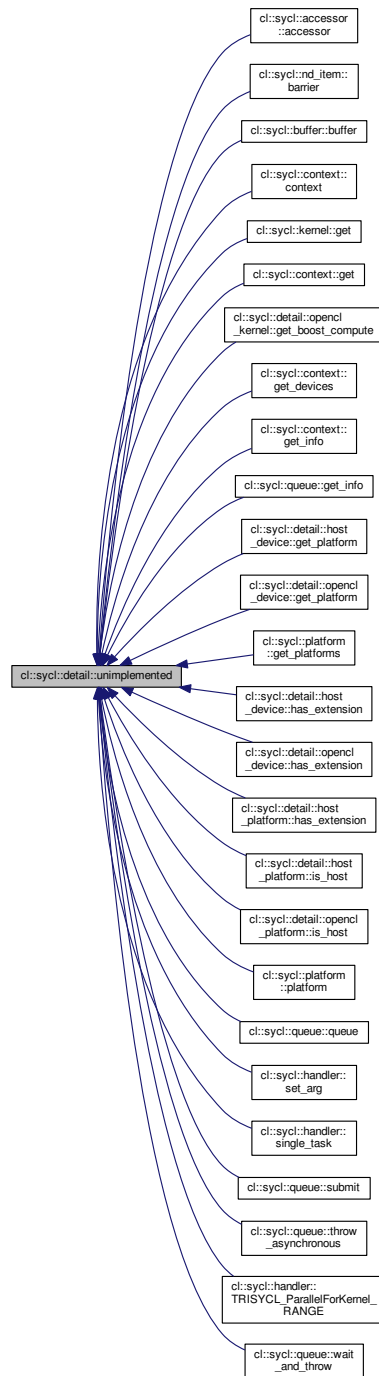
Definition at line 25 of file [unimplemented.hpp](#).

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::opengl_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::opengl_device::get_platform()`, `cl::sycl::platform::get_platforms()`, `cl::sycl::detail::host_device::has_extension()`, `cl::sycl::detail::opengl_device::has_extension()`, `cl::sycl::detail::host_platform::has_extension()`, `cl::sycl::detail::host_platform::is_host()`, `cl::sycl::detail::opengl_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()`.

```

00025         {
00026     std::cerr << "Error: using a non implemented feature!!!" << std::endl
00027               << "Please contribute to the open source implementation. :-)"
00028               << std::endl;
00029 }
  
```

Here is the caller graph for this function:



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 enqueueing 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

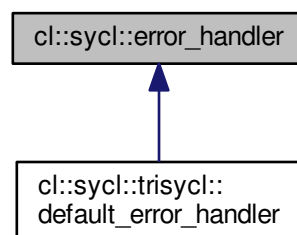
8.8.2 Class Documentation

8.8.2.1 struct `cl::sycl::error_handler`

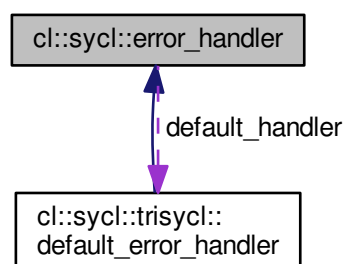
User supplied error handler to call a user-provided function when an error happens from a SYCL object that was constructed with this error handler.

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](#).

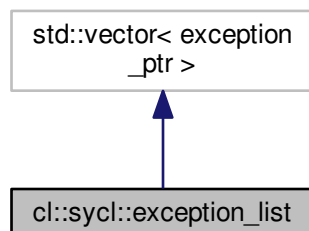
8.8.2.2 struct `cl::sycl::exception_list`

Exception list to store several exceptions.

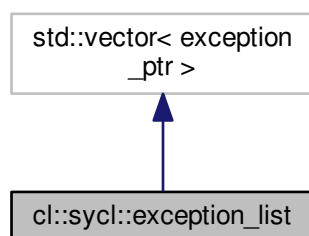
Todo Do we need to define it in SYCL or can we rely on plain C++17 one?

Definition at line 33 of file [exception.hpp](#).

Inheritance diagram for `cl::sycl::exception_list`:



Collaboration diagram for `cl::sycl::exception_list`:

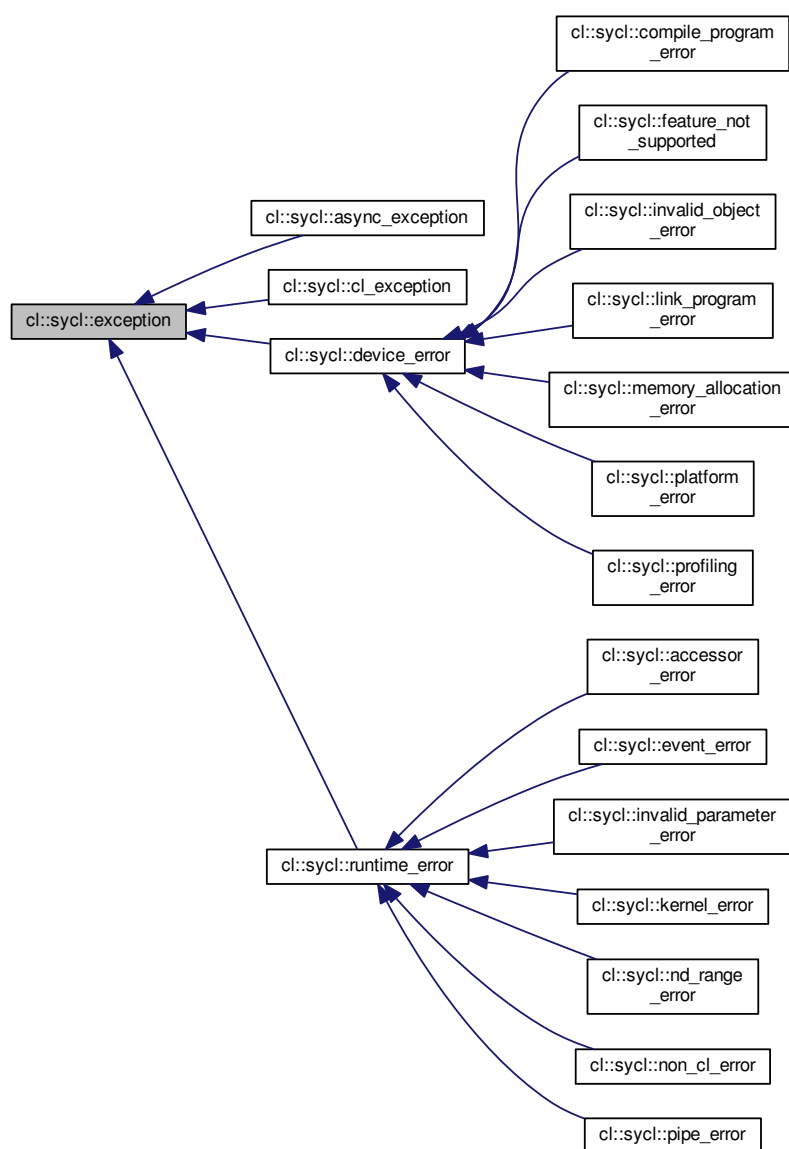


8.8.2.3 class `cl::sycl::exception`

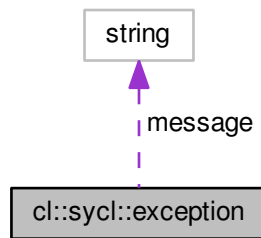
Encapsulate a SYCL error information.

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`.

```
00052                                     {
00053     return message;
00054 }
```

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.

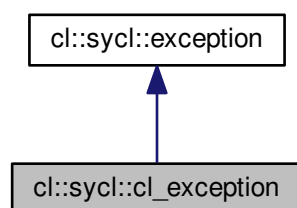
Definition at line 44 of file [exception.hpp](#).

8.8.2.4 `class cl::sycl::cl_exception`

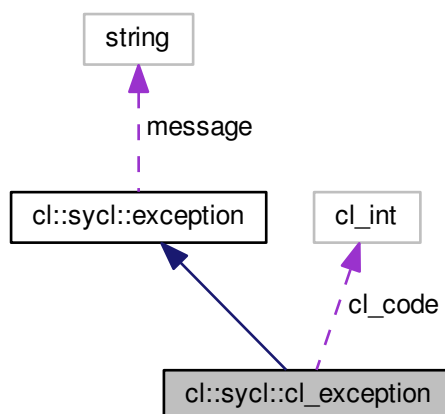
Returns the OpenCL error code encapsulated in the exception.

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.

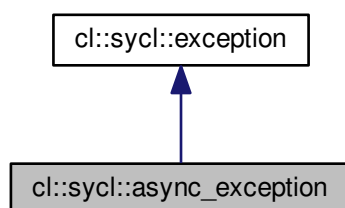
Definition at line 74 of file `exception.hpp`.

8.8.2.5 struct cl::sycl::async_exception

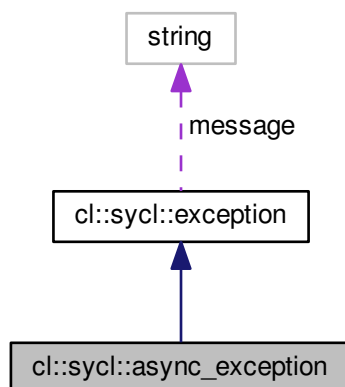
An error stored in an [exception_list](#) for asynchronous errors.

Definition at line 93 of file [exception.hpp](#).

Inheritance diagram for cl::sycl::async_exception:



Collaboration diagram for cl::sycl::async_exception:

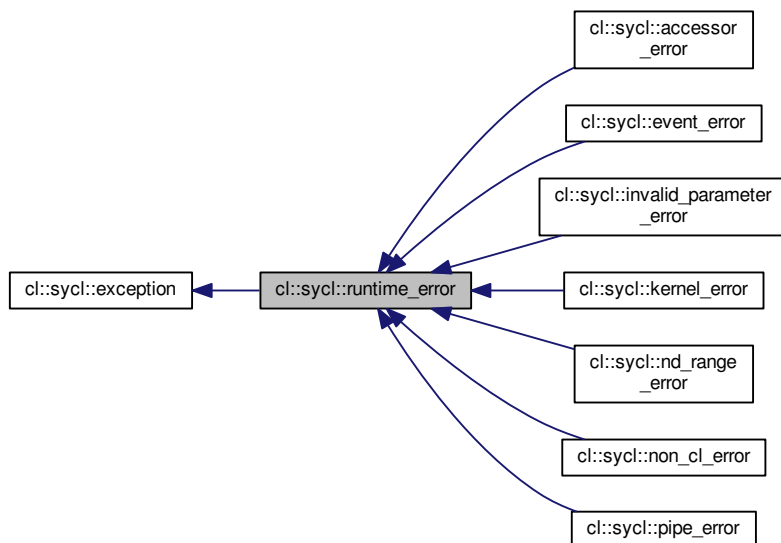


Additional Inherited Members

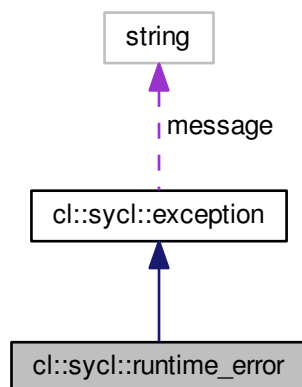
8.8.2.6 class cl::sycl::runtime_error

Definition at line 98 of file [exception.hpp](#).

Inheritance diagram for `cl::sycl::runtime_error`:



Collaboration diagram for `cl::sycl::runtime_error`:



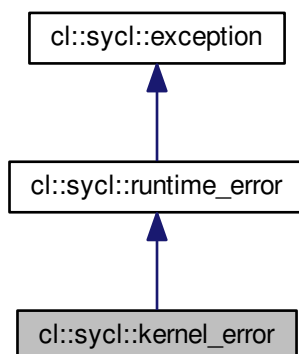
Additional Inherited Members

8.8.2.7 class `cl::sycl::kernel_error`

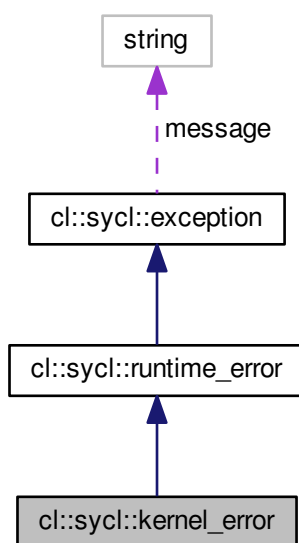
Error that occurred before or while enqueueing the SYCL kernel.

Definition at line 104 of file [exception.hpp](#).

Inheritance diagram for `cl::sycl::kernel_error`:



Collaboration diagram for `cl::sycl::kernel_error`:



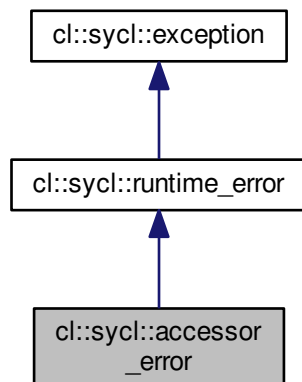
Additional Inherited Members

8.8.2.8 class `cl::sycl::accessor_error`

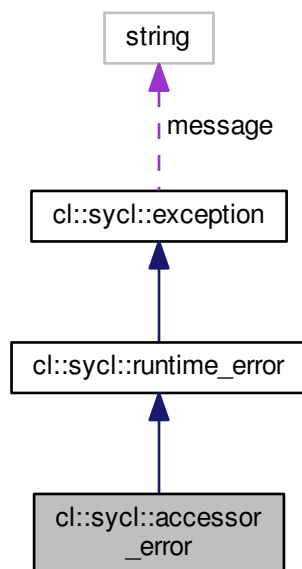
Error regarding the `cl::sycl::accessor` objects defined.

Definition at line 110 of file [exception.hpp](#).

Inheritance diagram for `cl::sycl::accessor_error`:



Collaboration diagram for `cl::sycl::accessor_error`:



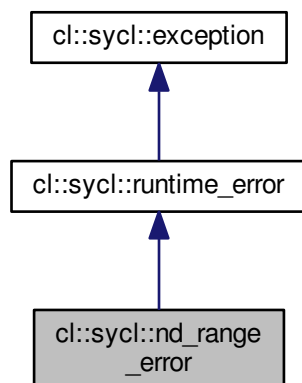
Additional Inherited Members

8.8.2.9 class `cl::sycl::nd_range_error`

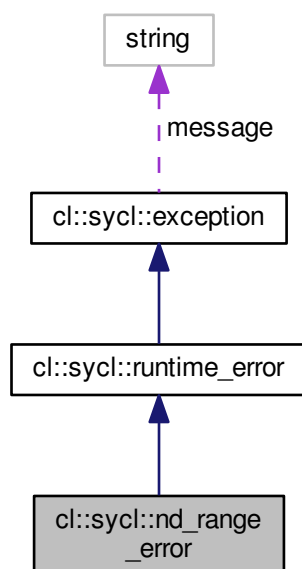
Error regarding the `cl::sycl::nd_range` specified for the SYCL kernel.

Definition at line 116 of file `exception.hpp`.

Inheritance diagram for `cl::sycl::nd_range_error`:



Collaboration diagram for `cl::sycl::nd_range_error`:



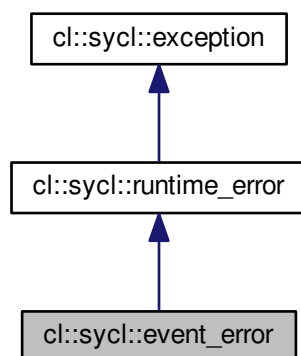
Additional Inherited Members

8.8.2.10 class `cl::sycl::event_error`

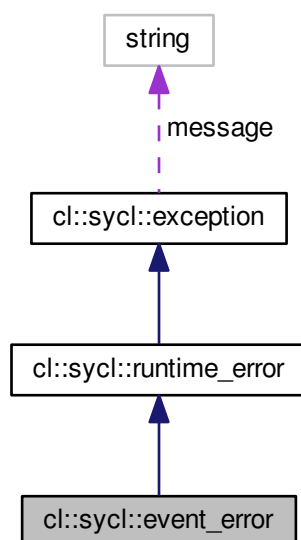
Error regarding associated `cl::sycl::event` objects.

Definition at line 122 of file `exception.hpp`.

Inheritance diagram for `cl::sycl::event_error`:



Collaboration diagram for `cl::sycl::event_error`:



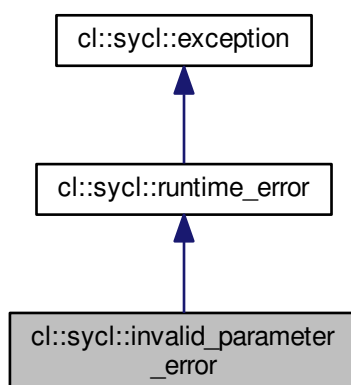
Additional Inherited Members

8.8.2.11 class `cl::sycl::invalid_parameter_error`

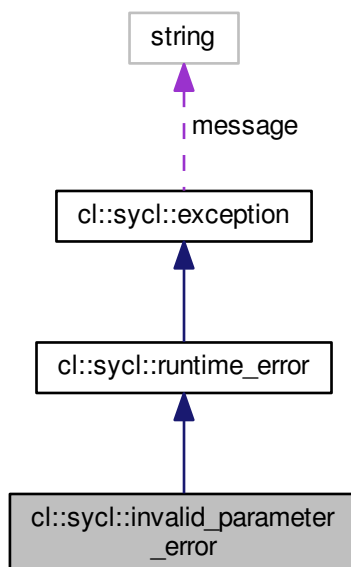
Error regarding parameters to the SYCL kernel, it may apply to any captured parameters to the kernel lambda.

Definition at line 130 of file [exception.hpp](#).

Inheritance diagram for `cl::sycl::invalid_parameter_error`:



Collaboration diagram for `cl::sycl::invalid_parameter_error`:



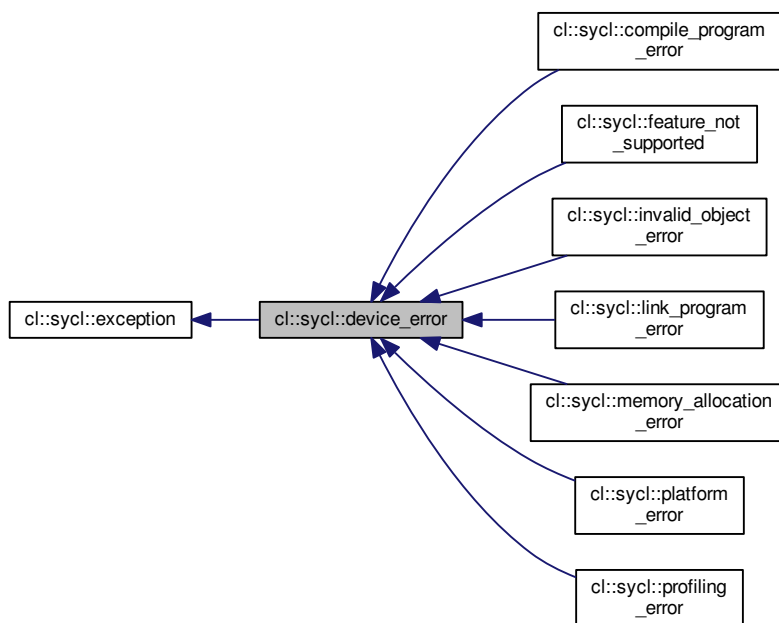
Additional Inherited Members

8.8.2.12 class `cl::sycl::device_error`

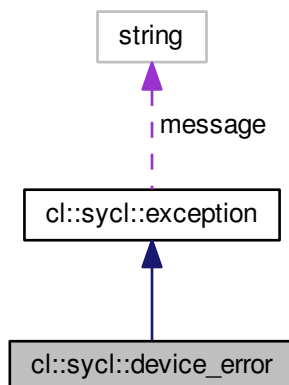
The SYCL device will trigger this exception on error.

Definition at line 136 of file [exception.hpp](#).

Inheritance diagram for `cl::sycl::device_error`:



Collaboration diagram for `cl::sycl::device_error`:



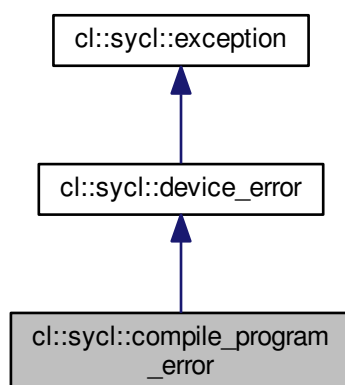
Additional Inherited Members

8.8.2.13 class `cl::sycl::compile_program_error`

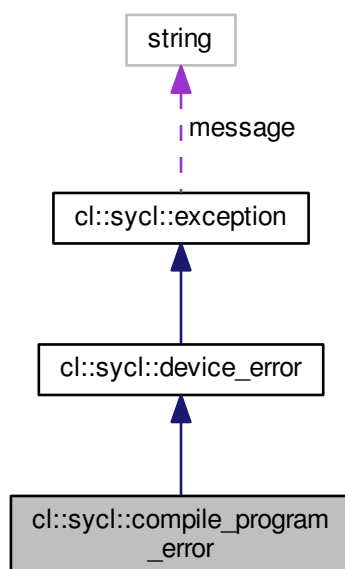
Error while compiling the SYCL kernel to a SYCL device.

Definition at line 142 of file [exception.hpp](#).

Inheritance diagram for `cl::sycl::compile_program_error`:



Collaboration diagram for `cl::sycl::compile_program_error`:



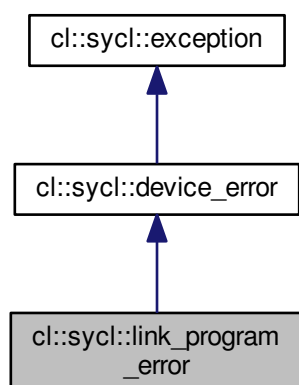
Additional Inherited Members

8.8.2.14 class `cl::sycl::link_program_error`

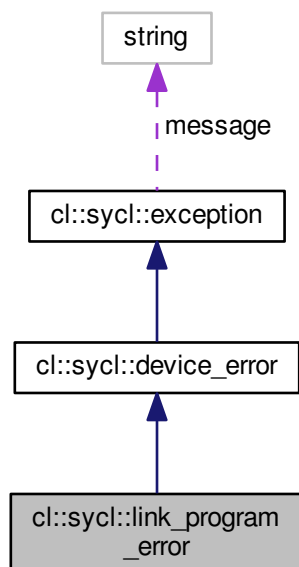
Error while linking the SYCL kernel to a SYCL device.

Definition at line 148 of file [exception.hpp](#).

Inheritance diagram for `cl::sycl::link_program_error`:



Collaboration diagram for `cl::sycl::link_program_error`:



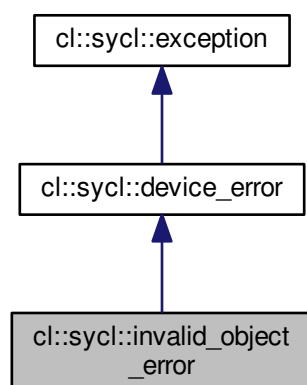
Additional Inherited Members

8.8.2.15 class `cl::sycl::invalid_object_error`

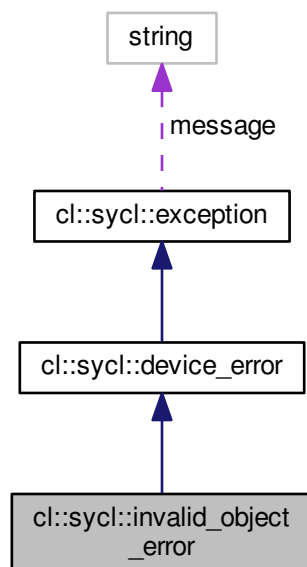
Error regarding any memory objects being used inside the kernel.

Definition at line 154 of file [exception.hpp](#).

Inheritance diagram for `cl::sycl::invalid_object_error`:



Collaboration diagram for `cl::sycl::invalid_object_error`:



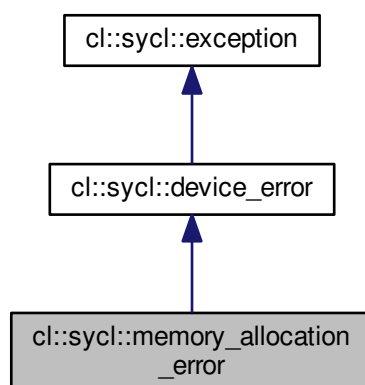
Additional Inherited Members

8.8.2.16 class `cl::sycl::memory_allocation_error`

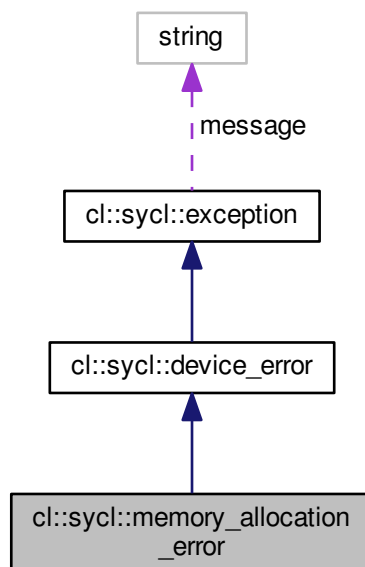
Error on memory allocation on the SYCL device for a SYCL kernel.

Definition at line 160 of file [exception.hpp](#).

Inheritance diagram for `cl::sycl::memory_allocation_error`:



Collaboration diagram for `cl::sycl::memory_allocation_error`:



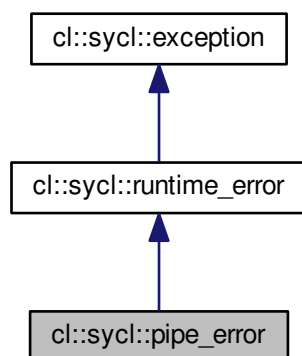
Additional Inherited Members

8.8.2.17 class `cl::sycl::pipe_error`

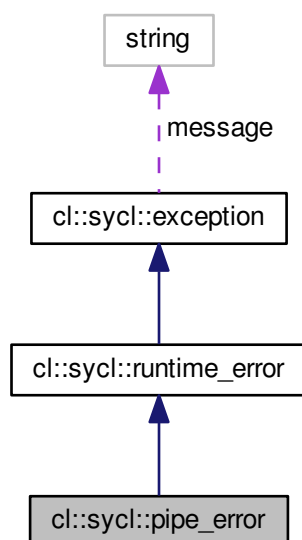
A failing pipe error will trigger this exception on error.

Definition at line 166 of file [exception.hpp](#).

Inheritance diagram for `cl::sycl::pipe_error`:



Collaboration diagram for `cl::sycl::pipe_error`:



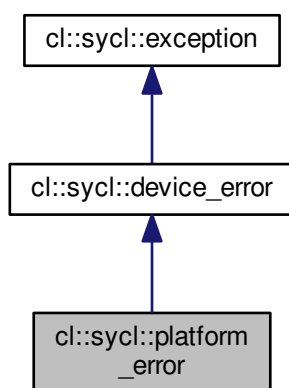
Additional Inherited Members

8.8.2.18 class `cl::sycl::platform_error`

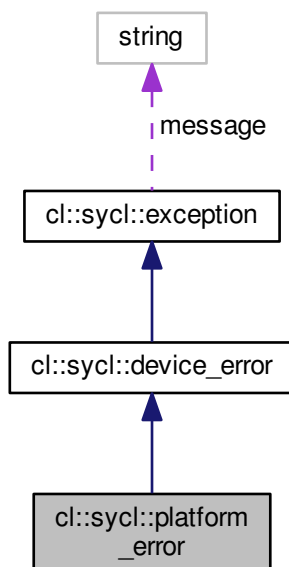
The SYCL platform will trigger this exception on error.

Definition at line 172 of file [exception.hpp](#).

Inheritance diagram for `cl::sycl::platform_error`:



Collaboration diagram for `cl::sycl::platform_error`:



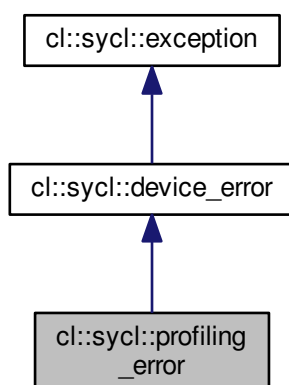
Additional Inherited Members

8.8.2.19 class `cl::sycl::profiling_error`

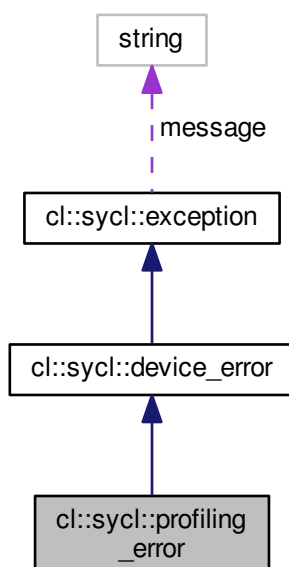
The SYCL runtime will trigger this error if there is an error when profiling info is enabled.

Definition at line 180 of file [exception.hpp](#).

Inheritance diagram for `cl::sycl::profiling_error`:



Collaboration diagram for `cl::sycl::profiling_error`:



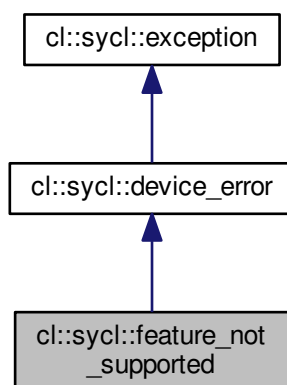
Additional Inherited Members

8.8.2.20 class `cl::sycl::feature_not_supported`

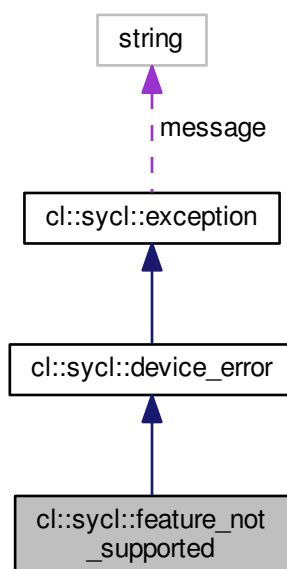
Exception thrown when an optional feature or extension is used in a kernel but its not available on the device the SYCL kernel is being enqueued on.

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.

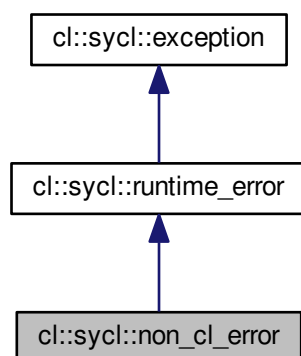
Todo Add to the specification

Todo Clean implementation

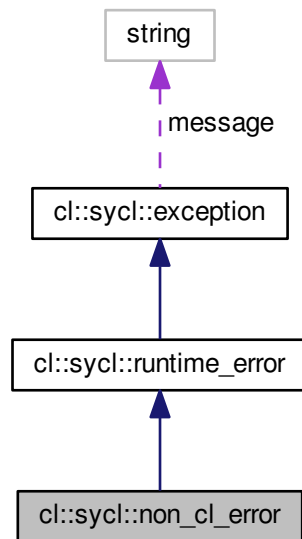
Todo Exceptions are named error in C++

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>`

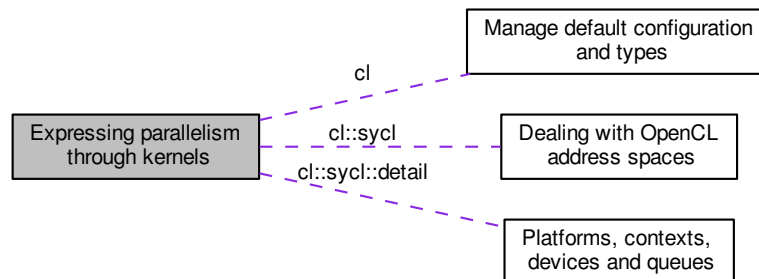
A shared pointer to an exception as in C++ specification.

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:



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

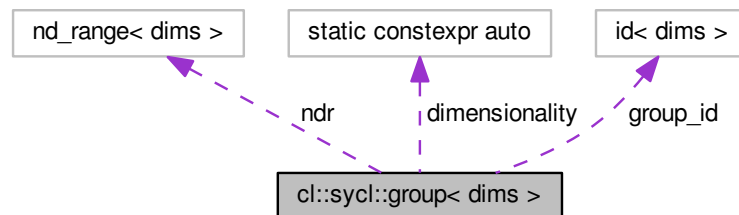
8.9.2.1 struct cl::sycl::group

```
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 constituent 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]`

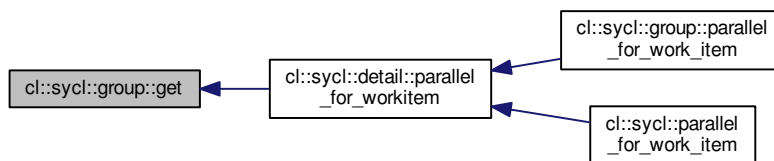
Return an id representing the index of the group within the `nd_range` for every dimension.

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:



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](#).

```
00126                                     {
00127     return get_global_range() [dimension];
00128 }
```

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](#).

```
00110                                     {
00111     return get_nd_range().get_group();
00112 }
```

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 constituent group range.

Definition at line 116 of file [group.hpp](#).

```
00116                                     {
00117     return get_group_range() [dimension];
00118 }
```

8.9.2.1.2.7 `template<std::size_t dims> size_t cl::sycl::group< dims >::get_linear () const` `[inline]`

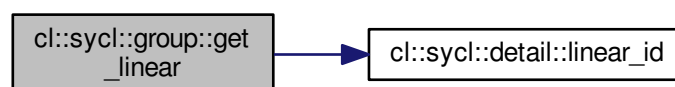
Get a linearized version of the group ID.

Definition at line 168 of file [group.hpp](#).

References [cl::sycl::detail::linear_id\(\)](#).

```
00168                                     {
00169     return detail::linear_id(get_group_range(), get());
00170 }
```

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.

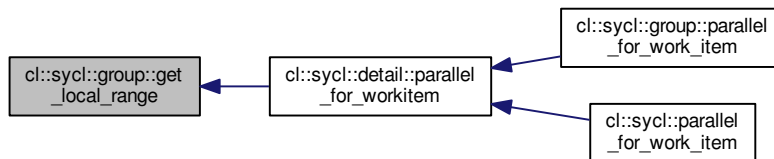
Todo Add to the specification

Definition at line 135 of file `group.hpp`.

Referenced by `cl::sycl::detail::parallel_for_workitem()`.

```
00135 { return get_nd_range().get_local(); }
```

Here is the caller graph for this function:



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`.

```
00142                                     {
00143     return get_local_range()[dimension];
00144 }
```

8.9.2.1.2.10 `template<std::size_t dims> nd_range<dims> cl::sycl::group< dims >::get_nd_range () const`
`[inline]`

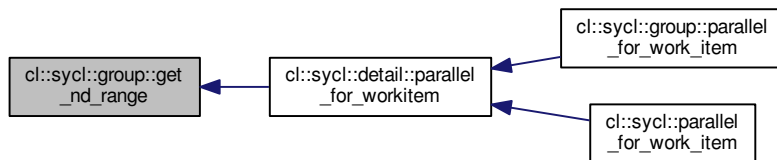
Todo Also provide this access to the current `nd_range`

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:



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

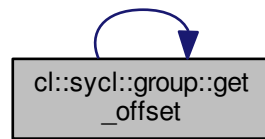
Definition at line 158 of file `group.hpp`.

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](#).

```

00097     {
00098         return group_id[dimension];
00099     }
  
```

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](#).

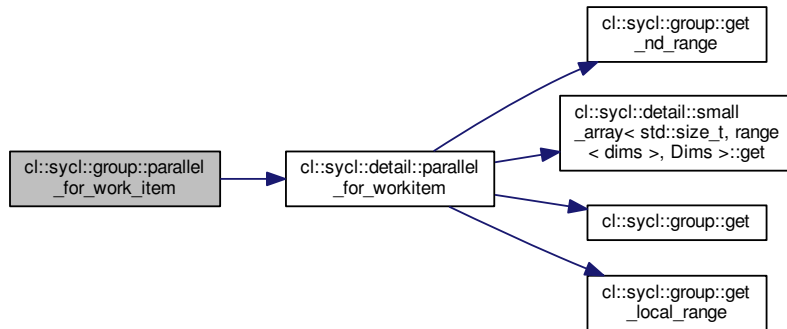
References [cl::sycl::detail::parallel_for_workitem\(\)](#).


```

00178     {
00179         detail::parallel_for_workitem(*this, f);
00180     }

```

Here is the call graph for this function:



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](#).

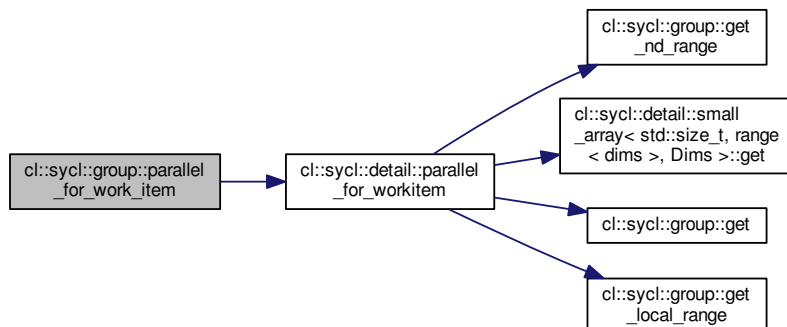
References [cl::sycl::detail::parallel_for_workitem\(\)](#).

```

00188     {
00189         auto item_adapter = [=] (nd_item<dimensionality> ndi) {
00190             item<dimensionality> i = ndi.get_item();
00191             f(i);
00192         };
00193         detail::parallel_for_workitem(*this, item_adapter);
00194     }

```

Here is the call graph for this function:



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](#).

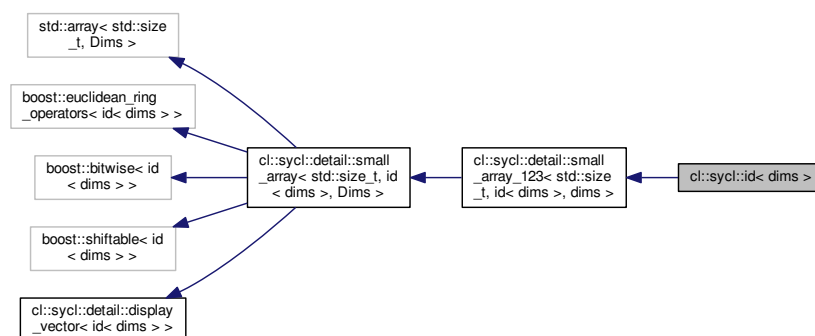
8.9.2.2 class `cl::sycl::id`

```
template<std::size_t dims = 1>
class cl::sycl::id< dims >
```

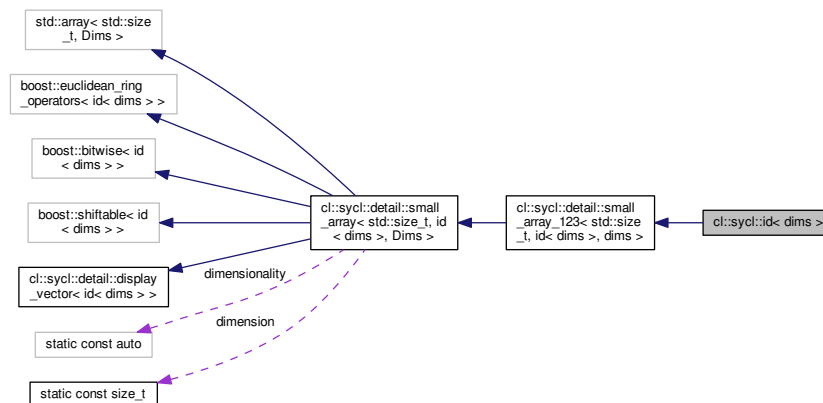
Define a multi-dimensional index, used for example to locate a work item.

Definition at line 31 of file [id.hpp](#).

Inheritance diagram for `cl::sycl::id< dims >`:



Collaboration diagram for `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:



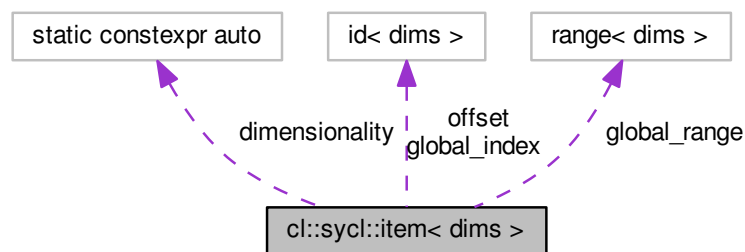
8.9.2.3 class `cl::sycl::item`

```
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\(\)](#).

```
00052         {} ) :
00053     global_range { global_size },
00054     global_index { global_index },
00055     offset { offset }
00056 {}
```

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]`

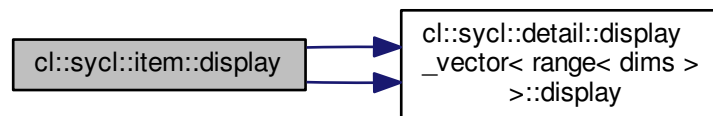
Display the value for debugging and validation purpose.

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\(\)](#).

```
00117         {
00118     global_range.display();
00119     global_index.display();
00120     offset.display();
00121 }
```

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.

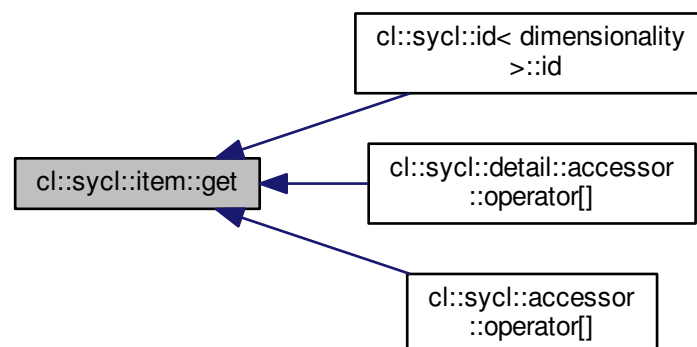
Definition at line 69 of file [item.hpp](#).

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\[\]\(\)](#).

```
00069 { return global_index; }
```

Here is the caller graph for this function:



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](#).

```
00075 { 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.

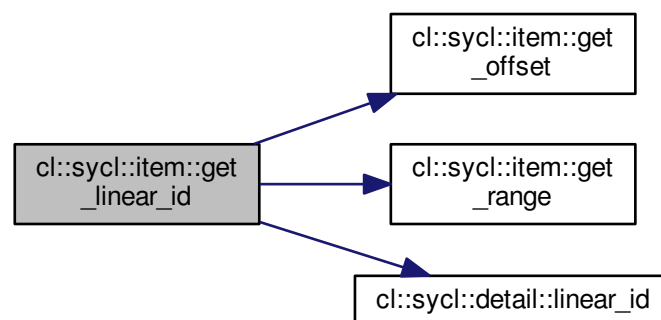
Computed as the flattened ID after the offset is subtracted.

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\(\)](#).

```
00104 {
00105     return detail::linear_id(get_range(), get(),
00106                             get_offset());
00106 }
```

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.

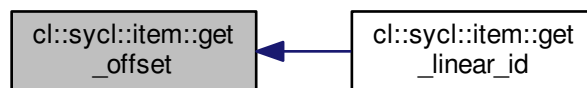
Definition at line 97 of file [item.hpp](#).

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.

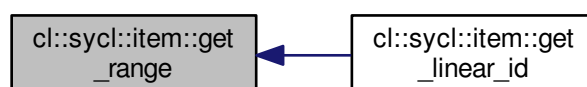
Definition at line 87 of file [item.hpp](#).

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\(\)](#).

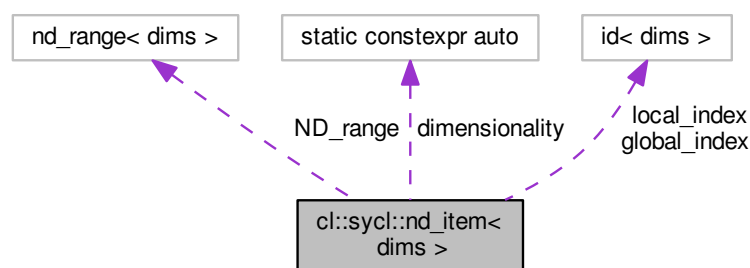
8.9.2.4 struct cl::sycl::nd_item

```
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](#).

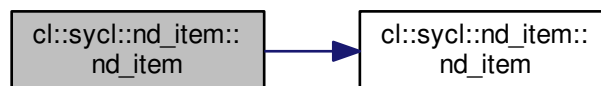
Todo This is for validation purpose. Hide this to the programmer somehow

Definition at line 62 of file [nd_item.hpp](#).

References [cl::sycl::nd_item< dims >::nd_item\(\)](#).

```
00063                                     :
00064     global_index { global_index },
00065     // Compute the local index using the offset and the group size
00066     local_index { (global_index - ndr.get_offset())%id<dims> { ndr.get_local() } },
00067     ND_range { ndr }
00068 {}
```

Here is the call graph for this function:



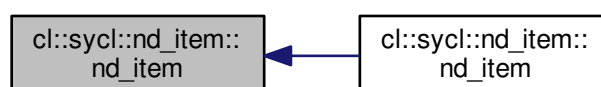
8.9.2.4.1.3 `template<std::size_t dims = 1> cl::sycl::nd_item< dims >::nd_item () [default]`

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.

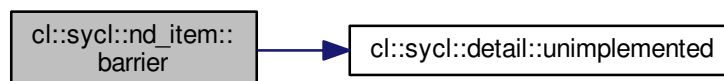
In addition, the barrier performs a fence operation ensuring that all memory accesses in the specified address space issued before the barrier complete before those issued after the barrier

Definition at line 198 of file `nd_item.hpp`.

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

```
00199                                     {
00200 #if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
00201     /* Use OpenMP barrier in the implementation with 1 OpenMP thread per
00202        work-item of the work-group */
00203 #pragma omp barrier
00204 #else
00205     // \todo To be implemented efficiently otherwise
00206     detail::unimplemented();
00207 #endif
00208 }
```

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.

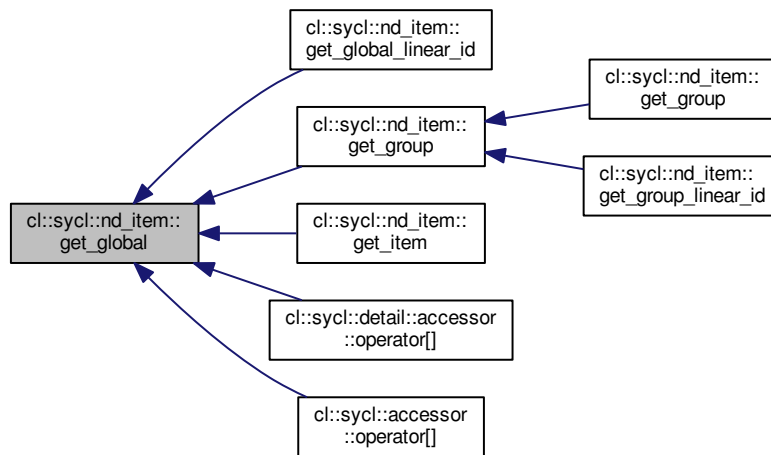
Definition at line 81 of file `nd_item.hpp`.

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[]()`.

```
00081 { 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.

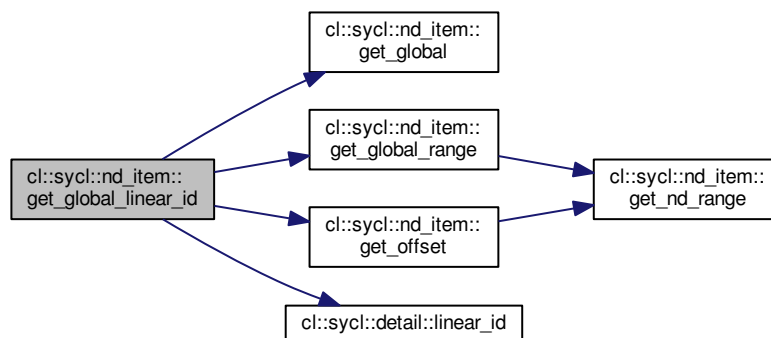
Definition at line 94 of file `nd_item.hpp`.

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()`.

```

00094         {
00095     return detail::linear_id(get_global_range(),
00096                             get_global(), get_offset());
00096     }
  
```

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<>`

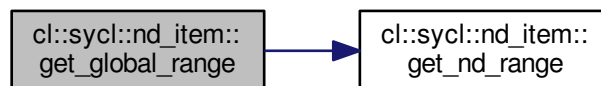
Definition at line 157 of file `nd_item.hpp`.

References `cl::sycl::nd_item< dims >::get_nd_range()`.

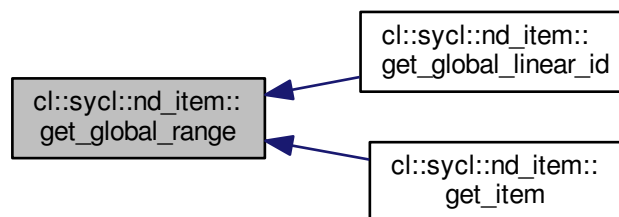
Referenced by `cl::sycl::nd_item< dims >::get_global_linear_id()`, and `cl::sycl::nd_item< dims >::get_item()`.

```
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()`.

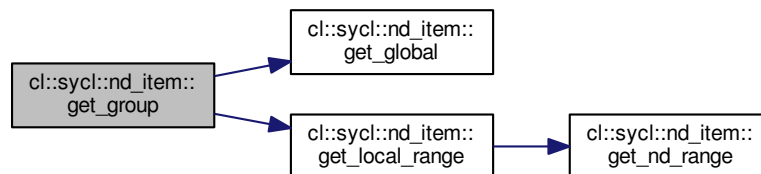
Referenced by `cl::sycl::nd_item< dims >::get_group()`, and `cl::sycl::nd_item< dims >::get_group_linear_id()`.


```

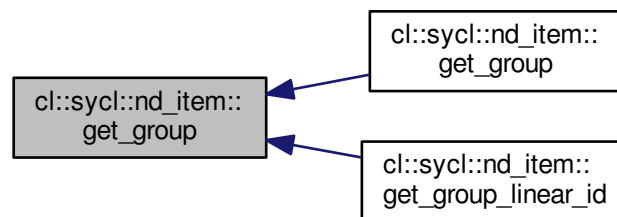
00123     {
00124     /* Convert get_local_range() to an id<> to remove ambiguity into using
00125        implicit conversion either from range<> to id<> or the opposite */
00126     return get_global()/id<dims> { get_local_range() };
00127     }

```

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`.

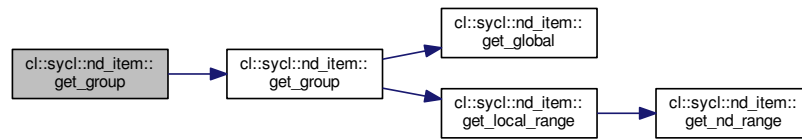
References `cl::sycl::nd_item< dims >::get_group()`.

```

00134     {
00135     return get_group()[dimension];
00136     }

```

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.

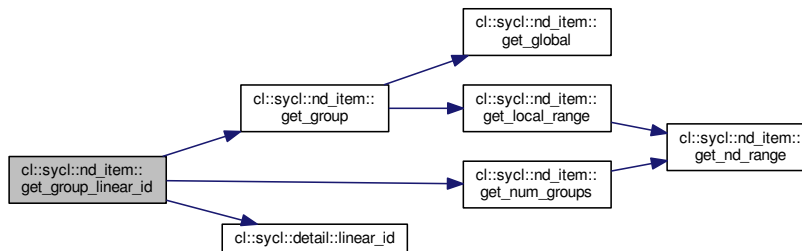
Definition at line 140 of file [nd_item.hpp](#).

References [cl::sycl::nd_item< dims >::get_group\(\)](#), [cl::sycl::nd_item< dims >::get_num_groups\(\)](#), and [cl::sycl::detail::linear_id\(\)](#).

```

00140         {
00141     return detail::linear_id(get_num_groups(),
00142                             get_group());
00142     }
  
```

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](#).

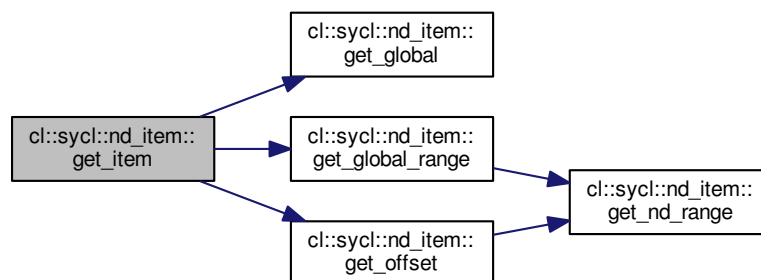
References [cl::sycl::nd_item< dims >::get_global\(\)](#), [cl::sycl::nd_item< dims >::get_global_range\(\)](#), and [cl::sycl::nd_item< dims >::get_offset\(\)](#).

```

00183     {
00184     return { get_global_range(), get_global(),
00185            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()`.

```

00102 { 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.

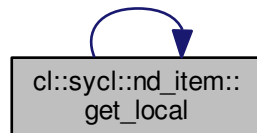
Definition at line 109 of file [nd_item.hpp](#).

References [cl::sycl::nd_item< dims >::get_local\(\)](#).

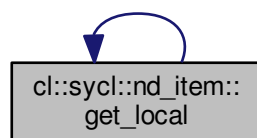
Referenced by [cl::sycl::nd_item< dims >::get_local\(\)](#).

```
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](#).

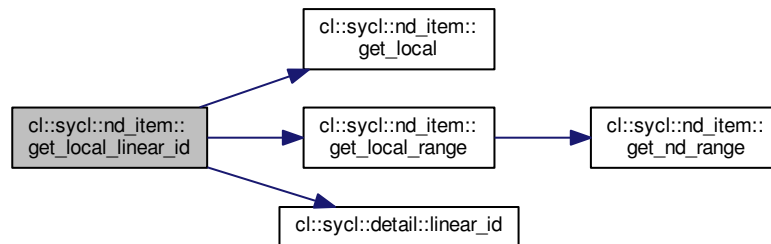
References [cl::sycl::nd_item< dims >::get_local\(\)](#), [cl::sycl::nd_item< dims >::get_local_range\(\)](#), and [cl::sycl::detail::linear_id\(\)](#).

```

00115     {
00116     return detail::linear_id(get_local_range(),
00117     get_local());
00117     }

```

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()`.

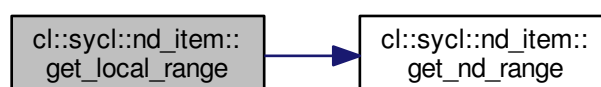
Referenced by `cl::sycl::nd_item< dims >::get_group()`, and `cl::sycl::nd_item< dims >::get_local_linear_id()`.

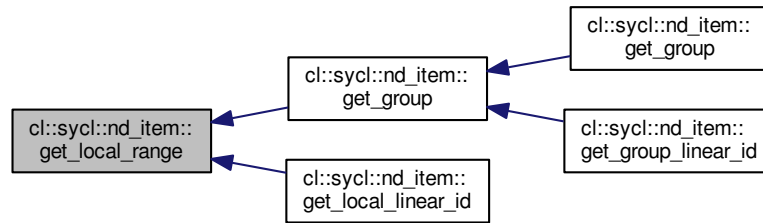
```

00163     {
00164     return get_nd_range().get_local();
00165     }

```

Here is the call graph for this function:

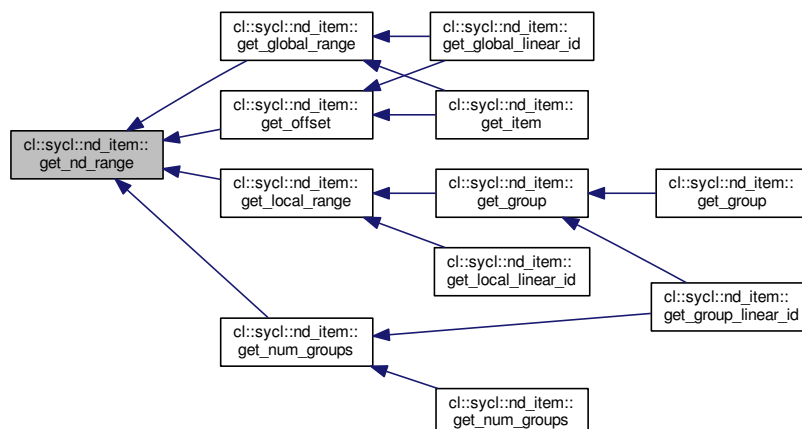




Return the `nd_range<>` of the current execution.

References [cl::sycl::nd_item< dims >::ND_range](#).

```
00176 { return ND_range; }
```



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](#).

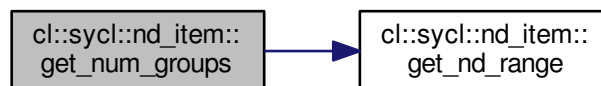
Definition at line 146 of file [nd_item.hpp](#).

References [cl::sycl::nd_item< dims >::get_nd_range\(\)](#).

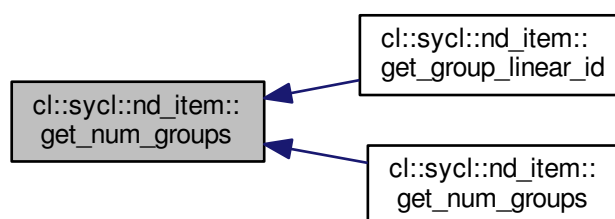
Referenced by [cl::sycl::nd_item< dims >::get_group_linear_id\(\)](#), and [cl::sycl::nd_item< dims >::get_num_↵groups\(\)](#).

```
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](#).

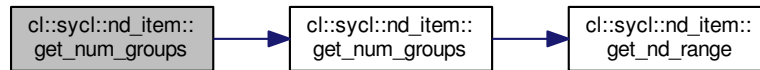
References [cl::sycl::nd_item< dims >::get_num_groups\(\)](#).

```

00151                                     {
00152     return get_num_groups() [dimension];
00153 }

```

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()`.

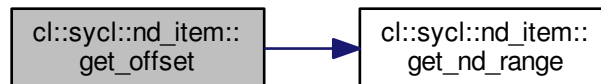
Referenced by `cl::sycl::nd_item< dims >::get_global_linear_id()`, and `cl::sycl::nd_item< dims >::get_item()`.

```

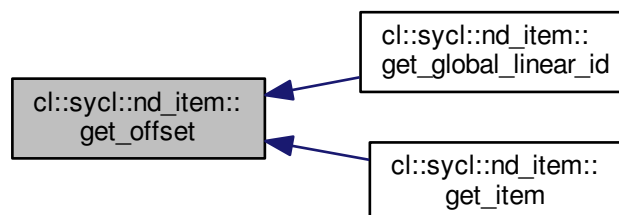
00172 { return get_nd_range().get_offset(); }

```

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\(\)](#).

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](#).

```
00052         {} ) :
00053     global_range { global_size }, local_range { local_size },
00054     offset { offset }
00055     { }
```

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]`

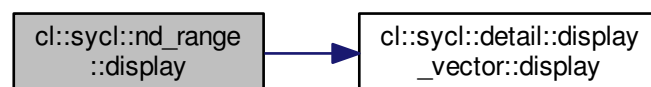
Display the value for debugging and validation purpose.

Definition at line 80 of file [nd_range.hpp](#).

References [cl::sycl::detail::display_vector< T >::display\(\)](#).

```
00080     {
00081     global_range.display();
00082     local_range.display();
00083     offset.display();
00084 }
```

Here is the call graph for this function:



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](#).

```
00058 { 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]`

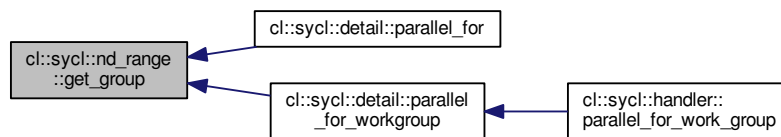
Get the range of work-groups needed to run this ND-range.

Definition at line 66 of file [nd_range.hpp](#).

Referenced by [cl::sycl::detail::parallel_for\(\)](#), and [cl::sycl::detail::parallel_for_workgroup\(\)](#).

```
00066 {
00067     /* This is basically global_range/local_range, round up to the
00068        next integer, in case the global eange is not a multiple of the
00069        local range. Note this is a motivating example to build a range
00070        from a scalar with a broadcasting constructor. */
00071     return (global_range + local_range - range<dims>{ 1 })/
00072         local_range;
00072 }
```

Here is the caller graph for this function:



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.

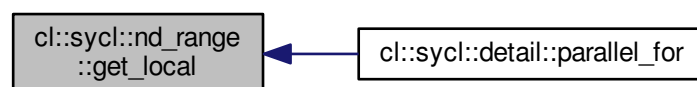
Definition at line 62 of file [nd_range.hpp](#).

References [cl::sycl::nd_range< dims >::local_range](#).

Referenced by [cl::sycl::detail::parallel_for\(\)](#).

```
00062 { return local_range; }
```

Here is the caller graph for this function:



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](#).

```
00048                                     {
00049     for (boost::multi_array_types::index _sycl_index = 0,
00050          _sycl_end = r[Range::dimensionality - level];
00051          _sycl_index < _sycl_end;
00052          _sycl_index++) {
00053         // Set the current value of the index for this dimension
00054         index[Range::dimensionality - level] = _sycl_index;
00055         // Iterate further on lower dimensions
00056         parallel_for_iterate<level - 1,
00057                               Range,
00058                               ParallelForFunctor,
00059                               Id> { r, f, index };
00060     }
00061 }
```

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](#).

```

00075                                     {
00076     // Create the OpenMP threads before the for-loop to avoid creating an
00077     // index in each iteration
00078     #pragma omp parallel
00079     {
00080         // Allocate an OpenMP thread-local index
00081         Id index;
00082         // Make a simple loop end condition for OpenMP
00083         boost::multi_array_types::index _sycl_end =
00084             r[Range::dimensionality - level];
00085         /* Distribute the iterations on the OpenMP threads. Some OpenMP
00086            "collapse" could be useful for small iteration space, but it
00087            would need some template specialization to have real contiguous
00088            loop nests */
00089         #pragma omp for
00090         for (boost::multi_array_types::index _sycl_index = 0;
00091              _sycl_index < _sycl_end;
00092              _sycl_index++) {
00093             // Set the current value of the index for this dimension
00094             index[Range::dimensionality - level] = _sycl_index;
00095             // Iterate further on lower dimensions
00096             parallel_for_iterate<level - 1,
00097                                   Range,
00098                                   ParallelForFuncor,
00099                                   Id> { r, f, index };
00100         }
00101     }
00102 }

```

8.9.2.8 struct `cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFuncor, Id >`

```

template<typename Range, typename ParallelForFuncor, typename Id>
struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFuncor, 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, ParallelForFuncor &f, Id &index)

8.9.2.8.1 Constructor & Destructor Documentation

```

8.9.2.8.1.1 template<typename Range , typename ParallelForFuncor , typename Id > cl::sycl::detail::parallel_for_↵
iterate< 0, Range, ParallelForFuncor, Id >::parallel_for_iterate ( Range r, ParallelForFuncor & f, Id & index )
[inline]

```

Definition at line 110 of file [parallelism.hpp](#).

```

00110                                     {
00111     f(index);
00112 }

```

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.

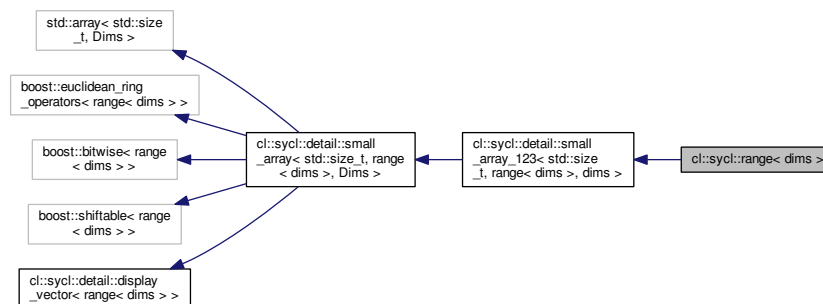
Todo use `std::size_t` `dims` instead of `int` `dims` in the specification?

Todo add to the specification this default parameter value?

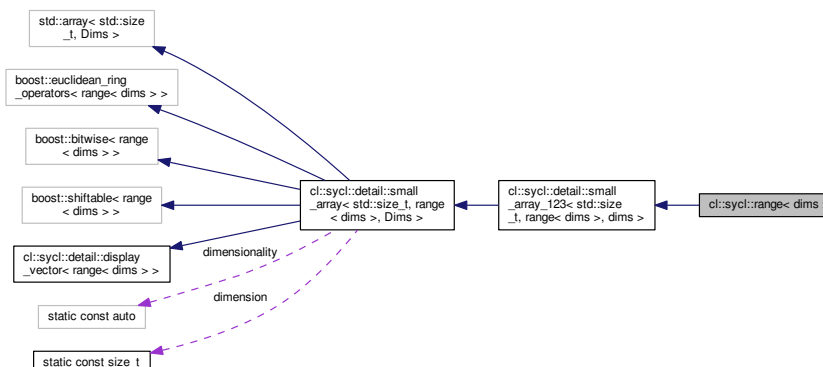
Todo add to the specification some way to specify an offset?

Definition at line 33 of file [range.hpp](#).

Inheritance diagram for `cl::sycl::range< dims >`:



Collaboration diagram for `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](#).

```
00049         {
00050         // Return the product of the sizes in each dimension
00051         return std::accumulate(this->cbegin(),
00052                                this->cend(),
00053                                1,
00054                                std::multiplies<size_t> {});
00055     }
```

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](#).

```
00066 { 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](#).

```
00067 { 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](#).

```
00074                                     {
00075     // Call constructor directly to allow narrowing
00076     return id<sizeof...(Args)>(Args...);
00077 }
```

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](#).

```
00074 {
00075     // Call constructor directly to allow narrowing
00076     return range<sizeof...(Args)>(Args...);
00077 }
```

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 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.

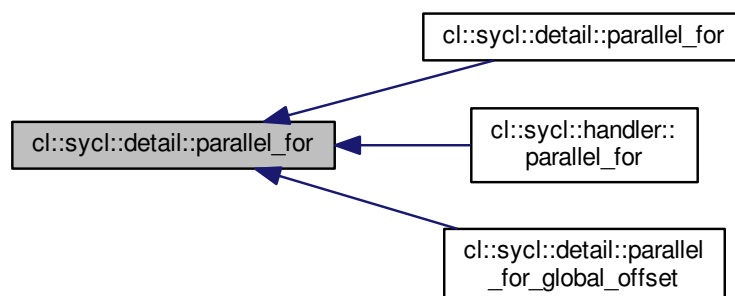
This implementation use OpenMP 3 if compiled with the right flag.

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\(\)](#).

```
00124 {
00125     #ifdef _OPENMP
00126         // Use OpenMP for the top loop level
00127         parallel_OpenMP_for_iterate<Dimensions,
00128                                     range<Dimensions>,
00129                                     ParallelForFunctor,
00130                                     id<Dimensions>> { r, f };
00131     #else
00132         // In a sequential execution there is only one index processed at a time
00133         id<Dimensions> index;
00134         parallel_for_iterate<Dimensions,
00135                             range<Dimensions>,
00136                             ParallelForFunctor,
00137                             id<Dimensions>> { r, f, index };
00138     #endif
00139 }
```

Here is the caller graph for this function:



8.9.3.10 `template<std::size_t Dimensions = 1, typename ParallelForFuncor > void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFuncor 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 ParallelForFuncor > void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFuncor 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(&ParallelForFuncor::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:



8.9.3.12 `template<std::size_t Dimensions = 1, typename ParallelForFuncor > void cl::sycl::detail::parallel_for (nd_range< Dimensions > r, ParallelForFuncor 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

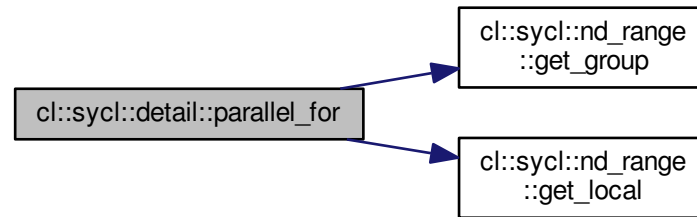
Todo Implement with `parallel_for_workgroup()/parallel_for_workitem()`

Definition at line 214 of file `parallelism.hpp`.

References `cl::sycl::nd_range< dims >::get_group()`, and `cl::sycl::nd_range< dims >::get_local()`.

```
00215                                     {
00216     // In a sequential execution there is only one index processed at a time
00217     nd_item<Dimensions> index { r };
00218     // To iterate on the work-group
00219     id<Dimensions> group;
00220     range<Dimensions> group_range = r.get_group();
00221     // To iterate on the local work-item
00222     id<Dimensions> local;
00223
00224     range<Dimensions> local_range = r.get_local();
00225
00226     // Reconstruct the nd_item from its group and local id
00227     auto reconstruct_item = [&] (id<Dimensions> l) {
00228         //local.display();
00229         // Reconstruct the global nd_item
00230         index.set_local(local);
00231         // Upgrade local_range to an id<> so that we can * with the group (an id<>)
00232         index.set_global(local + id<Dimensions>(local_range)*group);
00233         // Call the user kernel at last
00234         f(index);
00235     };
00236
00237     /* To recycle the parallel_for on range<>, wrap the ParallelForFuncor f
00238     into another functor that iterates inside the work-group and then
00239     calls f */
00240     auto iterate_in_work_group = [&] (id<Dimensions> g) {
00241         //group.display();
00242         // Then iterate on the local work-groups
00243         parallel_for_iterate<Dimensions,
00244             range<Dimensions>,
00245             decltype(reconstruct_item),
00246             id<Dimensions>>> { local_range,
00247                 reconstruct_item,
00248                 local };
00249     };
00250
00251     // First iterate on all the work-groups
00252     parallel_for_iterate<Dimensions,
00253         range<Dimensions>,
00254         decltype(iterate_in_work_group),
00255         id<Dimensions>>> { group_range,
00256             iterate_in_work_group,
00257             group };
00258 }
```

Here is the call graph for this function:



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.

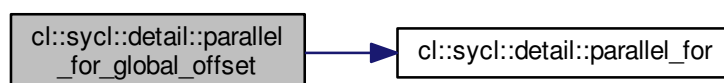
Definition at line 188 of file `parallelism.hpp`.

References `cl::sycl::detail::parallel_for()`.

```

00190
00191 // Reconstruct the item from its id<> and its offset
00192 auto reconstruct_item = [&] (id<Dimensions> l) {
00193     // Reconstruct the global item
00194     item<Dimensions> index { global_size, l + offset, offset };
00195     // Call the user kernel with the item<> instead of the id<>
00196     f(index);
00197 };
00198
00199 // First iterate on all the work-groups
00200 parallel_for(global_size, reconstruct_item);
00201 }
  
```

Here is the call graph for this function:



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

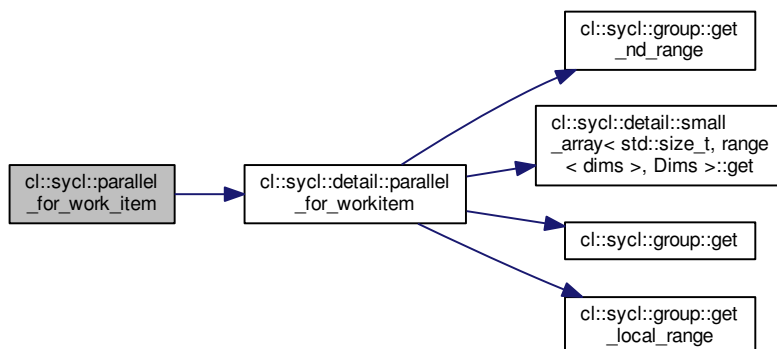
Todo Deprecate this function in the specification to use instead the group method

Definition at line 38 of file `parallelism.hpp`.

References `cl::sycl::detail::parallel_for_workitem()`.

```
00039                                     {
00040     detail::parallel_for_workitem(g, f);
00041 }
```

Here is the call graph for this function:



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()`.

```

00264                                     {
00265 // In a sequential execution there is only one index processed at a time
00266 group<Dimensions> g { r };
00267
00268 // First iterate on all the work-groups
00269 parallel_for_iterate<Dimensions,
00270                      range<Dimensions>,
00271                      ParallelForFunctor,
00272                      group<Dimensions>> {
00273     r.get_group(),
00274     f,
00275     g };
00276 }

```

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()`.

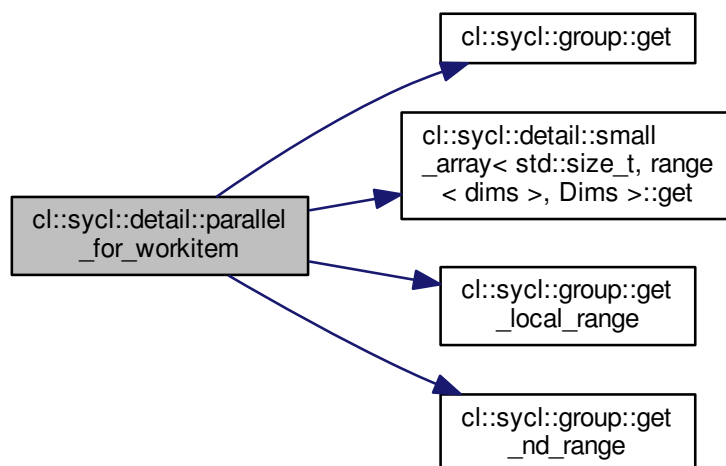
Referenced by `cl::sycl::parallel_for_work_item()`, and `cl::sycl::group< dims >::parallel_for_work_item()`.


```

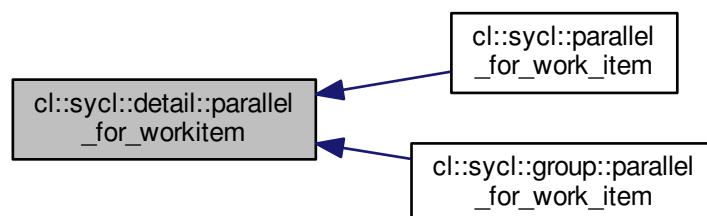
00285                                     {
00286 #if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
00287     /* To implement barriers With OpenMP, one thread is created for each
00288        work-item in the group and thus an OpenMP barrier has the same effect
00289        of an OpenCL barrier executed by the work-items in a workgroup
00290
00291        The issue is that the parallel_for_workitem() execution is slow even
00292        when nd_item::barrier() is not used
00293     */
00294
00295
00296     // Is the above comment true anymore ?
00297     // Maybe the following will be enough
00298     // #ifdef _OPENMP
00299
00300     // With OMP, one task is created for each work-item in the group
00301
00302     range<Dimensions> l_r = g.get_nd_range().get_local();
00303     int tot = l_r.get(0);
00304     for (int i = 1; i < (int) Dimensions; ++i){
00305         tot *= l_r.get(i);
00306     }
00307 #pragma omp parallel
00308 {
00309     #pragma omp single nowait
00310     {
00311         for (int th_id = 0; th_id < tot; ++th_id) {
00312 #pragma omp task firstprivate(th_id)
00313         {
00314             nd_item<Dimensions> index { g.get_nd_range() };
00315             id<Dimensions> local; // to initialize correctly
00316
00317             if (Dimensions == 1) {
00318                 local[0] = th_id;
00319             } else if (Dimensions == 2) {
00320                 local[0] = th_id / l_r.get(1);
00321                 local[1] = th_id - local[0]*l_r.get(1);
00322             } else if (Dimensions == 3) {
00323                 int tmp = l_r.get(1)*l_r.get(2);
00324                 local[0] = th_id / tmp;
00325                 local[1] = (th_id - local[0]*tmp) / l_r.get(1);
00326                 local[2] = th_id - local[0]*tmp - local[1]*l_r.get(1);
00327             }
00328             index.set_local(local);
00329             index.set_global(local + id<Dimensions>(l_r)*g.get());
00330             f(index);
00331         }
00332     }
00333 }
00334 }
00335 #else
00336 // In a sequential execution there is only one index processed at a time
00337 nd_item<Dimensions> index { g.get_nd_range() };
00338 // To iterate on the local work-item
00339 id<Dimensions> local;
00340
00341 // Reconstruct the nd_item from its group and local id
00342 auto reconstruct_item = [&] (id<Dimensions> l) {
00343     //local.display();
00344     //l.display();
00345     // Reconstruct the global nd_item
00346     index.set_local(local);
00347     // \todo Some strength reduction here
00348     index.set_global(local + id<Dimensions>(g.get_local_range())*g.get());
00349     // Call the user kernel at last
00350     f(index);
00351 };
00352
00353 // Then iterate on all the work-items of the work-group
00354 parallel_for_iterate<Dimensions,
00355                    range<Dimensions>,
00356                    decltype(reconstruct_item),
00357                    id<Dimensions>>> {
00358     g.get_local_range(),
00359     reconstruct_item,
00360     local };
00361 #endif
00362 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



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

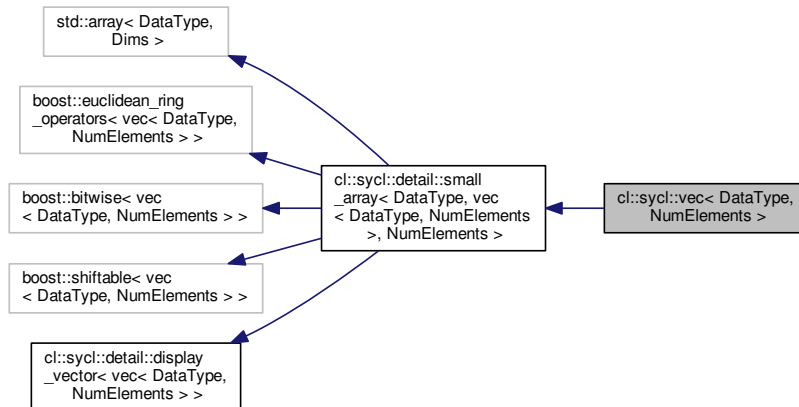
Todo move the implementation elsewhere

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

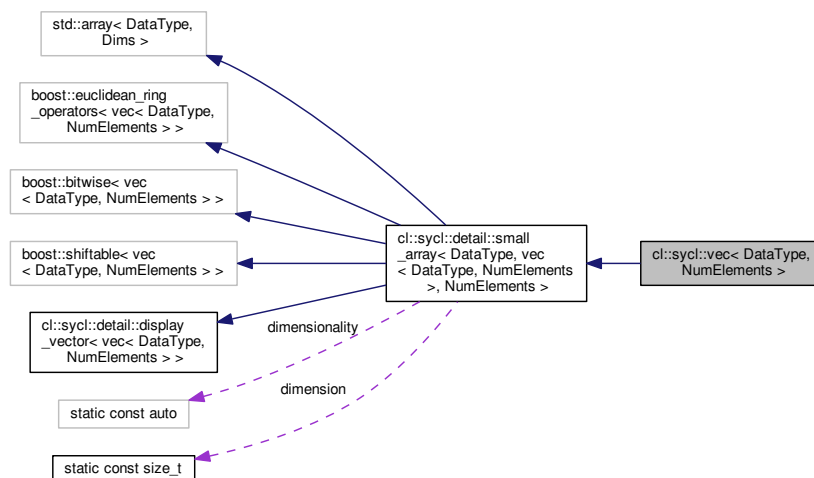
Todo rename in the specification `element_type` to `value_type`

Definition at line 42 of file [vec.hpp](#).

Inheritance diagram for `cl::sycl::vec< DataType, NumElements >`:



Collaboration diagram for `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` >, `NumElements` >

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](#).

```

00078                                     {
00079     static_assert(s <= V::dimension,
00080                  "The element i will not fit in the vector");
00081     return static_cast<std::array<Element, s>>(i);
00082 }
  
```

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](#).

```

00091                                     {
00092     return std::make_tuple(i);
00093 }
  
```

```
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](#).

```
00101 {
00102     // Concatenate the tuples returned by each flattening
00103     return std::tuple_cat(flatten<V>(i)...);
00104 }
```

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](#)
- [trisygl](#)

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 enqueueing 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](#).

```
00063         : char {
00064     local_space,
00065     global_space,
00066     global_and_local
00067 };
```

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](#).

```
00033     {
00034     read = 42, /**< Read-only access. Insist on the fact that
00035                 read_write != read + write */
00036     write, /**< Write-only access, but previous content *not* discarded
00037     read_write, /**< Read and write access
00038     discard_write, /**< Write-only access and previous content discarded
00039     discard_read_write, /**< Read and write access and previous
00040                         content discarded*/
00041     atomic /**< Atomic access
00042 };
```

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](#).

```
00048     {
00049         global_buffer = 2014, //< Just pick a random number...
00050         constant_buffer,
00051         local,
00052         image,
00053         host_buffer,
00054         host_image,
00055         image_array,
00056         pipe,
00057         blocking_pipe
00058     };
```

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 [opengl_device](#)
SYCL OpenCL device.
- class [opengl_kernel](#)
An abstraction of the OpenCL kernel.
- class [opengl_platform](#)
SYCL OpenCL platform. [More...](#)
- class [opengl_queue](#)
Some implementation details about the SYCL queue.
- struct [opengl_type](#)
Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device. [More...](#)
- struct [opengl_type< T, constant_address_space >](#)
Add an attribute for __constant address space. [More...](#)
- struct [opengl_type< T, generic_address_space >](#)
Add an attribute for __generic address space. [More...](#)
- struct [opengl_type< T, global_address_space >](#)
Add an attribute for __global address space. [More...](#)
- struct [opengl_type< T, local_address_space >](#)
Add an attribute for __local address space. [More...](#)
- struct [opengl_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 ParallelForFuncor >
void parallel_for_workitem (const group< Dimensions > &g, ParallelForFuncor 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::opencil_kernel > opencil_kernel::cache __attribute__((weak))`
- `template<std::size_t Dimensions = 1, typename ParallelForFuncor, typename Id >
void parallel_for (range< Dimensions > r, ParallelForFuncor f, Id)`
Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- `template<std::size_t Dimensions = 1, typename ParallelForFuncor >
void parallel_for (range< Dimensions > r, ParallelForFuncor 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 ParallelForFuncor >
void parallel_for (range< Dimensions > r, ParallelForFuncor 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 ParallelForFuncor >
void parallel_for_global_offset (range< Dimensions > global_size, id< Dimensions > offset, ParallelForFuncor f)`
Implementation of parallel_for with a range<> and an offset.
- `template<std::size_t Dimensions = 1, typename ParallelForFuncor >
void parallel_for (nd_range< Dimensions > r, ParallelForFuncor f)`
Implement a variation of parallel_for to take into account a nd_range<>
- `template<std::size_t Dimensions = 1, typename ParallelForFuncor >
void parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFuncor f)`
Implement the loop on the work-groups.

Variables

- `detail::cache< cl_device_id, detail::opencil_device > opencil_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](#).

```

00396                                     {
00397     command_group_handler->task->add_buffer(b, is_write_mode);
00398     return command_group_handler->task;
00399 }
  
```

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_width_char`, `device::native_vector_width_short`, `device::native_vector_width_int`, `device::native_vector_width_long_long`, `device::native_vector_width_float`, `device::native_vector_width_double`, `device::native_vector_width_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_width`, `device::image3d_max_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_OPENCL` `!=(= 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](#).

```
00037         : int {
00038     reference_count,
00039     num_devices,
00040     gl_interop
00041 };
```

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](#).

```
00056             : int {
00057     context,
00058     device,
00059     reference_count,
00060     properties
00061 };
```

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

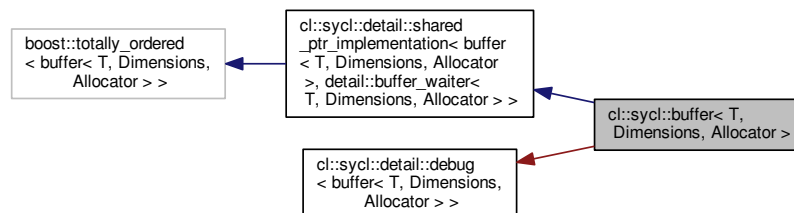
Class Documentation

10.1 cl::sycl::buffer< T, Dimensions, Allocator > Class Template Reference

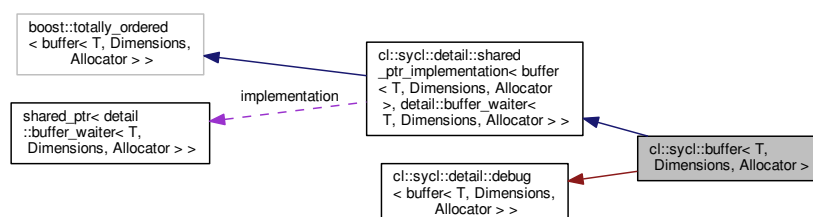
<T, Dimensions, Mode, Target>up data Data access and storage in SYCL

```
#include <accessor.hpp>
```

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



Collaboration diagram for cl::sycl::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 allo-`
`cator={})`
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< Dimen-`
`sions > &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 inferred 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	<i>r</i>	defines the size
in	<i>allocator</i>	is to be used by the SYCL runtime

Definition at line 111 of file [buffer.hpp](#).

References [cl::sycl::detail::waiter\(\)](#).

```

00111                                     {}
00112     : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
00113                                     { r }) }
00114     {}

```

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	<i>host_data</i>	points to the storage and values used by the buffer
in	<i>r</i>	defines the size
in	<i>allocator</i>	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\(\)](#).

```

00138                                     {}
00139     : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
00140                                     { host_data, r }) }
00141     {}

```

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	<i>host_data</i>	points to the storage and values used by the buffer
in	<i>r</i>	defines the size
in	<i>allocator</i>	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\(\)](#).

```

00160                                     {}))
00161     : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
00162         { host_data, r }) }
00163     {}
  
```

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

<code>in, out</code>	<code>host_data</code>	points to the storage and values used by the buffer
<code>in</code>	<code>r</code>	defines the size
<code>in</code>	<code>allocator</code>	is to be used by the SYCL runtime, of type <code>cl::sycl::buffer_allocator<T></code> 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

<code>in, out</code>	<code>host_data</code>	points to the storage and values used by the buffer
<code>in</code>	<code>r</code>	defines the size
<code>in, out</code>	<code>m</code>	is the mutex used to protect the data access
<code>in</code>	<code>allocator</code>	is to be used by the SYCL runtime, of type <code>cl::sycl::buffer_allocator<T></code> 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\(\)](#).

```
00217         {}
00218     : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
00219         { host_data, buffer_range }) }
00220     {}
```

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	<i>host_data</i>	points to the storage and values used to initialize the buffer
in	<i>r</i>	defines the size
in	<i>allocator</i>	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\(\)](#).

```
00245         {}
00246     // Just delegate to the constructor with normal pointer
00247     : buffer(host_data.get(), buffer_range, allocator) {
00248         // Then release the host_data memory
00249         host_data.release();
00250     }
```


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

<code>in, out</code>	<code>start_iterator</code>	points to the first element to copy
<code>in</code>	<code>end_iterator</code>	points to just after the last element to copy
<code>in</code>	<code>allocator</code>	is to be used by the SYCL runtime, of type <code>cl::sycl::buffer_allocator<T></code> 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()`.

```

00292         {} ) :
00293     implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
00294         { start_iterator, end_iterator }) }
00295     {}
  
```

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	<i>b</i>	is the buffer with the real data
in	<i>base_index</i>	specifies the origin of the sub-buffer inside the buffer b
in	<i>sub_range</i>	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	<i>mem_object</i>	is the OpenCL memory object to use
in, out	<i>from_queue</i>	is the queue associated to the memory object
in	<i>available_event</i>	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 constitutes 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

	<i>Mode</i>	is the requested access mode
	<i>Target</i>	is the type of object to be accessed
in	<i>command_group_handler</i>	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](#).

```
00365     {
00366     static_assert(Target == access::target::global_buffer
00367                   || Target == access::target::constant_buffer,
00368                   "get_access(handler) can only deal with access::global_buffer"
00369                   " or access::constant_buffer (for host_buffer accessor"
00370                   " do not use a command group handler");
00371     return { *this, command_group_handler };
00372 }
```

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

<i>Mode</i>	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](#).

```
00386     {
00387     static_assert(Target == access::target::host_buffer,
00388                   "get_access() without a command group handler is only"
00389                   " for host_buffer accessor");
00390     return { *this };
00391 }
```

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](#).

```
00415     {
00416     return implementation->implementation->get_count();
00417 }
```

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`.

```
00402         {
00403         /* Interpret the shape which is a pointer to the first element as an
00404            array of Dimensions elements so that the range<Dimensions>
00405            constructor is happy with this collection
00406         */
00407         return implementation->implementation->get_range();
00408     }
```

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`.

```
00428         {
00429         return implementation->implementation->get_size();
00430     }
```

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`.

```
00455         {
00456         return implementation->implementation->read_only();
00457     }
```

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`.

```
00487         {
00488     implementation->implementation->set_final_data(std::move(finalData));
00489     }
```

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 c.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`.

```
00445         {
00446     // Rely on the shared_ptr<> use_count()
00447     return implementation.use_count();
00448     }
```

The documentation for this class was generated from the following files:

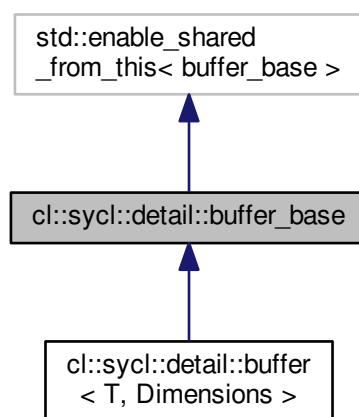
- `include/CL/sycl/accessor.hpp`
- `include/CL/sycl/buffer.hpp`

10.2 cl::sycl::detail::buffer_base Struct Reference

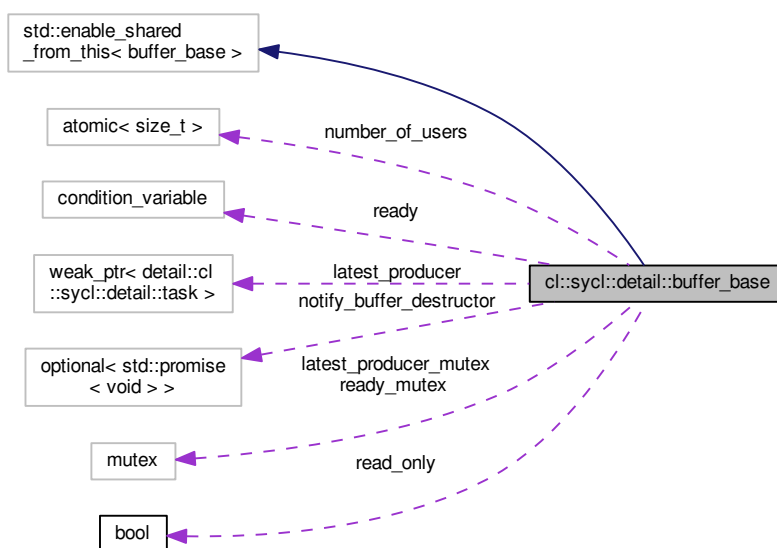
Factorize some template independent buffer aspects in a base class.

```
#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](#).

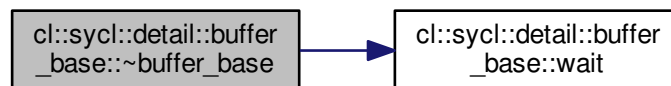
References [wait\(\)](#).

```

00073     {
00074         wait();
00075         // If there is the last SYCL user buffer waiting, notify it
00076         if (notify_buffer_destructor)
00077             notify_buffer_destructor->set_value();
00078     }

```

Here is the call graph for this function:



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](#).

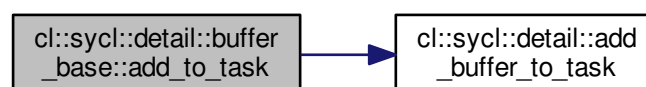
References [cl::sycl::detail::add_buffer_to_task\(\)](#).

```

00130     {
00131         return add_buffer_to_task(command_group_handler,
00132                                   shared_from_this(),
00133                                   is_write_mode);
00134     }

```

Here is the call graph for this function:



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](#).

```
00107         {
00108     std::lock_guard<std::mutex> lg { latest_producer_mutex };
00109     // Return the valid shared_ptr to the task, if any
00110     return latest_producer.lock();
00111 }
```

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](#).

```
00099         {
00100     if (--number_of_users == 0)
00101         // Notify the host consumers or the buffer destructor that it is ready
00102         ready.notify_all();
00103 }
```

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](#).

```
00118         {
00119     std::lock_guard<std::mutex> lg { latest_producer_mutex };
00120     using std::swap;
00121
00122     swap(newer_latest_producer, latest_producer);
00123     // Return the valid shared_ptr to the previous producing task, if any
00124     return newer_latest_producer.lock();
00125 }
```

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](#).

```
00092         {
00093     // Increment the use count
00094     ++number_of_users;
00095 }
```

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](#).

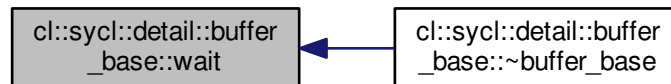
Referenced by [~buffer_base\(\)](#).

```

00082     {
00083         std::unique_lock<std::mutex> ul { ready_mutex };
00084         ready.wait(ul, [&] {
00085             // When there is no producer for this buffer, we are ready to use it
00086             return number_of_users == 0;
00087         });
00088     }

```

Here is the caller graph for this function:



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:

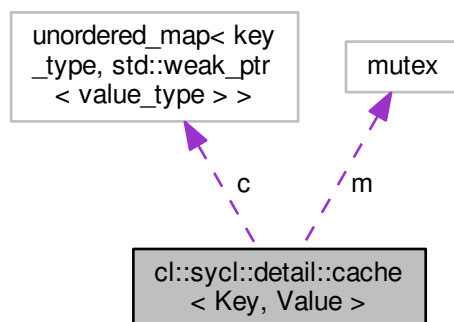
- [include/CL/sycl/buffer/detail/buffer_base.hpp](#)

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 >`:



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	<i>k</i>	is the key used to retrieve the value
in	<i>create_element</i>	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

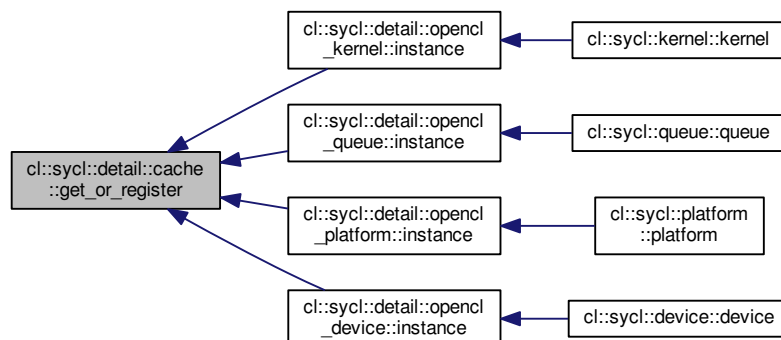
Definition at line 62 of file [cache.hpp](#).

Referenced by [cl::sycl::detail::opencil_kernel::instance\(\)](#), [cl::sycl::detail::opencil_queue::instance\(\)](#), [cl::sycl::detail::opencil_platform::instance\(\)](#), and [cl::sycl::detail::opencil_device::instance\(\)](#).

```

00063                                     {
00064     std::lock_guard<std::mutex> lg { m };
00065
00066     auto i = c.find(k);
00067     if (i != c.end())
00068         // Return the found element
00069         return std::shared_ptr<value_type>{ i->second };
00070
00071     // Otherwise create and insert a new element
00072     std::shared_ptr<value_type> e { create_element() };
00073     c.insert({ k, e });
00074     return e;
00075 }
```

Here is the caller graph for this function:



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

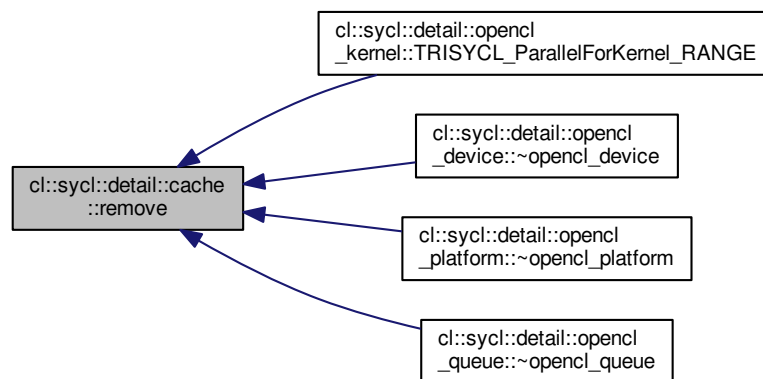
in	<i>k</i>	is the key associated to the value to remove from the cache
----	----------	---

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\(\)](#).

```
00083     {
00084         std::lock_guard<std::mutex> lg { m };
00085         c.erase(k);
00086     }
```

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](#).

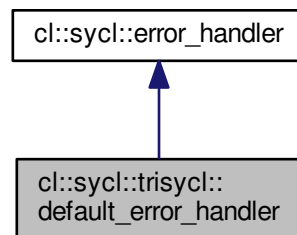
The documentation for this class was generated from the following file:

- [include/CL/sycl/detail/cache.hpp](#)

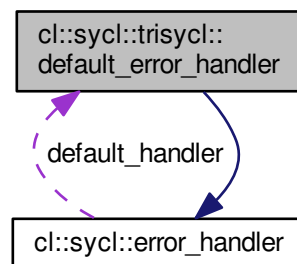
10.4 cl::sycl::trisycl::default_error_handler Struct Reference

```
#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

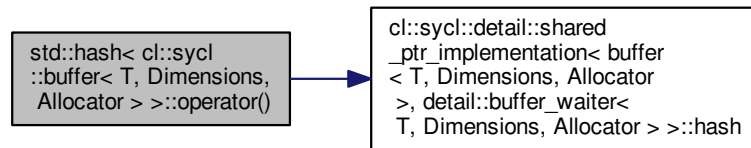
10.7.2.1 `template<typename T , std::size_t Dimensions, typename Allocator > auto std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >::operator() (const cl::sycl::buffer< T, Dimensions, Allocator > & b) const`
[inline]

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\(\)](#).

```
00510                                     {
00511     // Forward the hashing to the implementation
00512     return b.hash();
00513 }
```

Here is the call graph for this function:



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]`

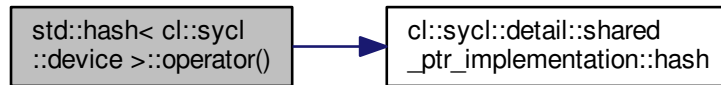
Definition at line [258](#) of file [device.hpp](#).

References [cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash\(\)](#).

```

00258
00259     // Forward the hashing to the implementation
00260     return d.hash();
00261 }
```

Here is the call graph for this function:



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]

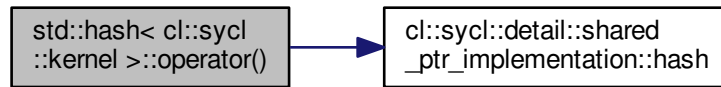
Definition at line [125](#) of file [kernel.hpp](#).

References [cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash\(\)](#).

```

00125                                     {
00126     // Forward the hashing to the implementation
00127     return k.hash();
00128 }
```

Here is the call graph for this function:



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]`

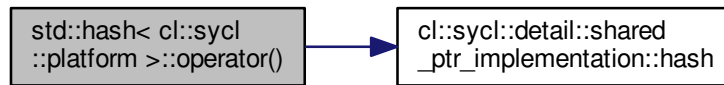
Definition at line 194 of file `platform.hpp`.

References `cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash()`.

```

00194                                     {
00195     // Forward the hashing to the implementation
00196     return p.hash();
00197 }
```

Here is the call graph for this function:



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]

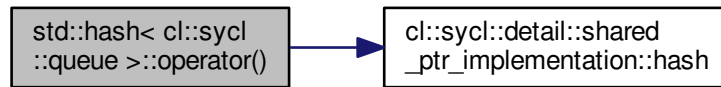
Definition at line [362](#) of file [queue.hpp](#).

References [cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash\(\)](#).

```

00362                                     {
00363     // Forward the hashing to the implementation
00364     return q.hash();
00365 }
```

Here is the call graph for this function:



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

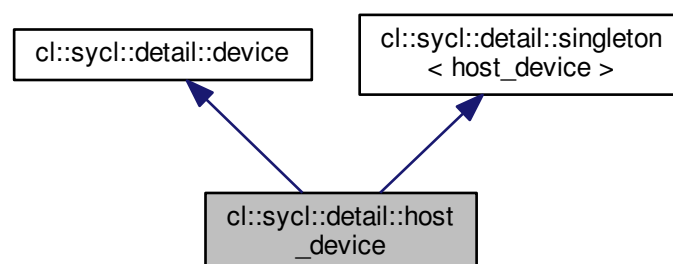
- `include/CL/sycl/queue.hpp`

10.12 cl::sycl::detail::host_device Class Reference

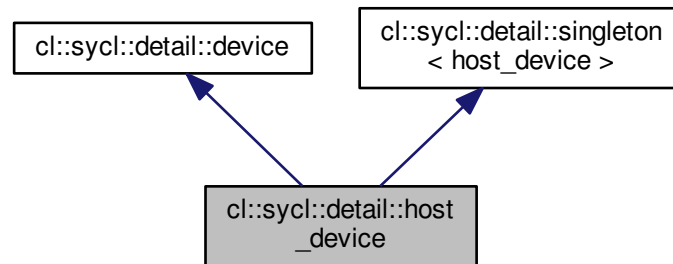
SYCL host device.

```
#include <host_device.hpp>
```

Inheritance diagram for `cl::sycl::detail::host_device`:



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](#).

```
00042                                     {
00043     throw non_cl_error("The host device has no OpenCL device");
00044 }
```

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

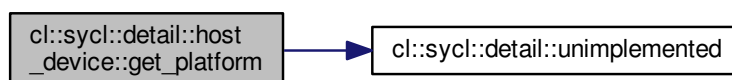
Implements [cl::sycl::detail::device](#).

Definition at line 78 of file [host_device.hpp](#).

References [cl::sycl::detail::unimplemented\(\)](#).

```
00078                                     {
00079     detail::unimplemented();
00080     return {};
00081 }
```

Here is the call graph for this function:



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

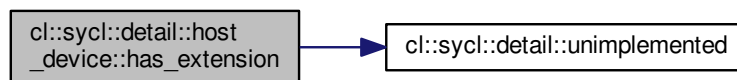
Implements `cl::sycl::detail::device`.

Definition at line 102 of file `host_device.hpp`.

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

```
00102                                     {
00103     detail::unimplemented();
00104     return {};
00105 }
```

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`.

```
00067                                     {
00068     return false;
00069 }
```

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`.

```
00055                                     {
00056     return false;
00057 }
```

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:

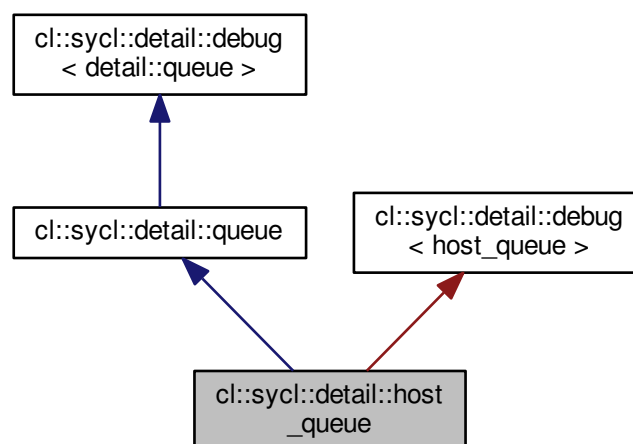
- [include/CL/sycl/device/detail/host_device.hpp](#)

10.13 cl::sycl::detail::host_queue Class Reference

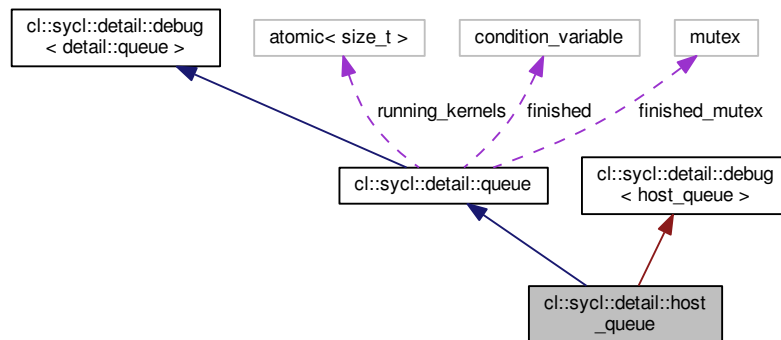
Some implementation details about the SYCL queue.

```
#include <host_queue.hpp>
```

Inheritance diagram for `cl::sycl::detail::host_queue`:



Collaboration diagram for `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](#).

```
00038                                     {
00039     throw non_cl_error("The host queue has no OpenCL command queue");
00040 }
```

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](#).

```
00048                                     {
00049     throw non_cl_error("The host queue has no OpenCL command queue");
00050 }
```

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](#).

```
00055                                     {
00056     // Return the default context which is the host context
00057     return {};
00058 }
```

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](#).

```
00062                                     {
00063     // Return the default device which is the host device
00064     return {};
00065 }
```

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](#).

```
00069                                     {
00070     return true;
00071 }
```

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

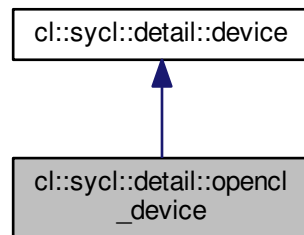
- [include/CL/sycl/queue/detail/host_queue.hpp](#)

10.14 cl::sycl::detail::opengl_device Class Reference

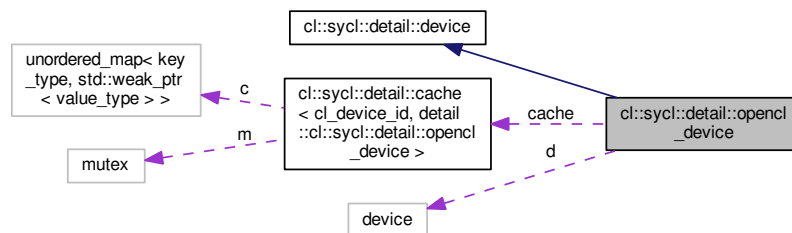
SYCL OpenCL device.

```
#include <opengl_device.hpp>
```

Inheritance diagram for `cl::sycl::detail::opengl_device`:



Collaboration diagram for `cl::sycl::detail::opengl_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.
- `~opengl_device` () override
Unregister from the cache on destruction.

Static Public Member Functions

- static std::shared_ptr< [opengl_device](#) > [instance](#) (const boost::compute::device &d)

Private Member Functions

- [opengl_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::opengl_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 [opengl_device.hpp](#).

10.14.2 Constructor & Destructor Documentation

10.14.2.1 `cl::sycl::detail::opengl_device::opengl_device (const boost::compute::device & d)` `[inline]`, `[private]`

Only the instance factory can built it.

Definition at line 120 of file [opengl_device.hpp](#).

```
00120 : d { d } {}
```

10.14.2.2 `cl::sycl::detail::opengl_device::~~opengl_device ()` `[inline]`, `[override]`

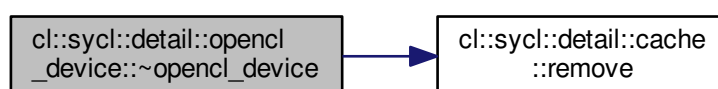
Unregister from the cache on destruction.

Definition at line 125 of file [opengl_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::openccl_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 [openccl_device.hpp](#).

```
00045                                     {
00046     return d.id();
00047 }
```

10.14.3.2 `cl::sycl::platform cl::sycl::detail::openccl_device::get_platform () const` `[inline]`, `[override]`, `[virtual]`

Return the platform of device.

Return synchronous errors via the SYCL exception class.

Todo To be implemented

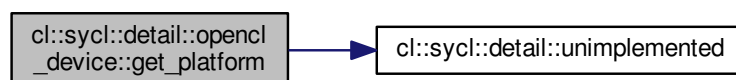
Implements [cl::sycl::detail::device](#).

Definition at line 80 of file [openccl_device.hpp](#).

References [cl::sycl::detail::unimplemented\(\)](#).

```
00080                                     {
00081     detail::unimplemented();
00082     return {};
00083 }
```

Here is the call graph for this function:



10.14.3.3 `bool cl::sycl::detail::opengl_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

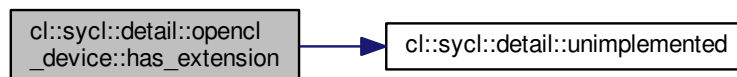
Implements `cl::sycl::detail::device`.

Definition at line 104 of file `opengl_device.hpp`.

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

```
00104                                     {
00105     detail::unimplemented();
00106     return {};
00107 }
```

Here is the call graph for this function:



10.14.3.4 `static std::shared_ptr<opengl_device> cl::sycl::detail::opengl_device::instance (const boost::compute::device & d)` `[inline]`, `[static]`

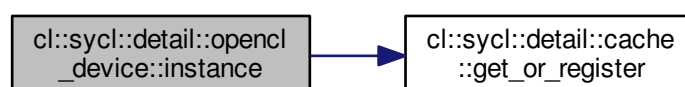
Definition at line 112 of file `opengl_device.hpp`.

References `cl::sycl::detail::cache< Key, Value >::get_or_register()`.

Referenced by `cl::sycl::device::device()`.

```
00112                                     {
00113     return cache.get_or_register(d.id(),
00114                                [&] { return new opengl_device { d }; });
00115 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.14.3.5 `bool cl::sycl::detail::opengl_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 [opengl_device.hpp](#).

```

00069                                     {
00070     return d.type() == boost::compute::device::accelerator;
00071 }
  
```

10.14.3.6 `bool cl::sycl::detail::opengl_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 [opengl_device.hpp](#).

```

00057                                     {
00058     return d.type() == boost::compute::device::cpu;
00059 }
  
```

10.14.3.7 `bool cl::sycl::detail::opengl_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 [opengl_device.hpp](#).

```

00063                                     {
00064     return d.type() == boost::compute::device::gpu;
00065 }
  
```

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](#).

```
00051                                     {
00052     return false;
00053 }
```

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:

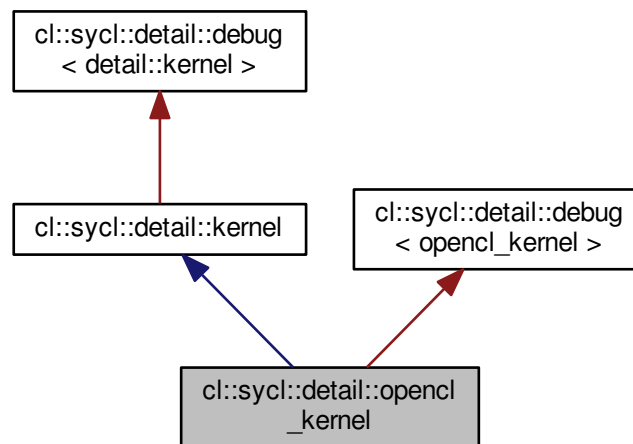
- [include/CL/sycl/device/detail/opencl_device.hpp](#)

10.15 cl::sycl::detail::opencl_kernel Class Reference

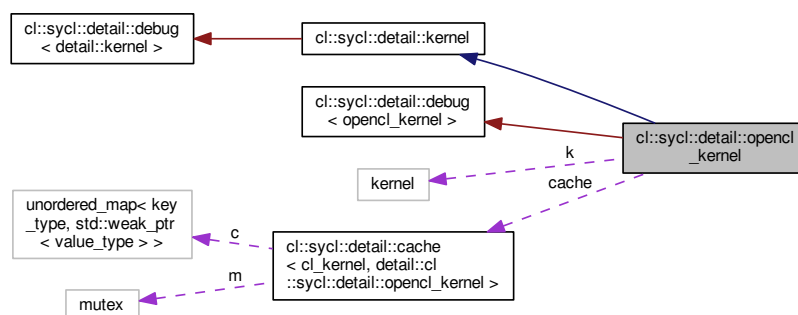
An abstraction of the OpenCL kernel.

```
#include <opencl_kernel.hpp>
```

Inheritance diagram for cl::sycl::detail::opencl_kernel:



Collaboration diagram for cl::sycl::detail::opencl_kernel:



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_ParallelForKernel_RANGE`(3)~`opencl_kernel`() override
Unregister from the cache on destruction.

Static Public Member Functions

- static std::shared_ptr< [opengl_kernel](#) > [instance](#) (const boost::compute::kernel &k)

Private Member Functions

- [opengl_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::opengl_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 [opengl_kernel.hpp](#).

10.15.2 Constructor & Destructor Documentation

10.15.2.1 `cl::sycl::detail::opengl_kernel::opengl_kernel (const boost::compute::kernel & k) [inline], [private]`

Definition at line 41 of file [opengl_kernel.hpp](#).

```
00041 : k { k } {}
```

10.15.3 Member Function Documentation

10.15.3.1 `cl_kernel cl::sycl::detail::opengl_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 [opengl_kernel.hpp](#).

```
00057                                     {
00058     /// \todo Test error and throw. Externalize this feature in Boost.Compute?
00059     clRetainKernel(k);
00060     return k.get();
00061 }
```

10.15.3.2 `boost::compute::kernel cl::sycl::detail::openccl_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 [openccl_kernel.hpp](#).

References [k](#), and [cl::sycl::detail::unimplemented\(\)](#).

```
00068                                     {
00069     return k;
00070 }
```

Here is the call graph for this function:



10.15.3.3 `static std::shared_ptr<openccl_kernel> cl::sycl::detail::openccl_kernel::instance (const boost::compute::kernel & k)` `[inline]`, `[static]`

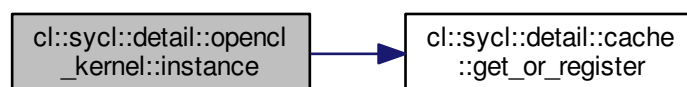
Definition at line 47 of file [openccl_kernel.hpp](#).

References [cl::sycl::detail::cache< Key, Value >::get_or_register\(\)](#).

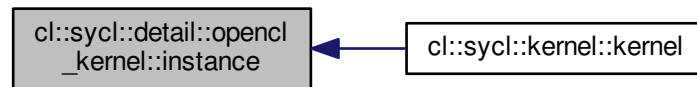
Referenced by [cl::sycl::kernel::kernel\(\)](#).

```
00047                                     {
00048     return cache.get_or_register(k.get(),
00049                                [&] { return new openccl_kernel { k }; });
00050 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.15.3.4 cl::sycl::detail::openccl_kernel::TRISYCL_ParallelForKernel_RANGE (1) [inline], [override]

Unregister from the cache on destruction.

Definition at line 110 of file [openccl_kernel.hpp](#).

References [cl::sycl::detail::__attribute__](#), [cache](#), and [cl::sycl::detail::cache< Key, Value >::remove\(\)](#).

```

00117 {
00118     cache.remove(k.get());
00119 }
```

Here is the call graph for this function:



10.15.4 Member Data Documentation

10.15.4.1 detail::cache<cl_kernel, detail::openccl_kernel> cl::sycl::detail::openccl_kernel::cache [static], [private]

A cache to always return the same alive kernel for a given OpenCL kernel.

C++11 guarantees the static construction is thread-safe

Definition at line 39 of file [openccl_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:

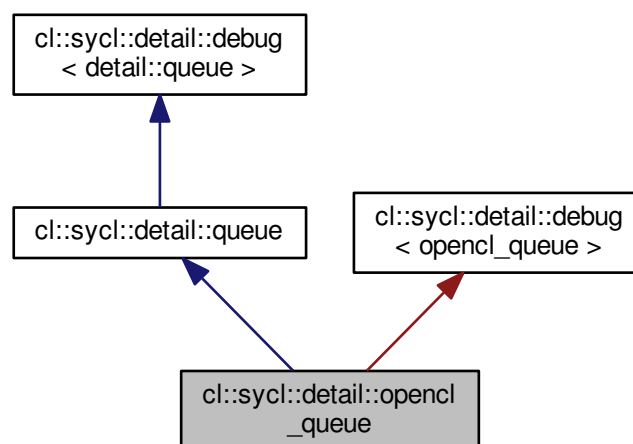
- [include/CL/sycl/kernel/detail/opencl_kernel.hpp](#)

10.16 `cl::sycl::detail::opencl_queue` Class Reference

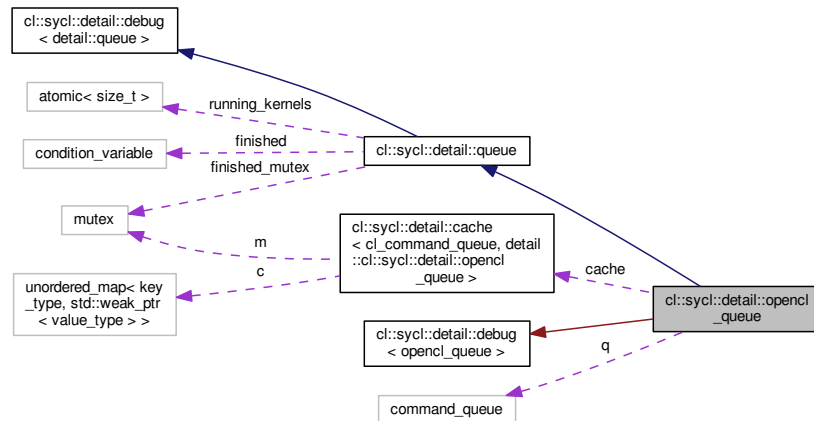
Some implementation details about the SYCL queue.

```
#include <opencl_queue.hpp>
```

Inheritance diagram for `cl::sycl::detail::opencl_queue`:



Collaboration diagram for cl::sycl::detail::opencl_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]`

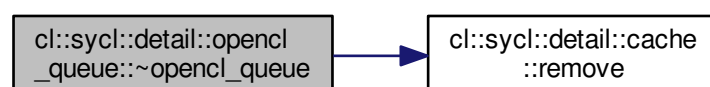
Unregister from the cache on destruction.

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::opengl_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 [opengl_queue.hpp](#).

```
00036                                     {
00037     return q.get();
00038 }
```

10.16.3.2 `boost::compute::command_queue& cl::sycl::detail::opengl_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 [opengl_queue.hpp](#).

References [q](#).

```
00042                                     {
00043     return q;
00044 }
```

10.16.3.3 `cl::sycl::context cl::sycl::detail::opengl_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 [opengl_queue.hpp](#).

```
00049                                     {
00050 //     return q.get_context();
00051     return {};
00052 }
```

10.16.3.4 `cl::sycl::device cl::sycl::detail::opencil_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 [opencil_queue.hpp](#).

```
00056                                     {
00057     return q.get_device();
00058 }
```

10.16.3.5 `static std::shared_ptr<opencil_queue> cl::sycl::detail::opencil_queue::instance (const boost::compute::command_queue & q)` `[inline],[static]`

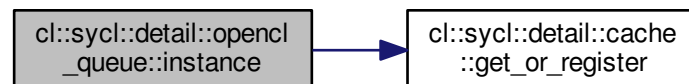
Definition at line 75 of file [opencil_queue.hpp](#).

References [cl::sycl::detail::cache< Key, Value >::get_or_register\(\)](#).

Referenced by [cl::sycl::queue::queue\(\)](#).

```
00075                                     {
00076     return cache.get_or_register(q.get(),
00077                                [&] { return new opencil_queue { q }; });
00078 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.16.3.6 `bool cl::sycl::detail::opcnl_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 `opcnl_queue.hpp`.

```
00062                                     {
00063     return false;
00064 }
```

10.16.4 Member Data Documentation

10.16.4.1 `detail::cache<cl_command_queue, detail::opcnl_queue> cl::sycl::detail::opcnl_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 `opcnl_queue.hpp`.

Referenced by `~opcnl_queue()`.

10.16.4.2 `boost::compute::command_queue cl::sycl::detail::opcnl_queue::q` `[private]`

Use the Boost Compute abstraction of the OpenCL command queue.

Definition at line 26 of file `opcnl_queue.hpp`.

Referenced by `get_boost_compute()`.

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

- `include/CL/sycl/queue/detail/opcnl_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:

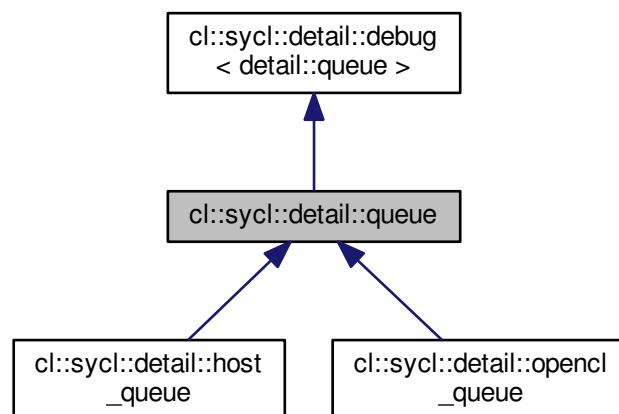
- [include/CL/sycl/info/param_traits.hpp](#)

10.18 cl::sycl::detail::queue Struct Reference

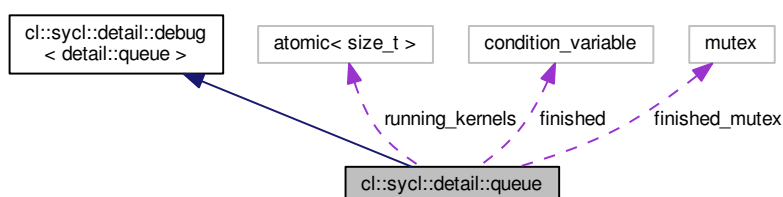
Some implementation details about the SYCL queue.

```
#include <queue.hpp>
```

Inheritance diagram for cl::sycl::detail::queue:



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](#).

```
00041     {
00042         running\_kernels = 0;
00043     }
```

10.18.2.2 `virtual cl::sycl::detail::queue::~~queue () [inline], [virtual]`

Wait for all kernel completion before the queue destruction.

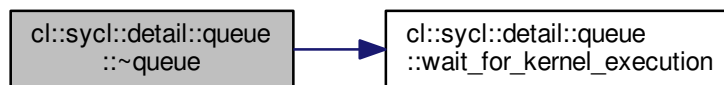
Todo Update according spec since queue destruction is non blocking

Definition at line 114 of file [queue.hpp](#).

References [wait_for_kernel_execution\(\)](#).

```
00114         {
00115             wait_for_kernel_execution();
00116         }
```

Here is the call graph for this function:



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](#).

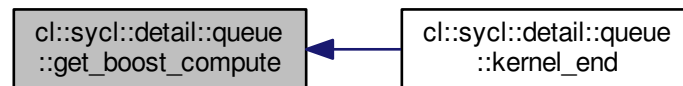
10.18.3.2 virtual boost::compute::command_queue& cl::sycl::detail::queue::get_boost_compute () [pure virtual]

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:



10.18.3.3 virtual cl::sycl::context cl::sycl::detail::queue::get_context () const [pure virtual]

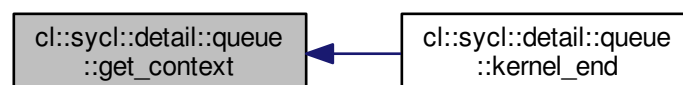
Return the SYCL queue's context.

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:



10.18.3.4 `virtual cl::sycl::device cl::sycl::detail::queue::get_device () const` [pure virtual]

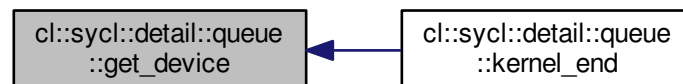
Return the SYCL device the queue is associated with.

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:



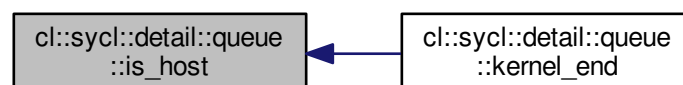
10.18.3.5 `virtual bool cl::sycl::detail::queue::is_host () const` [pure virtual]

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:



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](#).

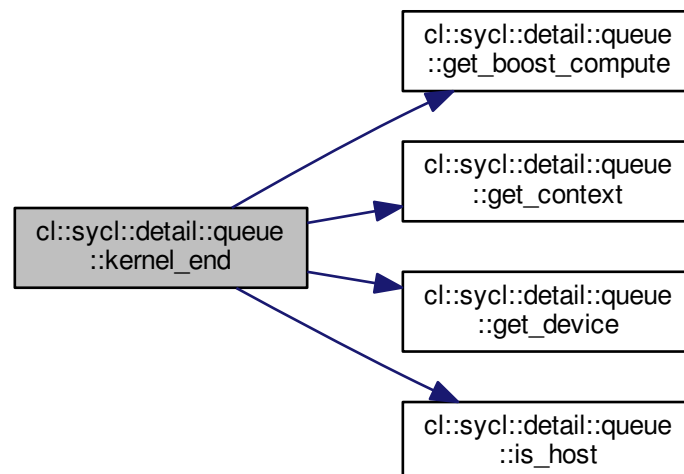
References [get_boost_compute\(\)](#), [get_context\(\)](#), [get_device\(\)](#), [is_host\(\)](#), and [TRISYCL_DUMP_T](#).

```

00066     {
00067         TRISYCL_DUMP_T("A kernel of the queue ended");
00068         if (--running_kernels == 0) {
00069             /* It was the last kernel running, so signal the queue just in
00070              * case it was working for it for completion */
00071             finished.notify_one();
00072         }
00073     }

```

Here is the call graph for this function:



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](#).

```

00058     {
00059         TRISYCL_DUMP_T("A kernel has been added to the queue");
00060         // One more kernel
00061         ++running_kernels;
00062     }

```

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](#).

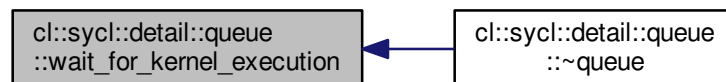
References [TRISYCL_DUMP_T](#).

Referenced by [~queue\(\)](#).

```

00047                                     {
00048     TRISYCL_DUMP_T("Queue waiting for kernel completion");
00049     std::unique_lock<std::mutex> ul { finished_mutex };
00050     finished.wait(ul, [&] {
00051         // When there is no kernel running in this queue, we are ready to go
00052         return running_kernels == 0;
00053     });
00054 }
```

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:

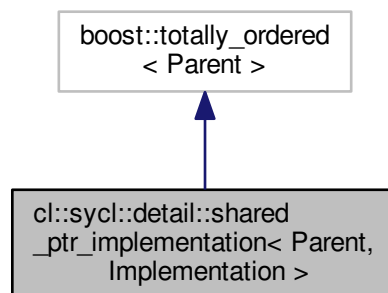
- [include/CL/sycl/queue/detail/queue.hpp](#)

10.19 `cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >` Struct Template Reference

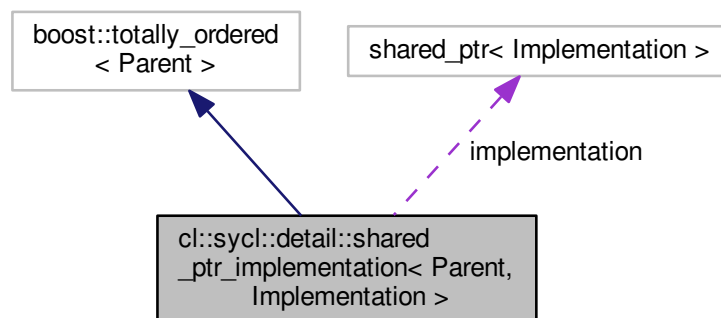
Provide an implementation as `shared_ptr` with total ordering and hashing to be used with algorithms and in (un)ordered containers.

```
#include <shared_ptr_implementation.hpp>
```

Inheritance diagram for `cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >`:



Collaboration diagram for `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](#).

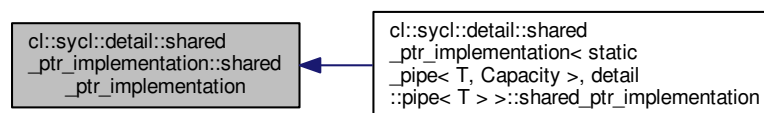
```
00053 : implementation { i } {}
```

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:



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]`

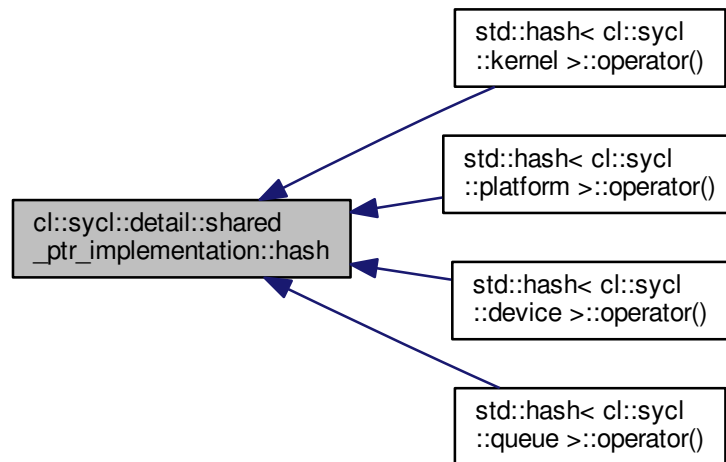
Forward the hashing for unordered containers to the implementation.

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:



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](#).

```

00079     {
00080     return implementation < other.implementation;
00081     }
  
```

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](#).

```

00066     {
00067     return implementation == other.implementation;
00068     }
  
```


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:

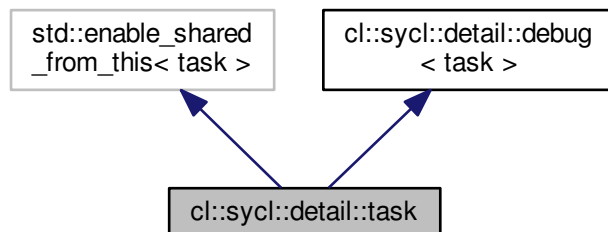
- `include/CL/sycl/detail/singleton.hpp`

10.21 cl::sycl::detail::task Struct Reference

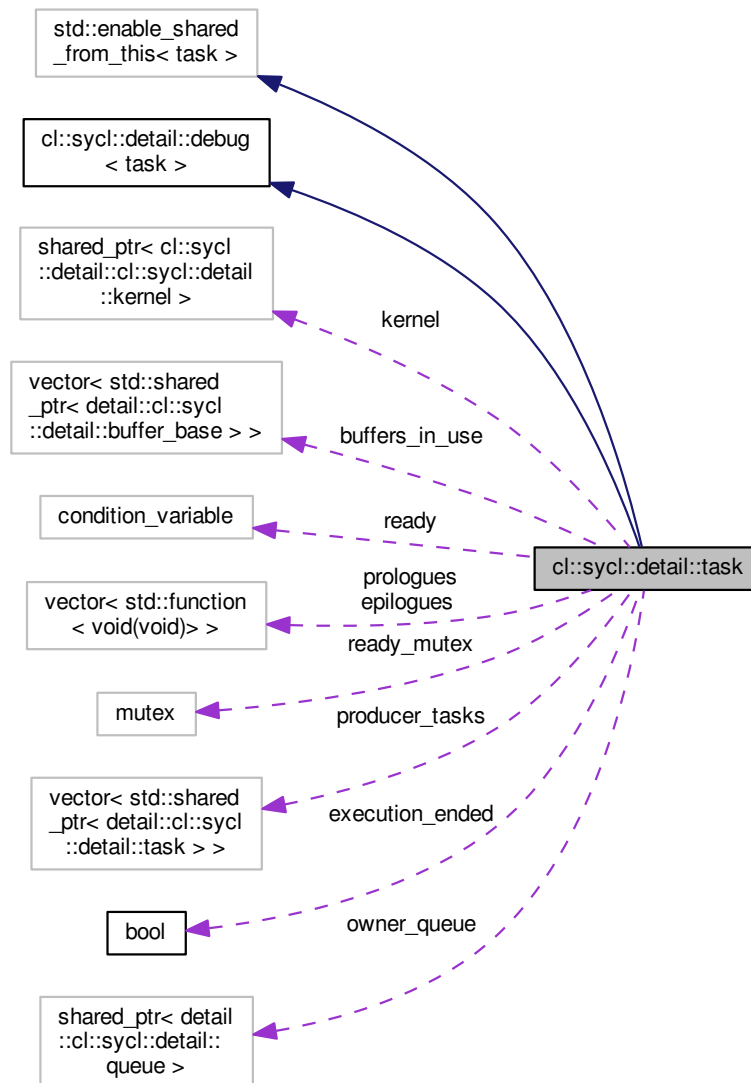
The abstraction to represent SYCL tasks executing inside `command_group`.

```
#include <task.hpp>
```

Inheritance diagram for `cl::sycl::detail::task`:



Collaboration diagram for cl::sycl::detail::task:



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](#).

```
00071      : 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](#).

```
00168      {
00169      TRISYCL_DUMP_T("Add buffer " << buf << " in task " << this);
00170      /* Keep track of the use of the buffer to notify its release at
00171         the end of the execution */
00172      buffers_in_use.push_back(buf);
00173      // To be sure the buffer does not disappear before the kernel can run
00174      buf->use();
00175
00176      std::shared_ptr<detail::task> latest_producer;
00177      if (is_write_mode) {
00178          /* Set this task as the latest producer of the buffer so that
00179             another kernel may wait on this task */
00180          latest_producer = buf->set_latest_producer(shared_from_this());
00181      }
00182      else
00183          latest_producer = buf->get_latest_producer();
00184
00185      /* If the buffer is to be produced by a task, add the task in the
00186         producer list to wait on it before running the task core */
00187      if (latest_producer)
00188          producer_tasks.push_back(latest_producer);
00189  }
```

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](#).

```
00219      {
00220      epilogues.push_back(f);
00221  }
```

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](#).

```
00213                                     {
00214     prologues.push_back (f);
00215 }
```

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](#).

```
00240                                     {
00241     if (!kernel)
00242         throw non_cl_error("Cannot use an OpenCL kernel in this context");
00243     return *kernel;
00244 }
```

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](#).

```
00225     {
00226     return owner\_queue;
00227 }
```

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](#).

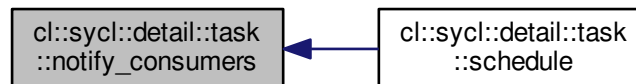
Referenced by [schedule\(\)](#).

```

00143         {
00144     TRISYCL_DUMP_T("Notify all the task waiting for this task " << this);
00145     execution_ended = true;
00146     /* \todo Verify that the memory model with the notify does not
00147        require some fence or atomic */
00148     ready.notify_all();
00149 }

```

Here is the caller graph for this function:



10.21.3.7 void cl::sycl::detail::task::postlude() [inline]

Execute the epilogues.

Definition at line 203 of file [task.hpp](#).

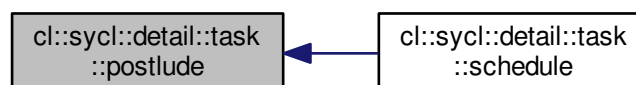
Referenced by [schedule\(\)](#).

```

00203         {
00204     for (const auto &p : epilogues)
00205         p();
00206     /* Free the functors that may own an accessor owning a buffer
00207        preventing the command group to complete */
00208     epilogues.clear();
00209 }

```

Here is the caller graph for this function:



10.21.3.8 void cl::sycl::detail::task::prelude () [inline]

Execute the prologues.

Definition at line 193 of file [task.hpp](#).

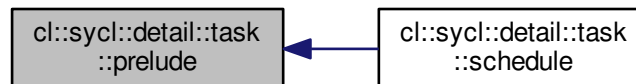
Referenced by [schedule\(\)](#).

```

00193         {
00194     for (const auto &p : prologues)
00195         p();
00196     /* Free the functors that may own an accessor owning a buffer
00197        preventing the command group to complete */
00198     prologues.clear();
00199 }

```

Here is the caller graph for this function:



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](#).

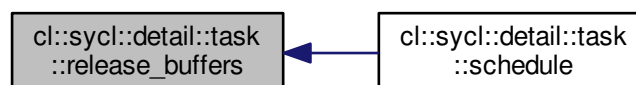
Referenced by [schedule\(\)](#).

```

00134         {
00135     TRISYCL_DUMP_T("Task " << this << " releases the written buffers");
00136     for (auto b: buffers_in_use)
00137         b->release();
00138     buffers_in_use.clear();
00139 }

```

Here is the caller graph for this function:



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](#).

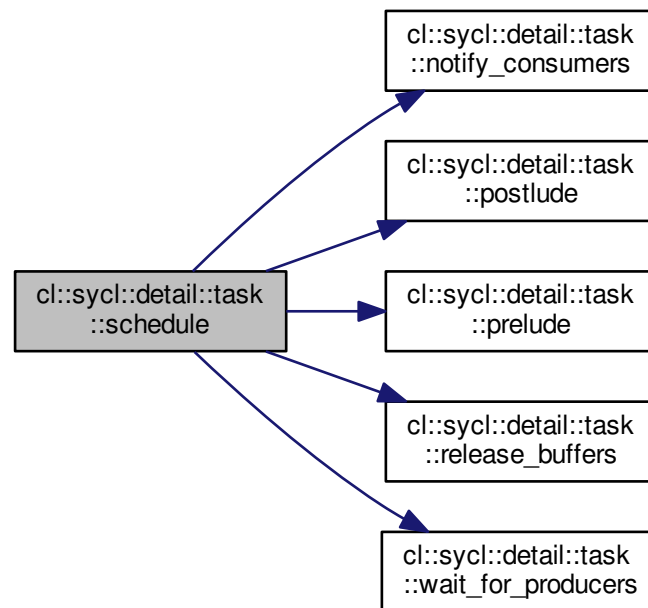
References [notify_consumers\(\)](#), [postlude\(\)](#), [prelude\(\)](#), [release_buffers\(\)](#), [TRISYCL_DUMP_T](#), and [wait_for_producers\(\)](#).

```

00075                                     {
00076     /* To keep a copy of the task shared_ptr after the end of the
00077        command group, capture it by copy in the following lambda. This
00078        should be easier in C++17 with move semantics on capture
00079    */
00080     auto task = shared_from_this();
00081     auto execution = [=] {
00082         // Wait for the required tasks to be ready
00083         task->wait_for_producers();
00084         task->prelude();
00085         TRISYCL_DUMP_T("Execute the kernel");
00086         // Execute the kernel
00087         f();
00088         task->postlude();
00089         // Release the buffers that have been written by this task
00090         task->release_buffers();
00091         // Notify the waiting tasks that we are done
00092         task->notify_consumers();
00093         // Notify the queue we are done
00094         owner_queue->kernel_end();
00095         TRISYCL_DUMP_T("Task thread exit");
00096     };
00097     /* Notify the queue that there is a kernel submitted to the
00098        queue. Do not do it in the task constructor so that we can deal
00099        with command group without kernel and if we put it inside the
00100        thread, the queue may have finished before the thread is
00101        scheduled */
00102     owner_queue->kernel_start();
00103     /* \todo it may be implementable with packaged_task that would
00104        deal with exceptions in kernels
00105    */
00106     #if TRISYCL_ASYNC
00107     /* If in asynchronous execution mode, execute the functor in a new
00108        thread */
00109     std::thread thread(execution);
00110     TRISYCL_DUMP_T("Task thread started");
00111     /** Detach the thread since it will synchronize by its own means
00112    */
00113     \todo This is an issue if there is an exception in the kernel
00114     /*
00115     thread.detach();
00116     #else
00117     // Just a synchronous execution otherwise
00118     execution();
00119     #endif
00120     }

```

Here is the call graph for this function:



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](#).

```

00231                                     {
00232     kernel = k;
00233 }
  
```

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](#).

```

00156     {
00157     TRISYCL_DUMP_T("The task wait for task " << this << " to end");
00158     std::unique_lock<std::mutex> ul { ready_mutex };
00159     ready.wait(ul, [&] { return execution_ended; });
00160 }
  
```

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\(\)](#).

```

00124         {
00125     TRISYCL_DUMP_T("Task " << this << " waits for the producer tasks");
00126     for (auto &t : producer_tasks)
00127         t->wait();
00128     // We can let the producers rest in peace
00129     producer_tasks.clear();
00130 }

```

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`

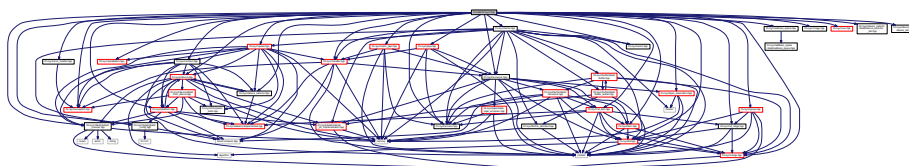
Chapter 11

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

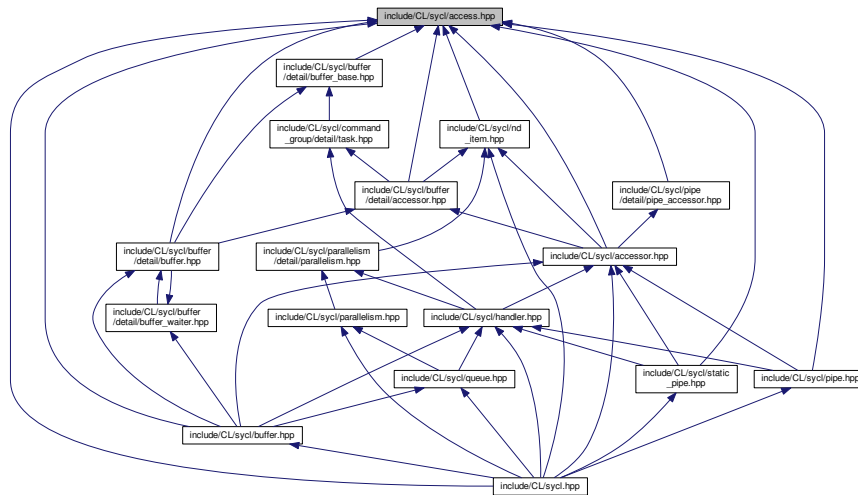
```

00001 /** \file
00002
00003     \mainpage
00004
00005     This is a simple C++ sequential OpenCL SYCL C++ header file to
00006     experiment with the OpenCL CL provisional specification.
00007
00008     For more information about OpenCL SYCL:
00009     http://www.khronos.org/sycl/
00010
00011     For more information on this project and to access to the source of
00012     this file, look at https://github.com/Xilinx/triSYCL
00013
00014     The Doxygen version of the implementation itself is in
00015     http://Xilinx.github.io/triSYCL/Doxygen/triSYCL/html and
00016     http://Xilinx.github.io/triSYCL/Doxygen/triSYCL/triSYCL-implementation-refman.pdf
00017
00018
00019     Ronan at keryell dot FR
00020
00021     Copyright 2014--2015 Advanced Micro Devices, Inc.
00022
00023     Copyright 2015--2016 Xilinx, Inc.
00024
00025     This file is distributed under the University of Illinois Open Source
00026     License. See LICENSE.TXT for details.
00027 */
00028
00029
00030 /** Some global triSYCL configuration */
00031 #include "CL/sycl/detail/global_config.hpp"
00032 #include "CL/sycl/detail/default_classes.hpp"
00033
00034
00035 /* All the SYCL components, one per file */
00036 #include "CL/sycl/access.hpp"
00037 #include "CL/sycl/accessor.hpp"
00038 #include "CL/sycl/address_space.hpp"
00039 #include "CL/sycl/buffer.hpp"
00040 #include "CL/sycl/context.hpp"
00041 #include "CL/sycl/device.hpp"
00042 #include "CL/sycl/device_selector.hpp"
00043 #include "CL/sycl/error_handler.hpp"
00044 #include "CL/sycl/event.hpp"
00045 #include "CL/sycl/exception.hpp"
00046 #include "CL/sycl/group.hpp"
00047 #include "CL/sycl/handler.hpp"
00048 #include "CL/sycl/id.hpp"
00049 #include "CL/sycl/image.hpp"
00050 #include "CL/sycl/item.hpp"
00051 #include "CL/sycl/nd_item.hpp"
00052 #include "CL/sycl/nd_range.hpp"
00053 #include "CL/sycl/parallelism.hpp"
00054 #include "CL/sycl/pipe.hpp"
00055 #include "CL/sycl/pipe_reservation.hpp"
00056 #include "CL/sycl/platform.hpp"
00057 #include "CL/sycl/queue.hpp"
00058 #include "CL/sycl/range.hpp"
00059 #include "CL/sycl/static_pipe.hpp"
00060 #include "CL/sycl/vec.hpp"
00061
00062 // Some includes at the end to break some dependencies
00063 #include "CL/sycl/device_selector/detail/device_selector_tail.hpp"
00064 #include "CL/sycl/device/detail/device_tail.hpp"
00065
00066 /*
00067     # Some Emacs stuff:
00068     ### Local Variables:
00069     ### ispell-local-dictionary: "american"
00070     ### eval: (flyspell-prog-mode)
00071     ### End:
00072 */

```

11.3 include/CL/sycl/access.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`
- `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

```

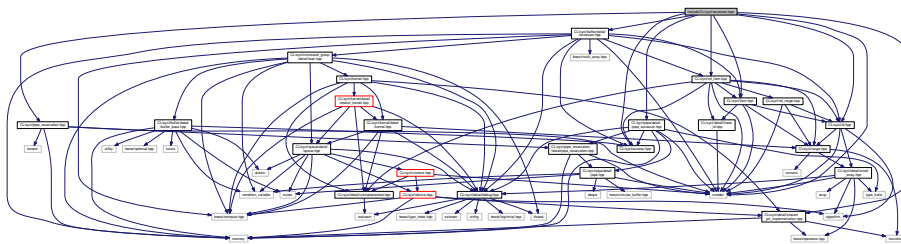
00001 #ifndef TRISYCL_SYCL_ACCESS_HPP
00002 #define TRISYCL_SYCL_ACCESS_HPP
00003
00004 /** \file The OpenCL SYCL access naming space
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 // SYCL dwells in the cl::sycl namespace
00013 namespace cl {
00014 namespace sycl {
00015
00016 /** \addtogroup data Data access and storage in SYCL
00017
00018     @{
00019 */
00020
00021 /** Describe the type of access by kernels.
00022
00023     \todo This values should be normalized to allow separate compilation
00024     with different implementations?
00025 */
00026 namespace access {
00027     /* By using "enum mode" here instead of "enum struct mode", we have for
00028     example "write" appearing both as cl::sycl::access::mode::write and
00029     cl::sycl::access::write, instead of only the last one. This seems
00030     more conform to the specification. */
00031
00032     /// This describes the type of the access mode to be used via accessor
00033     enum class mode {
00034         read = 42, /**< Read-only access. Insist on the fact that
00035             read_write != read + write */
00036         write, /**< Write-only access, but previous content *not* discarded
00037         read_write, /**< Read and write access
00038         discard_write, /**< Write-only access and previous content discarded
00039         discard_read_write, /**< Read and write access and previous
00040             content discarded*/
00041         atomic /**< Atomic access
00042     };
00043
00044
00045     /** The target enumeration describes the type of object to be accessed
00046     via the accessor
00047     */
00048     enum class target {
00049         global_buffer = 2014, /**< Just pick a random number...
00050         constant_buffer,
00051         local,
00052         image,
00053         host_buffer,
00054         host_image,
00055         image_array,
00056         pipe,
00057         blocking_pipe
00058     };
00059
00060
00061     /** Precise the address space a barrier needs to act on
00062     */
00063     enum class fence_space : char {
00064         local_space,
00065         global_space,
00066         global_and_local
00067     };
00068
00069 }
00070
00071 /// @} End the data Doxygen group
00072 }
00073 }
00074
00075 /**
00076     # Some Emacs stuff:
00077     ### Local Variables:
00078     ### ispell-local-dictionary: "american"
00079     ### eval: (flyspell-prog-mode)
00080     ### End:
00081 */
00082
00083 #endif // TRISYCL_SYCL_ACCESS_HPP

```

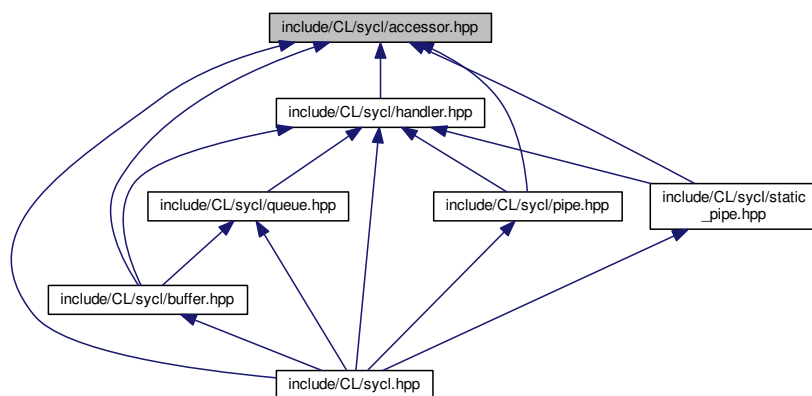

11.5 include/CL/sycl/accessor.hpp File Reference

```
#include <cstdint>
#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:



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

```

00001 #ifndef TRISYCL_SYCL_ACCESSOR_HPP
00002 #define TRISYCL_SYCL_ACCESSOR_HPP
00003
00004 /** \file The OpenCL SYCL accessor<>
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 #include <cstdint>
00013
00014 #include "CL/sycl/access.hpp"
00015 #include "CL/sycl/buffer/detail/accessor.hpp"
00016 #include "CL/sycl/detail/shared_ptr_implementation.hpp"
00017 #include "CL/sycl/id.hpp"
00018 #include "CL/sycl/item.hpp"
00019 #include "CL/sycl/nd_item.hpp"
00020 #include "CL/sycl/pipe_reservation.hpp"
00021 #include "CL/sycl/pipe/detail/pipe_accessor.hpp"
00022
00023 namespace cl {
00024 namespace sycl {
00025
00026 template <typename T, std::size_t Dimensions, typename Allocator>
00027 class buffer;
00028 template <typename T>
00029 class pipe;
00030 class handler;
00031
00032 /** \addtogroup data Data access and storage in SYCL
00033     @{
00034 */
00035
00036 /** The accessor abstracts the way buffer or pipe data are accessed
00037     inside a kernel in a multidimensional variable length array way.
00038
00039     \todo Implement it for images according so section 3.3.4.5
00040 */
00041 template <typename DataType,
00042           std::size_t Dimensions,
00043           access::mode AccessMode,
00044           access::target Target = access::target::global_buffer>
00045 class accessor :
00046     public detail::shared_ptr_implementation<accessor<DataType,
00047                                           Dimensions,
00048                                           AccessMode,
00049                                           Target>,
00050                                           detail::accessor<DataType,
00051                                           Dimensions,
00052                                           AccessMode,
00053                                           Target>> {
00054 public:
00055
00056     /// \todo in the specification: store the dimension for user request
00057     static constexpr auto dimensionality = Dimensions;
00058     using value_type = DataType;
00059     using reference = value_type&;
00060     using const_reference = const value_type&;

```

```

00061
00062 private:
00063
00064     using accessor_detail = detail::accessor<DataType,
00065                                             Dimensions,
00066                                             AccessMode,
00067                                             Target>;
00068
00069     // The type encapsulating the implementation
00070     using implementation_t =
00071         detail::shared_ptr_implementation<accessor<DataType,
00072                                             Dimensions,
00073                                             AccessMode,
00074                                             Target>,
00075                                             accessor_detail>;
00076
00077 public:
00078
00079     // Make the implementation member directly accessible in this class
00080     using implementation_t::implementation;
00081
00082     /** Construct a buffer accessor from a buffer using a command group
00083         handler object from the command group scope
00084
00085         Constructor only available for global_buffer or constant_buffer
00086         target.
00087
00088         access_target defines the form of access being obtained.
00089
00090         \todo Add template allocator type in all the accessor
00091         constructors in the specification or just use a more opaque
00092         Buffer type?
00093
00094         \todo fix specification where access mode should be target
00095         instead
00096     */
00097     template <typename Allocator>
00098     accessor(buffer<DataType, Dimensions, Allocator> &
00099             target_buffer,
00100             handler &command_group_handler) : implementation_t {
00101         new detail::accessor<DataType, Dimensions, AccessMode, Target>
00102         {
00103             target_buffer.implementation->implementation, command_group_handler }
00104     } {
00105         static_assert(Target == access::target::global_buffer
00106                     || Target == access::target::constant_buffer,
00107                     "access target should be global_buffer or constant_buffer "
00108                     "when a handler is used");
00109     }
00110
00111     /** Construct a buffer accessor from a buffer using a command group
00112         handler object from the command group scope
00113
00114         Constructor only available for host_buffer target.
00115
00116         access_target defines the form of access being obtained.
00117
00118         \todo add this lacking constructor to specification
00119     */
00120     template <typename Allocator>
00121     accessor(buffer<DataType, Dimensions, Allocator> &
00122             target_buffer)
00123         : implementation_t {
00124         new detail::accessor<DataType, Dimensions, AccessMode, Target>
00125         {
00126             target_buffer.implementation->implementation }
00127     } {
00128         static_assert(Target == access::target::host_buffer,
00129                     "without a handler, access target should be host_buffer");
00130     }
00131
00132     /** Construct a buffer accessor from a buffer given a specific range for
00133         access permissions and an offset that provides the starting point
00134         for the access range using a command group handler object from the
00135         command group scope
00136
00137         This accessor limits the processing of the buffer to the [offset,
00138         offset+range[ for every dimension. Any other parts of the buffer
00139         will be unaffected.
00140
00141         Constructor only available for access modes global_buffer,
00142         host_buffer or constant_buffer (see Table 3.25). access_target
00143         defines the form of access being obtained (see Table 3.26).
00144
00145         This accessor is recommended for discard-write and discard read

```

```

00144         write access modes, when the unaffected parts of the processing
00145         should be retained.
00146     */
00147     template <typename Allocator>
00148     accessor(buffer<DataType, Dimensions, Allocator> &
00149 target_buffer,
00149         handler &command_group_handler,
00150         range<Dimensions> offset,
00151         range<Dimensions> range) {
00152         detail::unimplemented();
00153     }
00154
00155
00156     /** Construct an accessor of dimensions Dimensions with elements of type
00157     DataType using the passed range to specify the size in each
00158     dimension
00159
00160     It needs as a parameter a command group handler object from the
00161     command group scope. Constructor only available if AccessMode is
00162     local, see Table 3.25.
00163     */
00164     accessor(range<Dimensions> allocation_size,
00165         handler &command_group_handler) {
00166         detail::unimplemented();
00167     }
00168
00169
00170     /** Use the accessor with integers à la [][][]
00171
00172     Use array_view_type::reference instead of auto& because it does not
00173     work in some dimensions.
00174     */
00175     typename accessor_detail::reference operator[](std::size_t index) {
00176         return (*implementation)[index];
00177     }
00178
00179
00180     /** Use the accessor with integers à la [][][]
00181
00182     Use array_view_type::reference instead of auto& because it does not
00183     work in some dimensions.
00184     */
00185     typename accessor_detail::reference operator[](std::size_t index)
00186     const {
00187         return (*implementation)[index];
00188     }
00189
00190     /// To use the accessor with [id<>]
00191     auto &operator[](id<dimensionality> index) {
00192         return (*implementation)[index];
00193     }
00194
00195
00196     /// To use the accessor with [id<>]
00197     auto &operator[](id<dimensionality> index) const {
00198         return (*implementation)[index];
00199     }
00200
00201
00202     /// To use an accessor with [item<>]
00203     auto &operator[](item<dimensionality> index) {
00204         return (*this)[index.get()];
00205     }
00206
00207
00208     /// To use an accessor with [item<>]
00209     auto &operator[](item<dimensionality> index) const {
00210         return (*this)[index.get()];
00211     }
00212
00213
00214     /** To use an accessor with an [nd_item<>]
00215
00216     \todo Add in the specification because used by HPC-GPU slide 22
00217     */
00218     auto &operator[](nd_item<dimensionality> index) {
00219         return (*this)[index.get_global()];
00220     }
00221
00222
00223     /** To use an accessor with an [nd_item<>]
00224
00225     \todo Add in the specification because used by HPC-GPU slide 22
00226     */
00226     auto &operator[](nd_item<dimensionality> index) const {
00227         return (*this)[index.get_global()];
00228     }

```

```

00229
00230
00231     /** Get the first element of the accessor
00232
00233         Useful with an accessor on a scalar for example.
00234
00235         \todo Add in the specification
00236     */
00237     typename accessor_detail::reference operator*() {
00238         return **implementation;
00239     }
00240
00241
00242     /** Get the first element of the accessor
00243
00244         Useful with an accessor on a scalar for example.
00245
00246         \todo Add in the specification?
00247
00248         \todo Add the concept of 0-dim buffer and accessor for scalar
00249         and use an implicit conversion to value_type reference to access
00250         the value with the accessor?
00251     */
00252     typename accessor_detail::reference operator*() const {
00253         return **implementation;
00254     }
00255
00256     /** Forward all the iterator functions to the implementation
00257
00258         \todo Add these functions to the specification
00259
00260         \todo The fact that the lambda capture make a const copy of the
00261         accessor is not yet elegantly managed... The issue is that
00262         begin()/end() dispatch is made according to the accessor
00263         constness and not from the array member constness...
00264
00265         \todo try to solve it by using some enable_if on array
00266         constness?
00267
00268         \todo The issue is that the end may not be known if it is
00269         implemented by a raw OpenCL cl_mem... So only provide on the
00270         device the iterators related to the start? Actually the accessor
00271         needs to know a part of the shape to have the multidimensional
00272         addressing. So this only require a size_t more...
00273
00274         \todo Factor out these in a template helper
00275     */
00276
00277
00278     // iterator begin() { return array.begin(); }
00279     typename accessor_detail::iterator begin() const {
00280         return implementation->begin();
00281     }
00282
00283
00284     // iterator end() { return array.end(); }
00285     typename accessor_detail::iterator end() const {
00286         return implementation->end();
00287     }
00288
00289
00290     // const_iterator begin() const { return implementation->begin(); }
00291
00292
00293     // const_iterator end() const { return implementation->end(); }
00294
00295
00296     typename accessor_detail::const_iterator cbegin() const {
00297         return implementation->cbegin();
00298     }
00299
00300
00301     typename accessor_detail::const_iterator cend() const {
00302         return implementation->cend();
00303     }
00304
00305
00306     typename accessor_detail::reverse_iterator
00307     rbegin() const {
00308         return implementation->rbegin();
00309     };
00310
00311     typename accessor_detail::reverse_iterator
00312     rend() const {
00313         return implementation->rend();
00314     }

```

```

00314
00315
00316 // const_reverse_iterator rbegin() const { return array.rbegin(); }
00317
00318
00319 // const_reverse_iterator rend() const { return array.rend(); }
00320
00321
00322 typename accessor_detail::const_reverse_iterator
00323 crbegin() const {
00324     return implementation->rbegin();
00325 }
00326
00327 typename accessor_detail::const_reverse_iterator
00328 crend() const {
00329     return implementation->rend();
00330 }
00331 };
00332
00333
00334 /** The pipe accessor abstracts the way pipe data are accessed inside
00335     a kernel
00336
00337     A specialization for an non-blocking pipe
00338 */
00339 template <typename DataType,
00340           access::mode AccessMode>
00341 class accessor<DataType, 1, AccessMode, access::target::pipe> :
00342     public detail::pipe_accessor<DataType, AccessMode, access::target::pipe> {
00343 public:
00344
00345     using accessor_detail =
00346         detail::pipe_accessor<DataType, AccessMode, access::target::pipe>
00347 ;
00348     using accessor_detail::accessor_detail;
00349
00350     /** Construct a pipe accessor from a pipe using a command group
00351         handler object from the command group scope
00352
00353         access_target defines the form of access being obtained.
00354     */
00355     accessor(pipe<DataType> &p, handler &command_group_handler)
00356         : accessor_detail { p.implementation, command_group_handler } { }
00357
00358     /// Make a reservation inside the pipe
00359     pipe_reservation<accessor> reserve(std::size_t size) const {
00360         return accessor_detail::reserve(size);
00361     }
00362
00363
00364     /// Get the underlying pipe implementation
00365     auto &get_pipe_detail() {
00366         return accessor_detail::get_pipe_detail();
00367     }
00368
00369 };
00370
00371
00372 /** The pipe accessor abstracts the way pipe data are accessed inside
00373     a kernel
00374
00375     A specialization for a blocking pipe
00376 */
00377 template <typename DataType,
00378           access::mode AccessMode>
00379 class accessor<DataType, 1, AccessMode, access::target::blocking_pipe> :
00380     public detail::pipe_accessor<DataType, AccessMode, access::target::blocking_pipe>
00381 {
00382 public:
00383
00384     using accessor_detail =
00385         detail::pipe_accessor<DataType, AccessMode, access::target::blocking_pipe>
00386 ;
00387     using accessor_detail::accessor_detail;
00388
00389     /** Construct a pipe accessor from a pipe using a command group
00390         handler object from the command group scope
00391
00392         access_target defines the form of access being obtained.
00393     */
00393     accessor(pipe<DataType> &p, handler &command_group_handler)
00394         : accessor_detail { p.implementation, command_group_handler } { }
00395
00396

```

```

00396
00397     /// Make a reservation inside the pipe
00398     pipe_reservation<accessor> reserve(std::size_t size) const {
00399         return accessor_detail::reserve(size);
00400     }
00401
00402
00403     /// Get the underlying pipe implementation
00404     auto &get_pipe_detail() {
00405         return accessor_detail::get_pipe_detail();
00406     }
00407
00408 };
00409
00410
00411 /** Top-level function to break circular dependencies on the the types
00412     to get the pipe implementation */
00413 template <typename Accessor>
00414 static inline auto &get_pipe_detail(Accessor &a) {
00415     return a.get_pipe_detail();
00416 }
00417
00418 /// @} End the data Doxygen group
00419
00420 }
00421 }
00422
00423 /*
00424     # Some Emacs stuff:
00425     ### Local Variables:
00426     ### ispell-local-dictionary: "american"
00427     ### eval: (flyspell-prog-mode)
00428     ### End:
00429 */
00430
00431 #endif // TRISYCL_SYCL_ACCESSOR_HPP

```

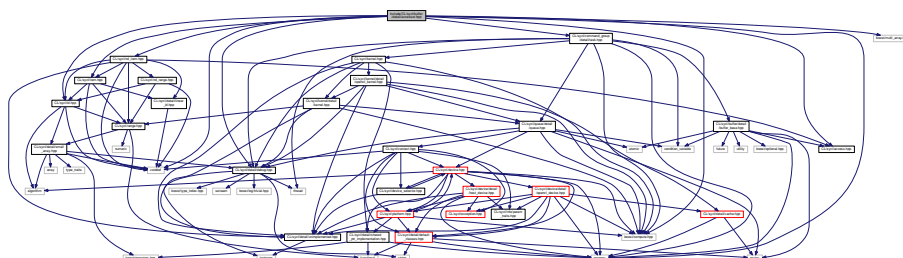
11.7 include/CL/sycl/buffer/detail/accessor.hpp File Reference

```

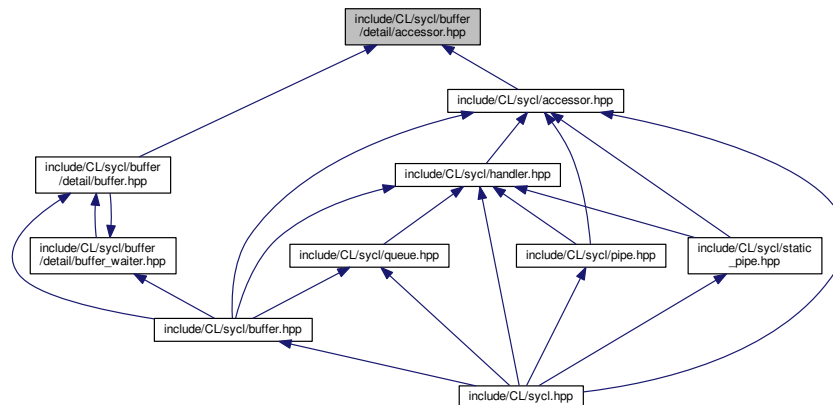
#include <cstdint>
#include <memory>
#include <boost/compute.hpp>
#include <boost/multi_array.hpp>
#include "CL/sycl/access.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"

```

Include dependency graph for accessor.hpp:



This graph shows which files directly or indirectly include this file:



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

```

00001 #ifndef TRISYCL_SYCL_ACCESSOR_DETAIL_ACCESSOR_HPP
00002 #define TRISYCL_SYCL_ACCESSOR_DETAIL_ACCESSOR_HPP
00003
00004 /** \file The OpenCL SYCL buffer accessor<> detail behind the scene
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 #include <cstdint>
00013 #include <memory>
00014
00015 #ifdef TRISYCL_OPENCL
00016 #include <boost/compute.hpp>
00017 #endif
00018 #include <boost/multi_array.hpp>
00019
00020 #include "CL/sycl/access.hpp"
00021 #include "CL/sycl/command_group/detail/task.hpp"
00022 #include "CL/sycl/detail/debug.hpp"

```



```

00023 #include "CL/sycl/id.hpp"
00024 #include "CL/sycl/item.hpp"
00025 #include "CL/sycl/nd_item.hpp"
00026
00027 namespace cl {
00028 namespace sycl {
00029
00030 class handler;
00031
00032 namespace detail {
00033
00034 // Forward declaration of detail::buffer for use in accessor
00035 template <typename T, std::size_t Dimensions> class buffer;
00036
00037 /** \addtogroup data Data access and storage in SYCL
00038     @{
00039     */
00040
00041 /** The buffer accessor abstracts the way buffer data are accessed
00042     inside a kernel in a multidimensional variable length array way.
00043
00044     This implementation relies on boost::multi_array to provide this
00045     nice syntax and behaviour.
00046
00047     Right now the aim of this class is just to access to the buffer in
00048     a read-write mode, even if capturing the multi_array_ref from a
00049     lambda make it const (since in examples we have lambda with [=]
00050     without mutable lambda).
00051
00052     \todo Use the access::mode
00053     */
00054 template <typename T,
00055           std::size_t Dimensions,
00056           access::mode Mode,
00057           access::target Target /* = access::global_buffer */>
00058 class accessor : public detail::debug<accessor<T,
00059           Dimensions,
00060           Mode,
00061           Target>> {
00062
00063     /** Keep a reference to the accessed buffer
00064
00065         Beware that it owns the buffer, which means that the accessor
00066         has to be destroyed to release the buffer and potentially
00067         unblock a kernel at the end of its execution
00068     */
00069     std::shared_ptr<detail::buffer<T, Dimensions>> buf;
00070
00071     /// The implementation is a multi_array_ref wrapper
00072     using array_view_type = boost::multi_array_ref<T, Dimensions>;
00073
00074     // The same type but writable
00075     using writable_array_view_type =
00076         typename std::remove_const<array_view_type>::type;
00077
00078     /** The way the buffer is really accessed
00079
00080         Use a mutable member because the accessor needs to be captured
00081         by value in the lambda which is then read-only. This is to avoid
00082         the user to use mutable lambda or have a lot of const_cast as
00083         previously done in this implementation
00084     */
00085     mutable array_view_type array;
00086
00087     /// The task where the accessor is used in
00088     std::shared_ptr<detail::task> task;
00089
00090 #ifdef TRISYCL_OPENCL
00091     /** The OpenCL buffer used by an OpenCL accessor
00092         boost::optional<boost::compute::buffer> cl_buf;
00093     #endif
00094
00095 public:
00096
00097     /** \todo in the specification: store the dimension for user request
00098
00099         \todo Use another name, such as from C++17 committee discussions.
00100     */
00101     static constexpr auto dimensionality = Dimensions;
00102
00103     /** \todo in the specification: store the types for user request as STL
00104         or C++AMP */
00105     using value_type = T;
00106     using element = T;
00107     using reference = typename array_view_type::reference;
00108     using const_reference = typename array_view_type::const_reference;
00109
00110     /** Inherit the iterator types from the implementation

```

```

00110
00111     \todo Add iterators to accessors in the specification
00112 */
00113 using iterator = typename array_view_type::iterator;
00114 using const_iterator = typename array_view_type::const_iterator;
00115 using reverse_iterator = typename array_view_type::reverse_iterator;
00116 using const_reverse_iterator =
00117     typename array_view_type::const_reverse_iterator;
00118
00119
00120 /** Construct a host accessor from an existing buffer
00121
00122     \todo fix the specification to rename target that shadows
00123     template parm
00124 */
00125 accessor(std::shared_ptr<detail::buffer<T, Dimensions>>
target_buffer) :
00126     buf { target_buffer }, array { target_buffer->access } {
00127     TRISYCL_DUMP_T("Create a host accessor write = " <<
is_write_access());
00128     static_assert(Target == access::target::host_buffer,
00129         "without a handler, access target should be host_buffer");
00130     /* The host needs to wait for all the producers of the buffer to
00131        have finished */
00132     buf->wait();
00133 }
00134
00135
00136 /** Construct a device accessor from an existing buffer
00137
00138     \todo fix the specification to rename target that shadows
00139     template parm
00140 */
00141 accessor(std::shared_ptr<detail::buffer<T, Dimensions>>
target_buffer,
00142     handler &command_group_handler) :
00143     buf { target_buffer }, array { target_buffer->access } {
00144     TRISYCL_DUMP_T("Create a kernel accessor write = " <<
is_write_access());
00145     static_assert(Target == access::target::global_buffer
00146         || Target == access::target::constant_buffer,
00147         "access target should be global_buffer or constant_buffer "
00148         "when a handler is used");
00149     // Register the buffer to the task dependencies
00150     task = buffer_add_to_task(buf, &command_group_handler,
is_write_access());
00151 }
00152
00153
00154 /** Returns the size of the underlying buffer in number of elements
00155
00156     \todo It is incompatible with buffer get_size() in the spec
00157 */
00158 std::size_t get_size() const {
00159     return array.num_elements();
00160 }
00161
00162
00163 /** Use the accessor with integers à la [][][]
00164
00165     Use array_view_type::reference instead of auto& because it does not
00166     work in some dimensions.
00167 */
00168 reference operator[](std::size_t index) {
00169     return array[index];
00170 }
00171
00172
00173 /** Use the accessor with integers à la [][][]
00174
00175     Use array_view_type::reference instead of auto& because it does not
00176     work in some dimensions.
00177 */
00178 reference operator[](std::size_t index) const {
00179     return array[index];
00180 }
00181
00182
00183 /// To use the accessor with [id<>]
00184 auto &operator[](id<dimensionality> index) {
00185     return array(index);
00186 }
00187
00188
00189 /// To use the accessor with [id<>]
00190 auto &operator[](id<dimensionality> index) const {
00191     return array(index);

```

```

00192     }
00193
00194
00195     /// To use an accessor with [item<>]
00196     auto &operator[](item<dimensionality> index) {
00197         return (*this)[index.get()];
00198     }
00199
00200
00201     /// To use an accessor with [item<>]
00202     auto &operator[](item<dimensionality> index) const {
00203         return (*this)[index.get()];
00204     }
00205
00206
00207     /** To use an accessor with an [nd_item<>]
00208
00209         \todo Add in the specification because used by HPC-GPU slide 22
00210     */
00211     auto &operator[](nd_item<dimensionality> index) {
00212         return (*this)[index.get_global()];
00213     }
00214
00215     /** To use an accessor with an [nd_item<>]
00216
00217         \todo Add in the specification because used by HPC-GPU slide 22
00218     */
00219     auto &operator[](nd_item<dimensionality> index) const {
00220         return (*this)[index.get_global()];
00221     }
00222
00223
00224     /** Get the first element of the accessor
00225
00226         Useful with an accessor on a scalar for example.
00227
00228         \todo Add in the specification
00229     */
00230     reference operator*() {
00231         return *array.data();
00232     }
00233
00234
00235     /** Get the first element of the accessor
00236
00237         Useful with an accessor on a scalar for example.
00238
00239         \todo Add in the specification?
00240
00241         \todo Add the concept of 0-dim buffer and accessor for scalar
00242         and use an implicit conversion to value_type reference to access
00243         the value with the accessor?
00244     */
00245     reference operator*() const {
00246         return *array.data();
00247     }
00248
00249
00250     /// Get the buffer used to create the accessor
00251     detail::buffer<T, Dimensions> &get_buffer() {
00252         return *buf;
00253     }
00254
00255
00256     /** Test if the accessor has a read access right
00257
00258         \todo Strangely, it is not really constexpr because it is not a
00259         static method...
00260
00261         \todo to move in the access::mode enum class and add to the
00262         specification ?
00263     */
00264     constexpr bool is_read_access() const {
00265         return Mode == access::mode::read
00266             || Mode == access::mode::read_write
00267             || Mode == access::mode::discard_read_write;
00268     }
00269
00270
00271     /** Test if the accessor has a write access right
00272
00273         \todo Strangely, it is not really constexpr because it is not a
00274         static method...
00275
00276         \todo to move in the access::mode enum class and add to the
00277         specification ?
00278     */

```

```

00279 constexpr bool is_write_access() const {
00280     return Mode == access::mode::write
00281         || Mode == access::mode::read_write
00282         || Mode == access::mode::discard_write
00283         || Mode == access::mode::discard_read_write;
00284 }
00285
00286
00287 /** Forward all the iterator functions to the implementation
00288
00289     \todo Add these functions to the specification
00290
00291     \todo The fact that the lambda capture make a const copy of the
00292     accessor is not yet elegantly managed... The issue is that
00293     begin()/end() dispatch is made according to the accessor
00294     constness and not from the array member constness...
00295
00296     \todo try to solve it by using some enable_if on array
00297     constness?
00298
00299     \todo The issue is that the end may not be known if it is
00300     implemented by a raw OpenCL cl_mem... So only provide on the
00301     device the iterators related to the start? Actually the accessor
00302     needs to know a part of the shape to have the multidimensional
00303     addressing. So this only require a size_t more...
00304
00305     \todo Factor out these in a template helper
00306
00307     \todo Do we need this in detail::accessor too or only in accessor?
00308 */
00309
00310 // iterator begin() { return array.begin(); }
00312 iterator begin() const {
00313     return const_cast<writable_array_view_type >(array).
begin();
00314 }
00315
00316 // iterator end() { return array.end(); }
00318 iterator end() const {
00319     return const_cast<writable_array_view_type >(array).
end();
00320 }
00321
00322 // const_iterator begin() const { return array.begin(); }
00323
00324 // const_iterator end() const { return array.end(); }
00325
00326 const_iterator cbegin() const { return array.begin(); }
00327
00328 const_iterator cend() const { return array.end(); }
00329
00330 // reverse_iterator rbegin() { return array.rbegin(); }
00332 reverse_iterator rbegin() const {
00333     return const_cast<writable_array_view_type >(array).
rbegin();
00334 }
00335
00336 // reverse_iterator rend() { return array.rend(); }
00338 reverse_iterator rend() const {
00339     return const_cast<writable_array_view_type >(array).
rend();
00340 }
00341
00342 // const_reverse_iterator rbegin() const { return array.rbegin(); }
00343
00344 // const_reverse_iterator rend() const { return array.rend(); }
00345
00346 const_reverse_iterator crbegin() const { return array.rbegin(); }
00347
00348 const_reverse_iterator crend() const { return array.rend(); }
00349
00350 private:
00351
00352 // The following function are used from handler
00353 friend handler;

```

```

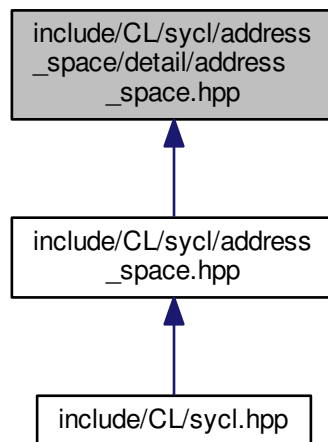
00362
00363 #ifndef TRISYCL_OPENCL
00364     /// Get the boost::compute::buffer or throw if unset
00365     auto get_cl_buffer() const {
00366         // This throws if not set
00367         return cl_buf.value();
00368     }
00369
00370
00371     /** Lazily associate a CL buffer to the SYCL buffer and copy data in
00372         if required
00373
00374         \todo Move this into the buffer with queue/device-based caching
00375     */
00376     void copy_in_cl_buffer() {
00377         // This should be a constexpr
00378         cl_mem_flags flags = is_read_access() && is_write_access() ?
00379             CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR
00380             : is_read_access() ? CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR
00381             : CL_MEM_WRITE_ONLY;
00382
00383         /** Create the OpenCL buffer and copy in data from the host if in
00384             read mode */
00385         cl_buf = { task->get_queue()->get_boost_compute().get_context(),
00386             get_size()*sizeof(value_type),
00387             flags,
00388             is_read_access() ? array.data() : 0 };
00389     }
00390
00391
00392     /** Copy back the CL buffer to the SYCL if required
00393
00394         \todo Move this into the buffer with queue/device-based caching
00395     */
00396     void copy_back_cl_buffer() {
00397         // \todo Use if constexpr in C++17
00398         if (is_write_access())
00399             task->get_queue()->get_boost_compute()
00400                 .enqueue_read_buffer(get_cl_buffer(),
00401                     0 /*< Offset */,
00402                     get_size()*sizeof(value_type),
00403                     array.data());
00404     }
00405 #endif
00406
00407 };
00408
00409 /// @} End the data Doxygen group
00410
00411 }
00412 }
00413 }
00414
00415 /**
00416     # Some Emacs stuff:
00417     ### Local Variables:
00418     ###  ispell-local-dictionary: "american"
00419     ###  eval: (flyspell-prog-mode)
00420     ###  End:
00421 */
00422
00423 #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

```
00001 #ifndef TRISYCL_SYCL_ADDRESS_SPACES_DETAIL_ADDRESS_SPACES_HPP
00002 #define TRISYCL_SYCL_ADDRESS_SPACES_DETAIL_ADDRESS_SPACES_HPP
00003
00004 /** \file
00005
00006     Implement OpenCL address spaces in SYCL with C++-style.
00007
00008     Ronan at Keryell point FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 namespace cl {
00015 namespace sycl {
00016 namespace detail {
00017
00018     /** \addtogroup address_spaces
00019         @{
00020     */
00021
00022     /** Generate a type with some real OpenCL 2 attribute if we are on an
```

```

00023     OpenCL device
00024
00025     In the general case, do not add any OpenCL address space qualifier */
00026 template <typename T, address_space AS>
00027 struct opengl_type {
00028     using type = T;
00029 };
00030
00031 /// Add an attribute for __constant address space
00032 template <typename T>
00033 struct opengl_type<T, constant_address_space> {
00034     using type = T
00035 #ifdef __SYCL_DEVICE_ONLY__
00036     /* Put the address space qualifier after the type so that we can
00037        construct pointer type with qualifier */
00038     __constant
00039 #endif
00040     ;
00041 };
00042
00043 /// Add an attribute for __generic address space
00044 template <typename T>
00045 struct opengl_type<T, generic_address_space> {
00046     using type = T
00047 #ifdef __SYCL_DEVICE_ONLY__
00048     /* Put the address space qualifier after the type so that we can
00049        construct pointer type with qualifier */
00050     __generic
00051 #endif
00052     ;
00053 };
00054
00055 /// Add an attribute for __global address space
00056 template <typename T>
00057 struct opengl_type<T, global_address_space> {
00058     using type = T
00059 #ifdef __SYCL_DEVICE_ONLY__
00060     /* Put the address space qualifier after the type so that we can
00061        construct pointer type with qualifier */
00062     __global
00063 #endif
00064     ;
00065 };
00066
00067 /// Add an attribute for __local address space
00068 template <typename T>
00069 struct opengl_type<T, local_address_space> {
00070     using type = T
00071 #ifdef __SYCL_DEVICE_ONLY__
00072     /* Put the address space qualifier after the type so that we can
00073        construct pointer type with qualifier */
00074     __local
00075 #endif
00076     ;
00077 };
00078
00079 /// Add an attribute for __private address space
00080 template <typename T>
00081 struct opengl_type<T, private_address_space> {
00082     using type = T
00083 #ifdef __SYCL_DEVICE_ONLY__
00084     /* Put the address space qualifier after the type so that we can
00085        construct pointer type with qualifier */
00086     __private
00087 #endif
00088     ;
00089 };
00090
00091
00092 /* Forward declare some classes to allow some recursion in conversion
00093    operators */
00094 template <typename SomeType, address_space SomeAS>
00095 struct address_space_array;
00096
00097 template <typename SomeType, address_space SomeAS>
00098 struct address_space_fundamental;
00099
00100 template <typename SomeType, address_space SomeAS>
00101 struct address_space_object;
00102
00103 template <typename SomeType, address_space SomeAS>
00104 struct address_space_ptr;
00105
00106 /** Dispatch the address space implementation according to the requested type
00107
00108     \param T is the type of the object to be created
00109

```



```

00110     \param AS is the address space to place the object into or to point to
00111     in the case of a pointer type
00112 */
00113 template <typename T, address_space AS>
00114 using addr_space =
00115     typename std::conditional<std::is_pointer<T>::value,
00116                             address_space_ptr<T, AS>,
00117     typename std::conditional<std::is_class<T>::value,
00118                             address_space_object<T, AS>,
00119     typename std::conditional<std::is_array<T>::value,
00120                             address_space_array<T, AS>,
00121                             address_space_fundamental<T, AS>
00122 >::type>::type>::type;
00123
00124
00125 /** Implementation of the base infrastructure to wrap something in an
00126     OpenCL address space
00127
00128     \param T is the type of the basic stuff to be created
00129
00130     \param AS is the address space to place the object into
00131
00132     \todo Verify/improve to deal with const/volatile?
00133 */
00134 template <typename T, address_space AS>
00135 struct address_space_base {
00136     /** Store the base type of the object
00137
00138         \todo Add to the specification
00139     */
00140     using type = T;
00141
00142     /** Store the base type of the object with OpenCL address space modifier
00143
00144         \todo Add to the specification
00145     */
00146     using opengl_type = typename opengl_type<T, AS>::type;
00147
00148     /** Set the address_space identifier that can be queried to know the
00149         pointer type */
00150     static auto constexpr address_space = AS;
00151 };
00152
00153
00154
00155 /** Implementation of a variable with an OpenCL address space
00156
00157     \param T is the type of the basic object to be created
00158
00159     \param AS is the address space to place the object into
00160 */
00161 template <typename T, address_space AS>
00162 struct address_space_variable : public address_space_base<T, AS> {
00163     /** Store the base type of the object with OpenCL address space modifier
00164
00165         \todo Add to the specification
00166     */
00167     using opengl_type = typename opengl_type<T, AS>::type;
00168
00169     /// Keep track of the base class as a short-cut
00170     using super = address_space_base<T, AS>;
00171
00172 protected:
00173
00174     /* C++11 helps a lot to be able to have the same constructors as the
00175         parent class here
00176
00177         \todo Add this to the list of required C++11 features needed for SYCL
00178     */
00179     opengl_type variable;
00180
00181 public:
00182
00183     /** Allow to create an address space version of an object or to convert
00184         one to be used by the classes inheriting by this one because it is
00185         not possible to directly initialize a base class member in C++ */
00186     address_space_variable(const T & v) : variable(v) { }
00187
00188
00189     /// Put back the default constructors canceled by the previous definition
00190     address_space_variable() = default;
00191
00192
00193     /** Conversion operator to allow a address_space_object<T> to be used
00194         as a T so that all the methods of a T and the built-in operators for
00195         T can be used on a address_space_object<T> too.
00196     */

```

```

00197         Use opcnl_type so that if we take the address of it, the address
00198         space is kept.
00199     */
00200     operator opcnl_type & () { return variable; }
00201
00202     /// Return the address of the value to implement pointers
00203     opcnl_type * get_address() { return &variable; }
00204
00205 };
00206
00207
00208 /** Implementation of a fundamental type with an OpenCL address space
00209
00210     \param T is the type of the basic object to be created
00211
00212     \param AS is the address space to place the object into
00213
00214     \todo Verify/improve to deal with const/volatile?
00215 */
00216 template <typename T, address_space AS>
00217 struct address_space_fundamental : public
00218     address_space_variable<T, AS> {
00219     /// Keep track of the base class as a short-cut
00220     using super = address_space_variable<T, AS>;
00221
00222     /// Inherit from base class constructors
00223     using super::address_space_variable;
00224
00225     /** Also request for the default constructors that have been disabled by
00226         the declaration of another constructor
00227
00228         This ensures for example that we can write
00229         \code
00230             generic<float *> q;
00231         \endcode
00232         without initialization.
00233     */
00234     address_space_fundamental() = default;
00235
00236
00237     /** Allow for example assignment of a global<float> to a priv<double>
00238         for example
00239
00240         Since it needs 2 implicit conversions, it does not work with the
00241         conversion operators already define, so add 1 more explicit
00242         conversion here so that the remaining implicit conversion can be
00243         found by the compiler.
00244
00245         Strangely
00246         \code
00247             template <typename SomeType, address_space SomeAS>
00248             address_space_base(addr_space<SomeType, SomeAS>& v)
00249             : variable(SomeType(v)) { }
00250         \endcode
00251         cannot be used here because SomeType cannot be inferred. So use
00252         address_space_base<> instead
00253
00254         Need to think further about it...
00255     */
00256     template <typename SomeType, cl::sycl::address_space SomeAS>
00257     address_space_fundamental(
00258         address_space_fundamental<SomeType, SomeAS>& v)
00259     {
00260         /* Strangely I cannot have it working in the initializer instead, for
00261             some cases */
00262         super::variable = SomeType(v);
00263     }
00264 };
00265
00266
00267 /** Implementation for an OpenCL address space pointer
00268
00269     \param T is the pointer type
00270
00271     Note that if \a T is not a pointer type, it is an error.
00272
00273     All the address space pointers inherit from it, which makes trivial
00274     the implementation of cl::sycl::multi_ptr<T, AS>
00275 */
00276 template <typename T, address_space AS>
00277 struct address_space_ptr : public address_space_fundamental<T, AS>
00278 > {
00279     /// Verify that \a T is really a pointer
00280     static_assert(std::is_pointer<T>::value,
00281         "T must be a pointer type");

```

```

00281
00282     /// Keep track of the base class as a short-cut
00283     using super = address_space_fundamental<T, AS>;
00284
00285     /// Inherit from base class constructors
00286     using super::address_space_fundamental;
00287
00288     /** Allow initialization of a pointer type from the address of an
00289         element with the same type and address space
00290     */
00291     address_space_ptr(address_space_fundamental<typename
std::pointer_traits<T>::element_type, AS> *p)
00292         : address_space_fundamental<T, AS> { p->get_address() } {}
00293
00294     /// Put back the default constructors canceled by the previous definition
00295     address_space_ptr() = default;
00296 };
00297
00298
00299 /** Implementation of an array variable with an OpenCL address space
00300
00301     \param T is the type of the basic object to be created
00302
00303     \param AS is the address space to place the object into
00304 */
00305 template <typename T, address_space AS>
00306 struct address_space_array : public address_space_variable<T, AS>
00307 {
00308     /// Keep track of the base class as a short-cut
00309     using super = address_space_variable<T, AS>;
00310
00311     /// Inherit from base class constructors
00312     using super::address_space_variable;
00313
00314     /** Allow to create an address space array from an array
00315     */
00316     address_space_array(const T &array) {
00317         std::copy(std::begin(array), std::end(array), std::begin(super::variable));
00318     };
00319
00320
00321     /** Allow to create an address space array from an initializer list
00322
00323         \todo Extend to more than 1 dimension
00324     */
00325     address_space_array(std::initializer_list<std::remove_extent_t<T>> list) {
00326         std::copy(std::begin(list), std::end(list), std::begin(super::variable));
00327     };
00328 };
00329
00330
00331 /** Implementation of an object type with an OpenCL address space
00332
00333     \param T is the type of the basic object to be created
00334
00335     \param AS is the address space to place the object into
00336
00337     The class implementation is just inheriting of T so that all methods
00338     and non-member operators on T work also on address_space_object<T>
00339
00340     \todo Verify/improve to deal with const/volatile?
00341
00342     \todo what about T having some final methods?
00343 */
00344 template <typename T, address_space AS>
00345 struct address_space_object : public opcnl_type<T, AS>::type,
00346                             public address_space_base<T, AS> {
00347     /** Store the base type of the object with OpenCL address space modifier
00348
00349         \todo Add to the specification
00350     */
00351     using opcnl_type = typename opcnl_type<T, AS>::type;
00352
00353     /* C++11 helps a lot to be able to have the same constructors as the
00354         parent class here but with an OpenCL address space
00355
00356         \todo Add this to the list of required C++11 features needed for SYCL
00357     */
00358     using opcnl_type::opcnl_type;
00359
00360
00361     /** Allow to create an address space version of an object or to
00362         convert one */
00363     address_space_object(T && v) : opcnl_type(v) { }
00364
00365     /** Conversion operator to allow a address_space_object<T> to be used

```

```

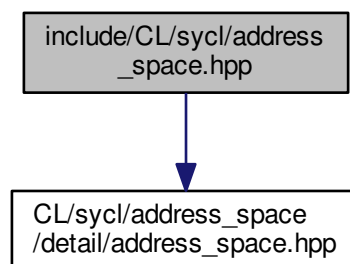
00366         as a T so that all the methods of a T and the built-in operators for
00367         T can be used on a address_space_object<T> too.
00368
00369         Use opcnl_type so that if we take the address of it, the address
00370         space is kept. */
00371     operator opcnl_type & () { return *this; }
00372
00373 };
00374
00375 /// @} End the address_spaces Doxygen group
00376
00377 }
00378 }
00379 }
00380
00381 /*
00382     # Some Emacs stuff:
00383     ### Local Variables:
00384     ### ispell-local-dictionary: "american"
00385     ### eval: (flyspell-prog-mode)
00386     ### End:
00387 */
00388
00389 #endif // TRISYCL_SYCL_ADDRESS_SPACES_DETAIL_ADDRESS_SPACES_HPP

```

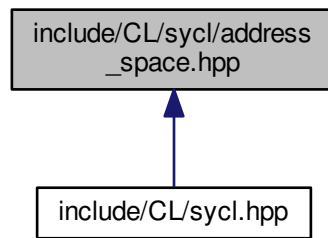
11.11 include/CL/sycl/address_space.hpp File Reference

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:
```



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

```
00001 #ifndef TRISYCL_SYCL_ADDRESS_SPACE_HPP
00002 #define TRISYCL_SYCL_ADDRESS_SPACE_HPP
00003
00004 /** \file
00005
00006     Implement OpenCL address spaces in SYCL with C++-style.
00007
00008     Note that in SYCL 1.2, only pointer types should be specified but
00009     in this implementation we generalize the concept to any type.
00010
00011     \todo Add the alias ..._ptr<T> = ...<T *>
00012
00013     Ronan at Keryell point FR
00014
00015     This file is distributed under the University of Illinois Open Source
00016     License. See LICENSE.TXT for details.
00017 */
00018
00019 namespace cl {
00020 namespace sycl {
00021
00022 /** \addtogroup address_spaces Dealing with OpenCL address spaces
00023     @{
00024 */
00025
00026 /** Enumerate the different OpenCL 2 address spaces */
00027 enum address_space {
00028     constant_address_space,
00029     generic_address_space,
00030     global_address_space,
00031     local_address_space,
00032     private_address_space,
00033 };
00034
00035 }
00036 }
00037 /// @} End the address_spaces Doxygen group
00038
00039
00040 #include "CL/sycl/address_space/detail/address_space.hpp"
00041
00042
00043 namespace cl {
00044 namespace sycl {
```

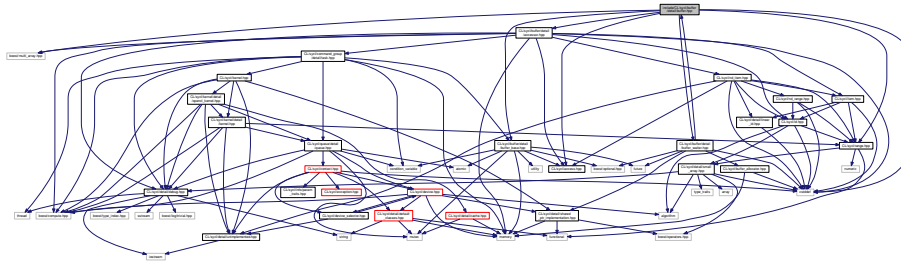
```

00045
00046 /** \addtogroup address_spaces
00047     @{
00048 */
00049
00050 /** Declare a variable to be in the OpenCL constant address space
00051     \param T is the type of the object
00052 */
00053 template <typename T>
00054 using constant = detail::addr_space<T, constant_address_space>
00055 ;
00056
00057
00058 /** Declare a variable to be in the OpenCL 2 generic address space
00059     \param T is the type of the object
00060 */
00061 template <typename T>
00062 using generic = detail::addr_space<T, generic_address_space>;
00063
00064
00065
00066 /** Declare a variable to be in the OpenCL global address space
00067     \param T is the type of the object
00068 */
00069 template <typename T>
00070 using global = detail::addr_space<T, global_address_space>
00071 ;
00072
00073
00074 /** Declare a variable to be in the OpenCL local address space
00075     \param T is the type of the object
00076 */
00077 template <typename T>
00078 using local = detail::addr_space<T, local_address_space>;
00079
00080
00081
00082 /** Declare a variable to be in the OpenCL private address space
00083     \param T is the type of the object
00084 */
00085 template <typename T>
00086 using priv = detail::addr_space<T, private_address_space>;
00087
00088
00089
00090 /** A pointer that can be statically associated to any address-space
00091     \param Pointer is the pointer type
00092     \param AS is the address space to point to
00093     Note that if \a Pointer is not a pointer type, it is an error.
00094 */
00095 template <typename Pointer, address_space AS>
00096 using multi_ptr = detail::address_space_ptr<Pointer, AS>;
00097
00098
00099
00100
00101
00102 /** Construct a cl::sycl::multi_ptr<> with the right type
00103     \param pointer is the address with its address space to point to
00104     \todo Implement the case with a plain pointer
00105 */
00106 template <typename T, address_space AS>
00107 multi_ptr<T, AS> make_multi(multi_ptr<T, AS> pointer) {
00108     return pointer;
00109 }
00110
00111
00112
00113
00114
00115 /// @} End the parallelism Doxygen group
00116
00117 /*
00118     # Some Emacs stuff:
00119     ### Local Variables:
00120     ### ispell-local-dictionary: "american"
00121     ### eval: (flyspell-prog-mode)
00122     ### End:
00123 */
00124
00125 #endif // TRISYCL_SYCL_ADDRESS_SPACE_HPP

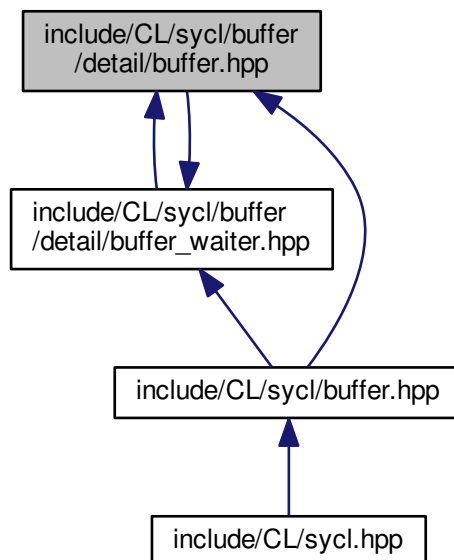
```

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:



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

```

00001 #ifndef TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_HPP
00002 #define TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_HPP
00003
00004 /** \file The OpenCL SYCL buffer<> detail implementation
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 #include <cstdint>
00013
00014 #include <boost/multi_array.hpp>
00015
00016 #include "CL/sycl/access.hpp"
00017 #include "CL/sycl/buffer/detail/accessor.hpp"
00018 #include "CL/sycl/buffer/detail/buffer_base.hpp"
00019 #include "CL/sycl/buffer/detail/buffer_waiter.hpp"
00020 #include "CL/sycl/range.hpp"
00021
00022 namespace cl {
00023 namespace sycl {
00024 namespace detail {
00025
00026 /** \addtogroup data Data access and storage in SYCL
00027     @{
00028 */
00029
00030 /** A SYCL buffer is a multidimensional variable length array (à la C99
00031     VLA or even Fortran before) that is used to store data to work on.
00032
00033     In the case we initialize it from a pointer, for now we just wrap the
00034     data with boost::multi_array_ref to provide the VLA semantics without
00035     any storage.
00036 */
00037 template <typename T,
00038           std::size_t Dimensions = 1>
00039 class buffer : public detail::buffer_base,
00040               public detail::debug<buffer<T, Dimensions>> {
00041 public:
00042
00043     // Extension to SYCL: provide pieces of STL container interface
00044     using element = T;
00045     using value_type = T;
00046
00047 private:
00048
00049     /** If some allocation is requested, it is managed by this multi_array
00050         to ease initialization from data */
00051     boost::multi_array<T, Dimensions> allocation;
00052
00053     // \todo Replace U and D somehow by T and Dimensions
00054     // To allow allocation access
00055     template <typename U,
00056               std::size_t D,
00057               access::mode Mode,

```

```

00058         access::target Target /* = access::global_buffer */>
00059     friend class detail::accessor;
00060
00061
00062     /** This is the multi-dimensional interface to the data that may point
00063         to either allocation in the case of storage managed by SYCL itself
00064         or to some other memory location in the case of host memory or
00065         storage<> abstraction use
00066     */
00067     boost::multi_array_ref<T, Dimensions> access;
00068
00069     /// The weak pointer to copy back data on buffer deletion
00070     weak_ptr_class<T> final_data;
00071
00072     /** The shared pointer in the case the buffer memory is shared with
00073         the host */
00074     shared_ptr_class<T> shared_data;
00075
00076     // Track if the buffer memory is provided as host memory
00077     bool host_write_back = false;
00078
00079 public:
00080
00081     /// Create a new read-write buffer of size \param r
00082     buffer(const range<Dimensions> &r) : buffer_base { false },
00083                                         allocation { r },
00084                                         access { allocation }
00085     {}
00086
00087     /** Create a new read-write buffer from \param host_data of size
00088         \param r without further allocation */
00089     buffer(T *host_data, const range<Dimensions> &r) :
00090     buffer_base { false },
00091                                         access { host_data, r },
00092                                         host_write_back { true }
00093     {}
00094
00095     /** Create a new read-only buffer from \param host_data of size \param r
00096         without further allocation
00097
00098         \todo Clarify the semantics in the spec. What happens if the
00099         host change the host_data after buffer creation?
00100     */
00101     buffer(const T *host_data, const range<Dimensions> &r) :
00102     /* \todo Need to solve this const buffer issue in a clean way
00103
00104         Just allocate memory? */
00105     buffer_base { true },
00106     access { const_cast<T *>(host_data), r }
00107     {}
00108
00109
00110     /** Create a new buffer with associated memory, using the data in
00111         host_data
00112
00113         The ownership of the host_data is shared between the runtime and the
00114         user. In order to enable both the user application and the SYCL
00115         runtime to use the same pointer, a cl::sycl::mutex_class is
00116         used.
00117     */
00118     buffer(shared_ptr_class<T> &host_data,
00119            const range<Dimensions> &r)
00120     : buffer_base { false },
00121     access { host_data.get(), r },
00122     shared_data { host_data }
00123     {}
00124
00125
00126     /// Create a new allocated 1D buffer from the given elements
00127     template <typename Iterator>
00128     buffer(Iterator start_iterator, Iterator end_iterator) :
00129     buffer_base { false },
00130     // The size of a multi_array is set at creation time
00131     allocation { boost::extents[std::distance(start_iterator, end_iterator)] },
00132     access { allocation }
00133     {
00134         /* Then assign allocation since this is the only multi_array
00135            method with this iterator interface */
00136         allocation.assign(start_iterator, end_iterator);
00137     }
00138
00139
00140     /** Create a new sub-buffer without allocation to have separate
00141         accessors later
00142     */
00143

```

```

00144     \todo To implement and deal with reference counting
00145     buffer(buffer<T, Dimensions> b,
00146            index<Dimensions> base_index,
00147            range<Dimensions> sub_range)
00148     */
00149
00150     /// \todo Allow CLHPP objects too?
00151     ///
00152     /*
00153     buffer(cl_mem mem_object,
00154            queue from_queue,
00155            event available_event)
00156     */
00157
00158     /** The buffer content may be copied back on destruction to some
00159         final location */
00160     ~buffer() {
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     }
00170
00171     // Use BOOST_DISABLE_ASSERTS at some time to disable range checking
00172
00173     /// Return an accessor of the required mode \param M
00174     /// \todo Remove if not used
00175     template <access::mode Mode,
00176              access::target Target = access::target::global_buffer
00177     >
00178     detail::accessor<T, Dimensions, Mode, Target>
00179     get_access() {
00180         return { *this };
00181     }
00182
00183     /** Return a range object representing the size of the buffer in
00184         terms of number of elements in each dimension as passed to the
00185         constructor
00186     */
00187     auto get_range() const {
00188         /* Interpret the shape which is a pointer to the first element as an
00189            array of Dimensions elements so that the range<Dimensions>
00190            constructor is happy with this collection
00191
00192            \todo Add also a constructor in range<> to accept a const
00193            std::size_t */
00194         return range<Dimensions> {
00195             *(const std::size_t (*)[Dimensions]) (allocation.shape())
00196         };
00197     }
00198
00199     /** Returns the total number of elements in the buffer
00200
00201         Equal to get_range()[0] * ... * get_range()[dimensions-1].
00202     */
00203     auto get_count() const {
00204         return allocation.num_elements();
00205     }
00206
00207     /** Returns the size of the buffer storage in bytes
00208
00209         Equal to get_count()*sizeof(T).
00210
00211         \todo rename to something else. In
00212         http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf
00213         it is named bytes() for example
00214     */
00215     size_t get_size() const {
00216         return get_count()*sizeof(T);
00217     }
00218
00219     /** Set the weak pointer to copy back data on buffer deletion
00220
00221         \todo Add a write kernel dependency on the buffer so the buffer
00222         destructor has to wait for the kernel execution if the buffer is
00223         also accessed through a write accessor
00224     */
00225     void set_final_data(weak_ptr_class<T> && finalData) {

```

```

00229     final_data = finalData;
00230 }
00231
00232 private:
00233
00234 /** Get a \c future to wait from inside the \c cl::sycl::buffer in
00235     case there is something to copy back to the host
00236
00237     \return A \c future in the \c optional if there is something to
00238     wait for, otherwise an empty \c optional
00239 */
00240 boost::optional<std::future<void>> get_destructor_future() {
00241     boost::optional<std::future<void>> f;
00242     /* If there is only 1 shared_ptr user of the buffer, this is the
00243        caller of this function, the \c buffer_waiter, so there is no
00244        need to get a \c future otherwise there will be a dead-lock if
00245        there is only 1 thread waiting for itself.
00246
00247        Since \c use_count() is applied to a \c shared_ptr just created
00248        for this purpose, it actually increase locally the count by 1,
00249        so check for 1 + 1 use count instead...
00250     */
00251     if (shared_from_this().use_count() > 2)
00252         // \todo Double check the specification and add unit tests
00253         if (host_write_back || !final_data.expired() || shared_data) {
00254             // Create a promise to wait for
00255             notify_buffer_destructor = std::promise<void> {};
00256             // And return the future to wait for it
00257             f = notify_buffer_destructor->get_future();
00258         }
00259     return f;
00260 }
00261
00262 // Allow buffer_waiter destructor to access get_destructor_future()
00263 // friend detail::buffer_waiter<T, Dimensions>::~buffer_waiter();
00264 /* \todo Work around to Clang bug
00265     https://llvm.org/bugs/show_bug.cgi?id=28873 cannot use destructor
00266     here */
00267 friend detail::buffer_waiter<T, Dimensions>;
00268 };
00269
00270 /** Proxy function to avoid some circular type recursion
00271
00272     \return a shared_ptr<task>
00273
00274     \todo To remove with some refactoring
00275 */
00276 template <typename BufferDetail>
00277 static std::shared_ptr<detail::task>
00278 buffer_add_to_task(BufferDetail buf,
00279                   handler *command_group_handler,
00280                   bool is_write_mode) {
00281     return buf->add_to_task(command_group_handler, is_write_mode);
00282 }
00283
00284 /// @} End the data Doxygen group
00285
00286 }
00287 }
00288 }
00289 }
00290 }
00291 }
00292
00293 /*
00294     # Some Emacs stuff:
00295     ### Local Variables:
00296     ### ispell-local-dictionary: "american"
00297     ### eval: (flyspell-prog-mode)
00298     ### End:
00299 */
00300
00301 #endif // TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_HPP

```

11.15 include/CL/sycl/buffer.hpp File Reference

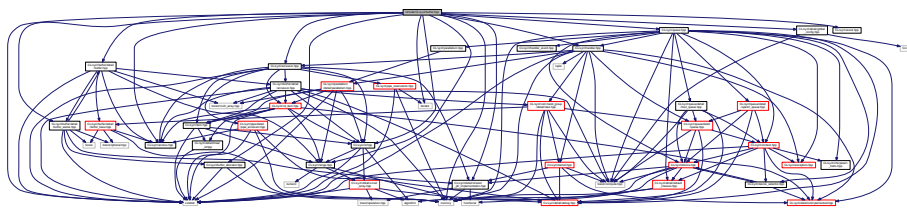
```
#include <cstddef>
```

```

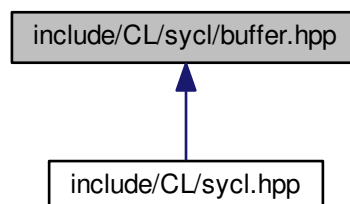
#include <iterator>
#include <memory>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/buffer/detail/buffer.hpp"
#include "CL/sycl/buffer/detail/buffer_waiter.hpp"
#include "CL/sycl/buffer_allocator.hpp"
#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#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 >](#)
 $\langle T, \text{Dimensions}, \text{Mode}, \text{Target} \rangle$ 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

```

00001 #ifndef TRISYCL_SYCL_BUFFER_HPP
00002 #define TRISYCL_SYCL_BUFFER_HPP
00003
00004 /** \file The OpenCL SYCL buffer<>
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 #include <cstdlib>
00013 #include <iterator>
00014 #include <memory>
00015
00016 #include "CL/sycl/access.hpp"
00017 #include "CL/sycl/accessor.hpp"
00018 #include "CL/sycl/buffer/detail/buffer.hpp"
00019 #include "CL/sycl/buffer/detail/buffer_waiter.hpp"
00020 #include "CL/sycl/buffer_allocator.hpp"
00021 #include "CL/sycl/detail/global_config.hpp"
00022 #include "CL/sycl/detail/shared_ptr_implementation.hpp"
00023 #include "CL/sycl/event.hpp"
00024 #include "CL/sycl/handler.hpp"
00025 #include "CL/sycl/id.hpp"
00026 #include "CL/sycl/queue.hpp"
00027 #include "CL/sycl/range.hpp"
00028
00029 namespace cl {
00030 namespace sycl {
00031
00032 /** \addtogroup<T, Dimensions, Mode, Target>up data Data access and storage in SYCL
00033     @{
00034 */
00035
00036 /** A SYCL buffer is a multidimensional variable length array (à la C99
00037     VIA or even Fortran before) that is used to store data to work on.
00038
00039     \todo We have some read-write buffers and some read-only buffers,
00040     according to the constructor called. So we could have some static
00041     checking for correctness with the accessors used, but we do not have a
00042     way in the specification to have a read-only buffer type for this.
00043
00044     \todo There is a naming inconsistency in the specification between
00045     buffer and accessor on T versus datatype
00046
00047     \todo Finish allocator implementation
00048
00049     \todo Think about the need of an allocator when constructing a buffer
00050     from other buffers
00051
00052     \todo Add constructors from arrays so that in C++17 the range and
00053     type can be inferred from the constructor
00054
00055     \todo Add constructors from array_ref
00056 */
00057 template <typename T,
00058           std::size_t Dimensions = 1,
00059           typename Allocator = buffer_allocator<T>>
00060 class buffer
00061     /* Use the underlying buffer waiter implementation that can be
00062     shared in the SYCL model */
00063     : public detail::shared_ptr_implementation<
00064         buffer<T, Dimensions, Allocator>,
00065         detail::buffer_waiter<T, Dimensions, Allocator>>,
00066         detail::debug<buffer<T, Dimensions, Allocator>> {
00067 public:
00068
00069     /// The STL-like types
00070     using value_type = T;
00071     using reference = value_type&;
00072     using const_reference = const value_type&;
00073     using allocator_type = Allocator;
00074
00075 private:
00076
00077     // The type encapsulating the implementation
00078     using implementation_t =
00079         detail::shared_ptr_implementation<
00080             buffer<T, Dimensions, Allocator>,
00081             detail::buffer_waiter<T, Dimensions, Allocator>
00082         >;
00083 public:

```

```

00084
00085 // Make the implementation member directly accessible in this class
00086 using implementation_t::implementation;
00087
00088 /** Use default constructors so that we can create a new buffer copy
00089     from another one, with either a l-value or an r-value (for
00090     std::move() for example).
00091
00092     Since we just copy the shared_ptr<> from the
00093     shared_ptr_implementation above, this is where/how the sharing
00094     magic is happening with reference counting in this case.
00095 */
00096 buffer() = default;
00097
00098
00099 /** Create a new buffer of the given size with
00100     storage managed by the SYCL runtime
00101
00102     The default behavior is to use the default host buffer
00103     allocator, in order to allow for host accesses. If the type of
00104     the buffer, has the const qualifier, then the default allocator
00105     will remove the qualifier to allow host access to the data.
00106
00107     \param[in] r defines the size
00108
00109     \param[in] allocator is to be used by the SYCL runtime
00110 */
00111 buffer(const range<Dimensions> &r, Allocator allocator = {})
00112 : implementation_t { detail::waiter(new
detail::buffer<T, Dimensions>
00113                                     { r }) }
00114 {}
00115
00116
00117 /** Create a new buffer with associated host memory
00118
00119     \param[in] host_data points to the storage and values used by
00120     the buffer
00121
00122     \param[in] r defines the size
00123
00124     \param[in] allocator is to be used by the SYCL runtime, of type
00125     cl::sycl::buffer_allocator<T> by default
00126
00127     The host address is const T, so the host accesses can be
00128     read-only.
00129
00130     However, the typename T is not const so the device accesses can
00131     be both read and write accesses. Since, the host_data is const,
00132     this buffer is only initialized with this memory and there is
00133     no write after its destruction, unless there is another final
00134     data address given after construction of the buffer.
00135 */
00136 buffer(const T *host_data,
00137        const range<Dimensions> &r,
00138        Allocator allocator = {})
00139 : implementation_t { detail::waiter(new
detail::buffer<T, Dimensions>
00140                                     { host_data, r }) }
00141 {}
00142
00143
00144 /** Create a new buffer with associated host memory
00145
00146     \param[inout] host_data points to the storage and values used by
00147     the buffer
00148
00149     \param[in] r defines the size
00150
00151     \param[in] allocator is to be used by the SYCL runtime, of type
00152     cl::sycl::buffer_allocator<T> by default
00153
00154     The memory is owned by the runtime during the lifetime of the
00155     object. Data is copied back to the host unless the user
00156     overrides the behavior using the set_final_data method. host_data
00157     points to the storage and values used by the buffer and
00158     range<dimensions> defines the size.
00159 */
00160 buffer(T *host_data, const range<Dimensions> &r, Allocator allocator = {})
00161 : implementation_t { detail::waiter(new
detail::buffer<T, Dimensions>
00162                                     { host_data, r }) }
00163 {}
00164
00165
00166 /** Create a new buffer with associated memory, using the data in
00167     host_data

```

```

00168
00169     \param[inout] host_data points to the storage and values used by
00170     the buffer
00171
00172     \param[in] r defines the size
00173
00174     \param[in] allocator is to be used by the SYCL runtime, of type
00175     cl::sycl::buffer_allocator<T> by default
00176
00177     The ownership of the host_data is shared between the runtime and the
00178     user. In order to enable both the user application and the SYCL
00179     runtime to use the same pointer, a cl::sycl::mutex_class is
00180     used. The mutex m is locked by the runtime whenever the data is in
00181     use and unlocked otherwise. Data is synchronized with host_data, when
00182     the mutex is unlocked by the runtime.
00183
00184     \todo update the specification to replace the pointer by a
00185     reference and provide the constructor with and without a mutex
00186 */
00187 buffer(shared_ptr_class<T> &host_data,
00188         const range<Dimensions> &buffer_range,
00189         cl::sycl::mutex_class &m,
00190         Allocator allocator = {}) {
00191     detail::unimplemented();
00192 }
00193
00194
00195 /** Create a new buffer with associated memory, using the data in
00196     host_data
00197
00198     \param[inout] host_data points to the storage and values used by
00199     the buffer
00200
00201     \param[in] r defines the size
00202
00203     \param[inout] m is the mutex used to protect the data access
00204
00205     \param[in] allocator is to be used by the SYCL runtime, of type
00206     cl::sycl::buffer_allocator<T> by default
00207
00208     The ownership of the host_data is shared between the runtime and the
00209     user. In order to enable both the user application and the SYCL
00210     runtime to use the same pointer, a cl::sycl::mutex_class is
00211     used.
00212
00213     \todo add this mutex-less constructor to the specification
00214 */
00215 buffer(shared_ptr_class<T> host_data,
00216         const range<Dimensions> &buffer_range,
00217         Allocator allocator = {})
00218 : implementation_t { detail::waiter(new
00219 detail::buffer<T, Dimensions>
00220     { host_data, buffer_range }) }
00221 {}
00222
00223 /** Create a new buffer which is initialized by host_data
00224
00225     \param[inout] host_data points to the storage and values used to
00226     initialize the buffer
00227
00228     \param[in] r defines the size
00229
00230     \param[in] allocator is to be used by the SYCL runtime, of type
00231     cl::sycl::buffer_allocator<T> by default
00232
00233     The SYCL runtime receives full ownership of the host_data unique_ptr
00234     and there in effect there is no synchronization with the application
00235     code using host_data.
00236
00237     \todo Update the API to add template <typename D =
00238     std::default_delete<T>> because the
00239     unique_ptr_class/std::unique_ptr have the destructor type as
00240     dependent
00241 */
00242 template <typename D = std::default_delete<T>>
00243 buffer(unique_ptr_class<T, D> &&host_data,
00244         const range<Dimensions> &buffer_range,
00245         Allocator allocator = {})
00246 // Just delegate to the constructor with normal pointer
00247 : buffer(host_data.get(), buffer_range, allocator) {
00248     // Then release the host_data memory
00249     host_data.release();
00250 }
00251
00252
00253 /** Create a new allocated 1D buffer initialized from the given

```



```

00254     elements ranging from first up to one before last
00255
00256     The data is copied to an intermediate memory position by the
00257     runtime. Data is written back to the same iterator set if the
00258     iterator is not a const iterator.
00259
00260     \param[inout] start_iterator points to the first element to copy
00261
00262     \param[in] end_iterator points to just after the last element to copy
00263
00264     \param[in] allocator is to be used by the SYCL runtime, of type
00265     cl::sycl::buffer_allocator<T> by default
00266
00267     \todo Implement the copy back at buffer destruction
00268
00269     \todo Generalize this for n-D and provide column-major and row-major
00270     initialization
00271
00272     \todo a reason to have this nD is that
00273           set_final_data(weak_ptr_class<T> & finalData) is actually
00274           doing this linearization anyway
00275
00276     \todo Allow read-only buffer construction too
00277
00278     \todo update the specification to deal with forward iterators
00279     instead and rewrite back only when it is non const and output
00280     iterator at least
00281
00282     \todo Allow initialization from ranges and collections à la STL
00283 */
00284 template <typename InputIterator,
00285           /* To force some iterator concept checking to avoid GCC 4.9
00286            diving into this when initializing from ({ int, int })
00287            which is a range<> and not an iterator... */
00288           typename ValueType =
00289           typename std::iterator_traits<InputIterator>::value_type>
00290 buffer(InputIterator start_iterator,
00291        InputIterator end_iterator,
00292        Allocator allocator = {}) :
00293     implementation_t { detail::waiter(new
00294     detail::buffer<T, Dimensions>
00295         { start_iterator, end_iterator }) }
00296 {}
00297
00298 /** Create a new sub-buffer without allocation to have separate
00299     accessors later
00300
00301     \param[inout] b is the buffer with the real data
00302
00303     \param[in] base_index specifies the origin of the sub-buffer inside the
00304     buffer b
00305
00306     \param[in] sub_range specifies the size of the sub-buffer
00307
00308     \todo To be implemented
00309
00310     \todo Update the specification to replace index by id
00311 */
00312 buffer(buffer<T, Dimensions, Allocator> &b,
00313        const id<Dimensions> &base_index,
00314        const range<Dimensions> &sub_range,
00315        Allocator allocator = {}) { detail::unimplemented(); }
00316
00317 #ifndef TRISYCL_OPENCL
00318 /** Create a buffer from an existing OpenCL memory object associated
00319     with a context after waiting for an event signaling the
00320     availability of the OpenCL data
00321
00322     \param[inout] mem_object is the OpenCL memory object to use
00323
00324     \param[inout] from_queue is the queue associated to the memory
00325     object
00326
00327     \param[in] available_event specifies the event to wait for if
00328     non null
00329
00330     Note that a buffer created from a cl_mem object will only have
00331     one underlying cl_mem for the lifetime of the buffer and use on
00332     an incompatible queue constitutes an error.
00333
00334     \todo To be implemented
00335
00336     \todo Improve the specification to allow CLHPP objects too
00337 */
00338
00339 buffer(cl_mem mem_object,

```

```

00340         queue from_queue,
00341         event available_event = {},
00342         Allocator allocator = {}) { detail::unimplemented(); }
00343 #endif
00344
00345
00346 // Use BOOST_DISABLE_ASSERTS at some time to disable range checking
00347
00348 /** Get an accessor to the buffer with the required mode
00349
00350     \param Mode is the requested access mode
00351
00352     \param Target is the type of object to be accessed
00353
00354     \param[in] command_group_handler is the command group handler in
00355     which the kernel is to be executed
00356
00357     \todo Do we need for an accessor to increase the reference count of
00358     a buffer object? It does make more sense for a host-side accessor.
00359
00360     \todo Implement the modes and targets
00361 */
00362 template <access::mode Mode,
00363           access::target Target = access::target::global_buffer
00364 >
00365 accessor<T, Dimensions, Mode, Target>
00366 get_access(handler &command_group_handler) {
00367     static_assert(Target == access::target::global_buffer
00368                 || Target == access::target::constant_buffer,
00369                 "get_access(handler) can only deal with access::global_buffer"
00370                 " or access::constant_buffer (for host_buffer accessor)"
00371                 " do not use a command group handler");
00372     return { *this, command_group_handler };
00373 }
00374
00375 /** Get a host accessor to the buffer with the required mode
00376
00377     \param Mode is the requested access mode
00378
00379     \todo Implement the modes
00380
00381     \todo More elegant solution
00382 */
00383 template <access::mode Mode,
00384           access::target Target = access::target::host_buffer>
00385 accessor<T, Dimensions, Mode, Target>
00386 get_access() {
00387     static_assert(Target == access::target::host_buffer,
00388                 "get_access() without a command group handler is only"
00389                 " for host_buffer accessor");
00390     return { *this };
00391 }
00392
00393
00394 /** Return a range object representing the size of the buffer in
00395     terms of number of elements in each dimension as passed to the
00396     constructor
00397
00398     \todo rename to the equivalent from array_ref proposals? Such
00399     as size() in
00400     http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0009r2.html
00401 */
00402 auto get_range() const {
00403     /* Interpret the shape which is a pointer to the first element as an
00404     array of Dimensions elements so that the range<Dimensions>
00405     constructor is happy with this collection
00406     */
00407     return implementation->implementation->get_range();
00408 }
00409
00410
00411 /** Returns the total number of elements in the buffer
00412
00413     Equal to get_range()[0] * ... * get_range()[dimensions-1].
00414 */
00415 auto get_count() const {
00416     return implementation->implementation->get_count();
00417 }
00418
00419
00420 /** Returns the size of the buffer storage in bytes
00421
00422     Equal to get_count()*sizeof(T).
00423
00424     \todo rename to something else. In
00425     http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf

```

```

00426         it is named bytes() for example
00427     */
00428     size_t get_size() const {
00429         return implementation->implementation->get_size();
00430     }
00431
00432
00433     /** Returns the number of buffers that are shared/referenced
00434
00435         For example
00436         \code
00437         cl::sycl::buffer<int> b { 1000 };
00438         // Here b.use_count() should return 1
00439         cl::sycl::buffer<int> c { b };
00440         // Here b.use_count() and c.use_count() should return 2
00441         \endcode
00442
00443         \todo Add to the specification, useful for validation
00444     */
00445     auto use_count() const {
00446         // Rely on the shared_ptr<> use_count()
00447         return implementation.use_count();
00448     }
00449
00450
00451     /** Ask for read-only status of the buffer
00452
00453         \todo Add to specification
00454     */
00455     bool is_read_only() const {
00456         return implementation->implementation->read_only();
00457     }
00458
00459
00460     /** Set destination of buffer data on destruction
00461
00462         The finalData points to the host memory to which, the outcome of all
00463         the buffer processing is going to be copied to.
00464
00465         This is the final pointer, which is going to be accessible after the
00466         destruction of the buffer and in the case where this is a valid
00467         pointer, the data are going to be copied to this host address.
00468
00469         finalData is different from the original host address, if the buffer
00470         was created associated with one. This is mainly to be used when a
00471         shared_ptr is given in the constructor and the output data will
00472         reside in a different location from the initialization data.
00473
00474         It is defined as a weak_ptr referring to a shared_ptr that is not
00475         associated with the cl::sycl::buffer, and so the cl::sycl::buffer
00476         will have no ownership of finalData.
00477
00478         \todo Update the API to take finalData by value instead of by
00479         reference. This way we can have an implicit conversion
00480         possible at the API call from a shared_ptr<>, avoiding an
00481         explicit weak_ptr<> creation
00482
00483         \todo figure out how set_final_data() interact with the other
00484         way to write back some data or with some data sharing with the
00485         host that can not be undone
00486     */
00487     void set_final_data(weak_ptr_class<T> finalData) {
00488         implementation->implementation->set_final_data(std::move(finalData));
00489     }
00490
00491 };
00492
00493 /// @} End the data Doxygen group
00494
00495 }
00496
00497
00498 /* Inject a custom specialization of std::hash to have the buffer
00499     usable into an unordered associative container
00500
00501     \todo Add this to the spec
00502 */
00503 namespace std {
00504     template <typename T,
00505             std::size_t Dimensions,
00506             typename Allocator>
00507     struct hash<cl::sycl::buffer<T, Dimensions, Allocator>> {
00508     };
00509
00510     auto operator() (const cl::sycl::buffer<T, Dimensions, Allocator>
00511                     &b) const {
00512         // Forward the hashing to the implementation

```

```

00512     return b.hash();
00513 }
00514
00515 };
00516
00517 }
00518
00519 /*
00520  # Some Emacs stuff:
00521  ### Local Variables:
00522  ### ispell-local-dictionary: "american"
00523  ### eval: (flyspell-prog-mode)
00524  ### End:
00525  */
00526
00527 #endif // TRISYCL_SYCL_BUFFER_HPP

```

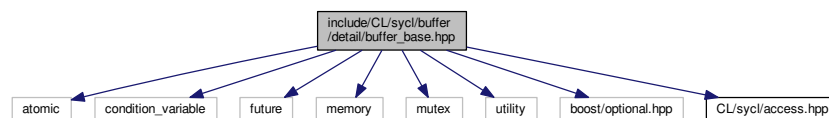
11.17 include/CL/sycl/detail/buffer_base.hpp File Reference

```

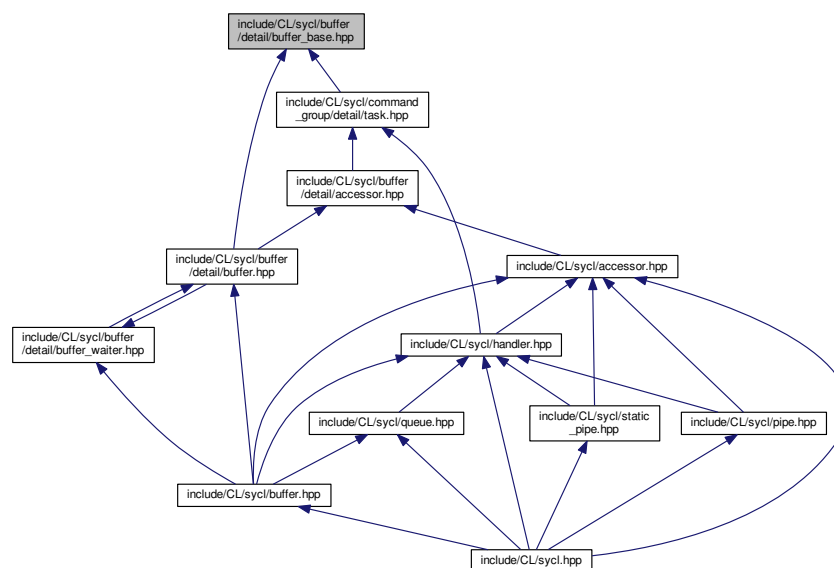
#include <atomic>
#include <condition_variable>
#include <future>
#include <memory>
#include <mutex>
#include <utility>
#include <boost/optional.hpp>
#include "CL/sycl/access.hpp"

```

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

```

00001 #ifndef TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_BASE_HPP
00002 #define TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_BASE_HPP
00003
00004 /** \file The buffer_base behind the buffers, independent of the data
00005     type
00006
00007     Ronan at Keryell point FR
00008
00009     This file is distributed under the University of Illinois Open Source
00010     License. See LICENSE.TXT for details.
00011 */
00012
00013 #include <atomic>
00014 #include <condition_variable>
00015 #include <future>
00016 #include <memory>
00017 #include <mutex>
00018 #include <utility>
00019
00020 // \todo Use C++17 optional when it is mainstream
00021 #include <boost/optional.hpp>
00022
00023 #include "CL/sycl/access.hpp"
00024
00025 namespace cl {
00026 namespace sycl {
00027
00028 class handler;
00029
00030 namespace detail {
00031
00032 struct task;
00033 struct buffer_base;
00034 inline static std::shared_ptr<detail::task>
00035 add_buffer_to_task(handler *command_group_handler,
00036                  std::shared_ptr<detail::buffer_base> b,
00037                  bool is_write_mode);
00038
00039 /** Factorize some template independent buffer aspects in a base class
00040     */
00041 struct buffer_base : public std::enable_shared_from_this<buffer_base> {
00042     /// If the data are read-only, store the information for later optimization.
00043     /// \todo Replace this by a static read-only type for the buffer
00044     bool read_only;
00045
00046     /// Keep track of the number of kernel accessors using this buffer
00047     std::atomic<size_t> number_of_users;
00048
00049     /// Track the latest task to produce this buffer
00050     std::weak_ptr<detail::task> latest_producer;

```

```

00051     /// To protect the access to latest_producer
00052     std::mutex latest_producer_mutex;
00053
00054     /// To signal when this buffer ready
00055     std::condition_variable ready;
00056     /// To protect the access to the condition variable
00057     std::mutex ready_mutex;
00058
00059     /** If the SYCL user buffer destructor is blocking, use this to
00060         block until this buffer implementation is destroyed.
00061
00062         Use a void promise since there is no value to send, only
00063         waiting */
00064     boost::optional<std::promise<void>> notify_buffer_destructor;
00065
00066     /// Create a buffer base
00067     buffer_base(bool read_only) : read_only { read_only },
00068                                   number_of_users { 0 } {}
00069
00070
00071
00072     /// The destructor wait for not being used anymore
00073     ~buffer_base() {
00074         wait();
00075         // If there is the last SYCL user buffer waiting, notify it
00076         if (notify_buffer_destructor)
00077             notify_buffer_destructor->set_value();
00078     }
00079
00080
00081     /// Wait for this buffer to be ready, which is no longer in use
00082     void wait() {
00083         std::unique_lock<std::mutex> ul { ready_mutex };
00084         ready.wait(ul, [&] {
00085             // When there is no producer for this buffer, we are ready to use it
00086             return number_of_users == 0;
00087         });
00088     }
00089
00090
00091     /// Mark this buffer in use by a task
00092     void use() {
00093         // Increment the use count
00094         ++number_of_users;
00095     }
00096
00097
00098     /// A task has released the buffer
00099     void release() {
00100         if (--number_of_users == 0)
00101             // Notify the host consumers or the buffer destructor that it is ready
00102             ready.notify_all();
00103     }
00104
00105
00106     /// Return the latest producer for the buffer
00107     std::shared_ptr<detail::task> get_latest_producer() {
00108         std::lock_guard<std::mutex> lg { latest_producer_mutex };
00109         // Return the valid shared_ptr to the task, if any
00110         return latest_producer.lock();
00111     }
00112
00113
00114     /** Return the latest producer for the buffer and set another
00115         future producer
00116     */
00117     std::shared_ptr<detail::task>
00118     set_latest_producer(std::weak_ptr<detail::task> newer_latest_producer) {
00119         std::lock_guard<std::mutex> lg { latest_producer_mutex };
00120         using std::swap;
00121
00122         swap(newer_latest_producer, latest_producer);
00123         // Return the valid shared_ptr to the previous producing task, if any
00124         return newer_latest_producer.lock();
00125     }
00126
00127
00128     /// Add a buffer to the task running the command group
00129     std::shared_ptr<detail::task>
00130     add_to_task(handler *command_group_handler, bool is_write_mode) {
00131         return add_buffer_to_task(command_group_handler,
00132                                   shared_from_this(),
00133                                   is_write_mode);
00134     }
00135
00136 };
00137

```

```

00138 }
00139 }
00140 }
00141
00142 /*
00143     # Some Emacs stuff:
00144     ### Local Variables:
00145     ###   ispell-local-dictionary: "american"
00146     ###   eval: (flyspell-prog-mode)
00147     ### End:
00148 */
00149
00150 #endif // TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_BASE_HPP

```

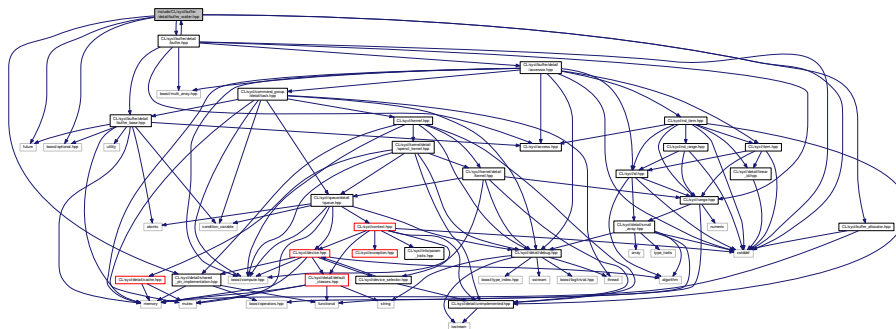
11.19 include/CL/sycl/buffer/detail/buffer_waiter.hpp File Reference

```

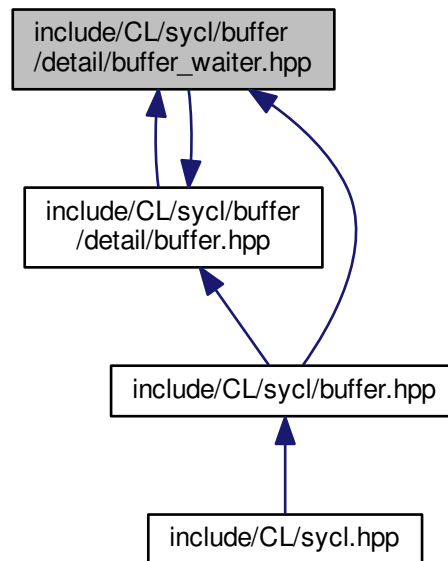
#include <cstdint>
#include <future>
#include <boost/optional.hpp>
#include "CL/sycl/buffer/detail/buffer.hpp"
#include "CL/sycl/buffer_allocator.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"

```

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

```

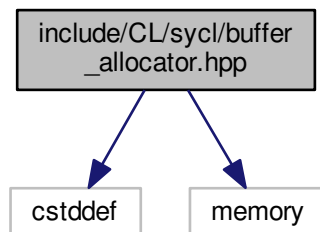
00001 #ifndef TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_WAITER_HPP
00002 #define TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_WAITER_HPP
00003
00004 /** \file A helper class to wait for the buffer<> detail
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013 #include <future>
00014
00015 // \todo Use C++17 optional when it is mainstream
00016 #include <boost/optional.hpp>
00017
00018 #include "CL/sycl/buffer/detail/buffer.hpp"
00019 #include "CL/sycl/buffer_allocator.hpp"
00020 #include "CL/sycl/detail/shared_ptr_implementation.hpp"
00021
00022 namespace cl {
00023 namespace sycl {
00024 namespace detail {
00025
00026 /** \addtogroup data Data access and storage in SYCL
00027     @{
00028 */
00029
00030 /** A helper class to wait for the final buffer destruction if the
00031     conditions for blocking are met
00032 */
00033 template <typename T,
00034           std::size_t Dimensions = 1,
00035           typename Allocator = buffer_allocator<T>>
00036 class buffer_waiter {
00037     public detail::shared_ptr_implementation<buffer_waiter<T,
00038                                           Dimensions,
00039                                           Allocator>,
00040                                           detail::buffer<T, Dimensions>>,
00041           detail::debug<buffer_waiter<T, Dimensions, Allocator>> {
00042
00043         // The type encapsulating the implementation
00044         using implementation_t =
00045             detail::shared_ptr_implementation<buffer_waiter<T, Dimensions, Allocator>
00046             ,
00047             detail::buffer<T, Dimensions>>;
00048     public:
00049
00050         // Make the implementation member directly accessible in this class
00051         using implementation_t::implementation;
00052
00053         /// Create a new buffer_waiter on top of a detail::buffer
00054         buffer_waiter(detail::buffer<T, Dimensions> *b) :
00055             implementation_t { b } {}
00056
00057         /** The buffer_waiter destructor waits for any data to be written
00058             back to the host, if any
00059         */
00060         ~buffer_waiter() {
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         }
00074     };
00075
00076
00077     /// Helper function to create a new buffer_waiter
00078     template <typename T,
00079             std::size_t Dimensions = 1>
00080     inline auto waiter(detail::buffer<T, Dimensions> *b) {
00081         return new buffer_waiter<T, Dimensions> { b };
00082     }

```

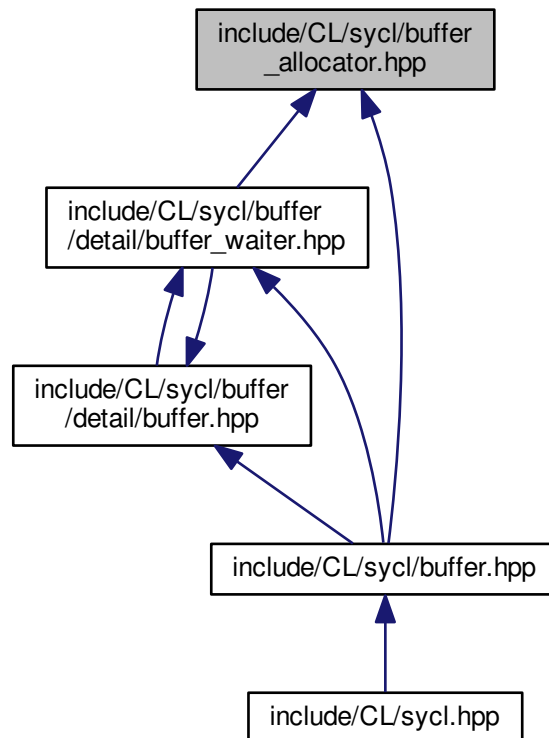
```
00083
00084 /// @} End the data Doxygen group
00085
00086 }
00087 }
00088 }
00089
00090 /*
00091     # Some Emacs stuff:
00092     ### Local Variables:
00093     ###   ispell-local-dictionary: "american"
00094     ###   eval: (flyspell-prog-mode)
00095     ### End:
00096 */
00097
00098 #endif /// // 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:



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

```

00001 #ifndef TRISYCL_SYCL_BUFFER_ALLOCATOR_HPP
00002 #define TRISYCL_SYCL_BUFFER_ALLOCATOR_HPP
00003
00004 /** \file The OpenCL SYCL buffer_allocator
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  #include <cstdint>
00013  #include <memory>
00014
00015  namespace cl {
00016  namespace sycl {
00017
00018  /** \addtogroup data Data access and storage in SYCL
00019      @{
00020  */
00021
00022  /** The default buffer allocator used by the runtime, when no allocator is
00023      defined by the user
00024
00025      Reuse the C++ default allocator.
00026  */
00027  template <typename T>
00028  using buffer_allocator = std::allocator<T>;
00029
00030  /// @} End the data Doxygen group
00031
00032  }
00033  }
00034
00035  /*
00036      # Some Emacs stuff:
00037      ### Local Variables:
00038      ###  ispell-local-dictionary: "american"
00039      ###  eval: (flyspell-prog-mode)
00040      ###  End:
00041  */
00042
00043  #endif // TRISYCL_SYCL_BUFFER_ALLOCATOR_HPP

```

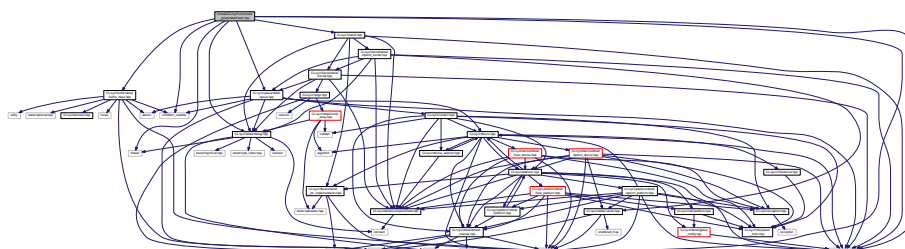
11.23 include/CL/sycl/command_group/detail/task.hpp File Reference

```

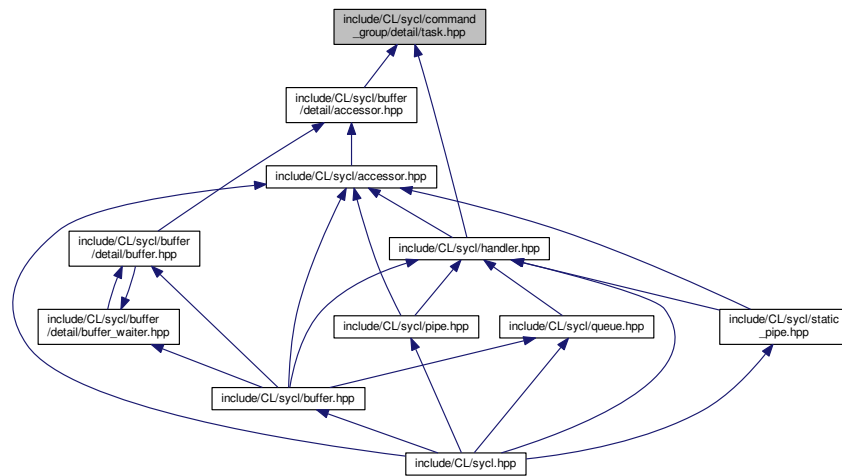
#include <condition_variable>
#include <memory>
#include <thread>
#include <boost/compute.hpp>
#include "CL/sycl/buffer/detail/buffer_base.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/kernel.hpp"
#include "CL/sycl/queue/detail/queue.hpp"

```

Include dependency graph for task.hpp:



This graph shows which files directly or indirectly include this file:



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

```

00001 #ifndef TRISYCL_SYCL_TASK_HPP
00002 #define TRISYCL_SYCL_TASK_HPP
00003
00004 /** \file The concept of task behind the scene
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 #include <condition_variable>
00013 #include <memory>
00014 #include <thread>
00015
00016 #ifdef TRISYCL_OPENCL
00017 #include <boost/compute.hpp>
00018 #endif
00019
00020 #include "CL/sycl/buffer/detail/buffer_base.hpp"
00021 #include "CL/sycl/detail/debug.hpp"
00022 #include "CL/sycl/kernel.hpp"
00023 #include "CL/sycl/queue/detail/queue.hpp"
00024
00025 namespace cl {

```

```

00026 namespace sycl {
00027 namespace detail {
00028
00029 /** The abstraction to represent SYCL tasks executing inside command_group
00030
00031     "enable_shared_from_this" allows to access the shared_ptr behind the
00032     scene.
00033 */
00034 struct task : public std::enable_shared_from_this<task>,
00035             public detail::debug<task> {
00036
00037     /** List of the buffers used by this task
00038
00039         \todo Use a set to check that some buffers are not used many
00040         times at least on writing
00041     */
00042     std::vector<std::shared_ptr<detail::buffer_base>> buffers_in_use;
00043
00044     /** The tasks producing the buffers used by this task
00045     std::vector<std::shared_ptr<detail::task>> producer_tasks;
00046
00047     /** Keep track of any prologue to be executed before the kernel
00048     std::vector<std::function<void(void)>> prologues;
00049
00050     /** Keep track of any epilogue to be executed after the kernel
00051     std::vector<std::function<void(void)>> epilogues;
00052
00053     /** Store if the execution ended, to be notified by task_ready
00054     bool execution_ended = false;
00055
00056     /** To signal when this task is ready
00057     std::condition_variable ready;
00058
00059     /** To protect the access to the condition variable
00060     std::mutex ready_mutex;
00061
00062     /** Keep track of the queue used to submission to notify kernel completion
00063     or to run OpenCL kernels on */
00064     std::shared_ptr<detail::queue> owner_queue;
00065
00066     std::shared_ptr<cl::sycl::detail::kernel> kernel;
00067
00068
00069     /** Create a task from a submitting queue
00070     task(const std::shared_ptr<detail::queue> &q)
00071         : owner_queue { q } {}
00072
00073
00074     /** Add a new task to the task graph and schedule for execution
00075     void schedule(std::function<void(void)> f) {
00076         /* To keep a copy of the task shared_ptr after the end of the
00077         command group, capture it by copy in the following lambda. This
00078         should be easier in C++17 with move semantics on capture
00079         */
00080         auto task = shared_from_this();
00081         auto execution = [=] {
00082             // Wait for the required tasks to be ready
00083             task->wait_for_producers();
00084             task->prelude();
00085             TRISYCL_DUMP_T("Execute the kernel");
00086             // Execute the kernel
00087             f();
00088             task->postlude();
00089             // Release the buffers that have been written by this task
00090             task->release_buffers();
00091             // Notify the waiting tasks that we are done
00092             task->notify_consumers();
00093             // Notify the queue we are done
00094             owner_queue->kernel_end();
00095             TRISYCL_DUMP_T("Task thread exit");
00096         };
00097         /* Notify the queue that there is a kernel submitted to the
00098         queue. Do not do it in the task constructor so that we can deal
00099         with command group without kernel and if we put it inside the
00100         thread, the queue may have finished before the thread is
00101         scheduled */
00102         owner_queue->kernel_start();
00103         /* \todo it may be implementable with packaged_task that would
00104         deal with exceptions in kernels
00105         */
00106         #if TRISYCL_ASYNC
00107         /* If in asynchronous execution mode, execute the functor in a new
00108         thread */
00109         std::thread thread(execution);
00110         TRISYCL_DUMP_T("Task thread started");
00111         /** Detach the thread since it will synchronize by its own means
00112

```

```

00113         \todo This is an issue if there is an exception in the kernel
00114         */
00115         thread.detach();
00116     #else
00117         // Just a synchronous execution otherwise
00118         execution();
00119     #endif
00120 }
00121
00122
00123 /// Wait for the required producer tasks to be ready
00124 void wait_for_producers() {
00125     TRISYCL_DUMP_T("Task " << this << " waits for the producer tasks");
00126     for (auto &t : producer_tasks)
00127         t->wait();
00128     // We can let the producers rest in peace
00129     producer_tasks.clear();
00130 }
00131
00132
00133 /// Release the buffers that have been used by this task
00134 void release_buffers() {
00135     TRISYCL_DUMP_T("Task " << this << " releases the written buffers");
00136     for (auto b: buffers_in_use)
00137         b->release();
00138     buffers_in_use.clear();
00139 }
00140
00141
00142 /// Notify the waiting tasks that we are done
00143 void notify_consumers() {
00144     TRISYCL_DUMP_T("Notify all the task waiting for this task " << this);
00145     execution_ended = true;
00146     /* \todo Verify that the memory model with the notify does not
00147        require some fence or atomic */
00148     ready.notify_all();
00149 }
00150
00151
00152 /** Wait for this task to be ready
00153
00154     This is to be called from another thread
00155 */
00156 void wait() {
00157     TRISYCL_DUMP_T("The task wait for task " << this << " to end");
00158     std::unique_lock<std::mutex> ul { ready_mutex };
00159     ready.wait(ul, [&] { return execution_ended; });
00160 }
00161
00162
00163 /** Register a buffer to this task
00164
00165     This is how the dependency graph is incrementally built.
00166 */
00167 void add_buffer(std::shared_ptr<detail::buffer_base> &buf,
00168               bool is_write_mode) {
00169     TRISYCL_DUMP_T("Add buffer " << buf << " in task " << this);
00170     /* Keep track of the use of the buffer to notify its release at
00171        the end of the execution */
00172     buffers_in_use.push_back(buf);
00173     // To be sure the buffer does not disappear before the kernel can run
00174     buf->use();
00175
00176     std::shared_ptr<detail::task> latest_producer;
00177     if (is_write_mode) {
00178         /* Set this task as the latest producer of the buffer so that
00179            another kernel may wait on this task */
00180         latest_producer = buf->set_latest_producer(shared_from_this());
00181     }
00182     else
00183         latest_producer = buf->get_latest_producer();
00184
00185     /* If the buffer is to be produced by a task, add the task in the
00186        producer list to wait on it before running the task core */
00187     if (latest_producer)
00188         producer_tasks.push_back(latest_producer);
00189 }
00190
00191
00192 /// Execute the prologues
00193 void prelude() {
00194     for (const auto &p : prologues)
00195         p();
00196     /* Free the functors that may own an accessor owning a buffer
00197        preventing the command group to complete */
00198     prologues.clear();
00199 }

```

```

00200
00201
00202     /// Execute the epilogues
00203 void postlude() {
00204     for (const auto &p : epilogues)
00205         p();
00206     /* Free the functors that may own an accessor owning a buffer
00207        preventing the command group to complete */
00208     epilogues.clear();
00209 }
00210
00211
00212     /// Add a function to the prelude to run before kernel execution
00213 void add_prelude(const std::function<void(void)> &f) {
00214     prologues.push_back(f);
00215 }
00216
00217
00218     /// Add a function to the postlude to run after kernel execution
00219 void add_postlude(const std::function<void(void)> &f) {
00220     epilogues.push_back(f);
00221 }
00222
00223
00224     /// Get the queue behind the task to run a kernel on
00225 auto get_queue() {
00226     return owner_queue;
00227 }
00228
00229
00230     /// Set the kernel running this task if any
00231 void set_kernel(const std::shared_ptr<cl::sycl::detail::kernel> &k) {
00232     kernel = k;
00233 }
00234
00235
00236     /** Get the kernel running if any
00237
00238         \todo Specify this error in the spec
00239     */
00240 cl::sycl::detail::kernel &get_kernel() {
00241     if (!kernel)
00242         throw non_cl_error("Cannot use an OpenCL kernel in this context");
00243     return *kernel;
00244 }
00245
00246 };
00247
00248 }
00249 }
00250 }
00251
00252 /*
00253     # Some Emacs stuff:
00254     ### Local Variables:
00255     ### ispell-local-dictionary: "american"
00256     ### eval: (flyspell-prog-mode)
00257     ### End:
00258 */
00259
00260 #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"

```


[illegible]

- class `cl::sycl::context`
SYCL context. [More...](#)

- `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

```

00001 #ifndef TRISYCL_SYCL_CONTEXT_HPP
00002 #define TRISYCL_SYCL_CONTEXT_HPP
00003
00004 /** \file The OpenCL SYCL context
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 #include <cstdint>
00013
00014 #include "CL/sycl/detail/default_classes.hpp"
00015 #include "CL/sycl/detail/unimplemented.hpp"
00016 #include "CL/sycl/device.hpp"
00017 #include "CL/sycl/device_selector.hpp"
00018 #include "CL/sycl/exception.hpp"
00019 #include "CL/sycl/info/param_traits.hpp"
00020 #include "CL/sycl/platform.hpp"
00021
00022 namespace cl {
00023 namespace sycl {
00024
00025     /** \addtogroup execution Platforms, contexts, devices and queues
00026         @{
00027     */
00028
00029     namespace info {
00030
00031         using gl_context_interop = bool;
00032
00033         /** Context information descriptors
00034
00035             \todo Should be unsigned int to be consistent with others?
00036         */
00037         enum class context : int {
00038             reference_count,
00039             num_devices,
00040             gl_interop
00041         };
00042
00043
00044         /** Query the return type for get_info() on context stuff
00045
00046             \todo To be implemented
00047         */
00048         TRISYCL_INFO_PARAM_TRAITS_ANY_T(info::context, void)
00049
00050     }
00051
00052
00053     /** SYCL context
00054
00055         The context class encapsulates an OpenCL context, which is implicitly
00056         created and the lifetime of the context instance defines the lifetime
00057         of the underlying OpenCL context instance.
00058
00059         On destruction clReleaseContext is called.
00060
00061         The default context is the SYCL host context containing only the SYCL
00062         host device.
00063     */

```

```

00064     \todo The implementation is quite minimal for now.
00065 */
00066 class context {
00067 public:
00068     /** Constructs a context object for SYCL host using an async_handler for
00071         handling asynchronous errors
00072
00073         Note that the default case asyncHandler = nullptr is handled by the
00074         default constructor.
00075     */
00076     explicit context(async_handler asyncHandler) {
00077         detail::unimplemented();
00078     }
00079
00080 #ifdef TRISYCL_OPENCL
00081     /** Context constructor, where the underlying OpenCL context is given as
00083         a parameter
00084
00085         The constructor executes a retain on the cl_context.
00086
00087         Return synchronous errors via the SYCL exception class and
00088         asynchronous errors are handled via the async_handler, if provided.
00089     */
00090     context(cl_context clContext, async_handler asyncHandler = nullptr) {
00091         detail::unimplemented();
00092     }
00093 #endif
00094
00095     /** Constructs a context object using a device_selector object
00096
00097         The context is constructed with a single device retrieved from the
00098         device_selector object provided.
00099
00100         Return synchronous errors via the SYCL exception class and
00101         asynchronous errors are handled via the async_handler, if provided.
00102     */
00103     context(const device_selector &deviceSelector,
00104             info::gl_context_interop interopFlag,
00105             async_handler asyncHandler = nullptr) {
00106         detail::unimplemented();
00107     }
00108
00109     /** Constructs a context object using a device object
00110
00111         Return synchronous errors via the SYCL exception class and
00112         asynchronous errors are handled via the async_handler, if provided.
00113     */
00114     context(const device &dev,
00115             info::gl_context_interop interopFlag,
00116             async_handler asyncHandler = nullptr) {
00117         detail::unimplemented();
00118     }
00119
00120     /** Constructs a context object using a platform object
00121
00122         Return synchronous errors via the SYCL exception class and
00123         asynchronous errors are handled via the async_handler, if provided.
00124     */
00125     context(const platform &plt,
00126             info::gl_context_interop interopFlag,
00127             async_handler asyncHandler = nullptr) {
00128         detail::unimplemented();
00129     }
00130
00131     /** Constructs a context object using a vector_class of device objects
00132
00133         Return synchronous errors via the SYCL exception class and
00134         asynchronous errors are handled via the async_handler, if provided.
00135
00136         \todo Update the specification to replace vector by collection
00137         concept.
00138     */
00139     context(const vector_class<device> &deviceList,
00140             info::gl_context_interop interopFlag,
00141             async_handler asyncHandler = nullptr) {
00142         detail::unimplemented();
00143     }
00144
00145     /** Default constructor that chooses the context according the
00148         heuristics of the default selector
00149
00150

```

```

00151         Return synchronous errors via the SYCL exception class.
00152
00153         Get the default constructors back.
00154     */
00155     context() = default;
00156
00157
00158 #ifndef TRISYCL_OPENCL
00159     /* Returns the underlying cl_context object, after retaining the cl_context.
00160
00161         Retains a reference to the returned cl_context object.
00162
00163         Caller should release it when finished.
00164     */
00165     cl_context get() const {
00166         detail::unimplemented();
00167         return {};
00168     }
00169 #endif
00170
00171
00172     /// Specifies whether the context is in SYCL Host Execution Mode.
00173     bool is_host() const {
00174         return true;
00175     }
00176
00177
00178     /** Returns the SYCL platform that the context is initialized for
00179
00180         \todo To be implemented
00181     */
00182     platform get_platform();
00183
00184
00185     /** Returns the set of devices that are part of this context
00186
00187         \todo To be implemented
00188     */
00189     vector_class<device> get_devices() const {
00190         detail::unimplemented();
00191         return {};
00192     }
00193
00194
00195     /** Queries OpenCL information for the under-lying cl context
00196
00197         \todo To be implemented
00198     */
00199     template <info::context Param>
00200     typename info::param_traits<info::context, Param>::type
00201     get_info() const {
00202         detail::unimplemented();
00203         return {};
00204     }
00205 };
00206
00207 /// @} to end the execution Doxygen group
00208 }
00209 }
00210 }
00211
00212 /*
00213     # Some Emacs stuff:
00214     ### Local Variables:
00215     ### ispell-local-dictionary: "american"
00216     ### eval: (flyspell-prog-mode)
00217     ### End:
00218 */
00219
00220 #endif // TRISYCL_SYCL_CONTEXT_HPP

```

11.27 include/CL/sycl/detail/array_tuple_helpers.hpp File Reference

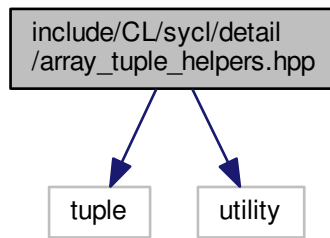
Some helpers to do array-tuple conversions.

```

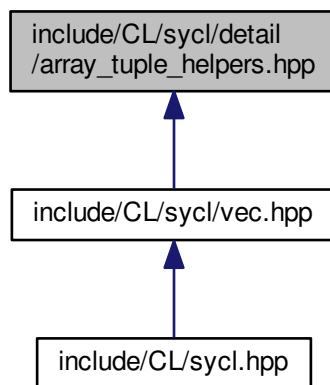
#include <tuple>
#include <utility>

```

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

```
00001 #ifndef TRISYCL_SYCL_DETAIL_ARRAY_TUPLE_HELPERS_HPP
00002 #define TRISYCL_SYCL_DETAIL_ARRAY_TUPLE_HELPERS_HPP
00003
00004 /** \file
00005
00006     Some helpers to do array-tuple conversions
00007
00008     Used for example to implement cl::sycl::vec<> class.
00009
00010     Ronan at Keryell point FR
00011
00012     This file is distributed under the University of Illinois Open Source
00013     License. See LICENSE.TXT for details.
00014 */
00015
00016 #include <tuple>
00017 #include <utility>
00018
00019 namespace cl {
00020 namespace sycl {
00021 namespace detail {
00022
00023 /** \addtogroup array_tuple_helpers Helpers to do array and tuple conversion
00024
00025     @{
00026 */
00027
00028 /** Helper to construct an array from initializer elements provided as a
00029     tuple
00030
00031     The trick is to get the std::index_sequence<> that represent 0,
00032     1,..., dimension-1 as a variadic template pack Is that we can
00033     iterate on, in this function.
00034 */
00035 template <typename V, typename Tuple, size_t... Is>
00036 std::array<typename V::element_type, V::dimension>
00037 tuple_to_array_iterate(Tuple t, std::index_sequence<Is...>) {
00038     /* The effect is like a static for-loop with Is counting from 0 to
00039        dimension-1 and thus constructing a uniform initialization { }
```

```

00040     construction from each tuple element:
00041     { std::get<0>(t), std::get<1>(t), ..., std::get<dimension-1>(t) }
00042
00043     The static cast is here to avoid the warning when there is a loss
00044     of precision, for example when initializing an int from a float.
00045 */
00046 return { { static_cast<typename V::element_type>(std::get<Is>(t))... } };
00047 }
00048
00049
00050 /** Construct an array from initializer elements provided as a tuple
00051 */
00052 template <typename V, typename Tuple>
00053 auto tuple_to_array(Tuple t) {
00054     /* Construct an index_sequence with 0, 1, ..., (size of the tuple-1)
00055     so that tuple_to_array_iterate can statically iterate on it */
00056     return tuple_to_array_iterate<V>(t,
00057                                     std::make_index_sequence<std::tuple_size<Tuple>::value>{});
00058 }
00059
00060
00061 /** Allows optional expansion of a 1-element tuple to a V::dimension
00062     tuple to replicate scalar values in vector initialization
00063 */
00064 template <typename V, typename Tuple, bool expansion = false>
00065 struct expand_to_vector {
00066     static_assert(V::dimension == std::tuple_size<Tuple>::value,
00067                 "The number of elements in initialization should match the dimension of the vector");
00068
00069     // By default, act as a pass-through and do not do any expansion
00070     static auto expand(Tuple t) { return t; }
00071 };
00072
00073
00074
00075 /** Specialization in the case we ask for expansion */
00076 template <typename V, typename Tuple>
00077 struct expand_to_vector<V, Tuple, true> {
00078     static_assert(std::tuple_size<Tuple>::value == 1,
00079                 "Since it is a vector initialization from a scalar there should be only one initializer
00080 value");
00081
00082     /** Construct a tuple from a value
00083
00084         \param value is used to initialize each tuple element
00085
00086         \param size is the number of elements of the tuple to be generated
00087
00088         The trick is to get the std::index_sequence<> that represent 0,
00089         1,..., dimension-1 as a variadic template pack Is that we can
00090         iterate on, in this function.
00091     */
00092     template <typename Value, size_t... Is>
00093     static auto fill_tuple(Value e, std::index_sequence<Is...>) {
00094         /* The effect is like a static for-loop with Is counting from 0 to
00095         dimension-1 and thus replicating the pattern to have
00096         make_tuple( (0, e), (1, e), ... (n - 1, e) )
00097
00098         Since the "," operator is just here to throw away the Is value
00099         (which is needed for the pack expansion...), at the end this is
00100         equivalent to:
00101         make_tuple( e, e, ..., e )
00102     */
00103     return std::make_tuple(((void)Is, e)...);
00104 }
00105
00106
00107 /** We expand the 1-element tuple by replicating into a tuple with the
00108     size of the vector */
00109 static auto expand(Tuple t) {
00110     return fill_tuple(std::get<0>(t),
00111                     std::make_index_sequence<V::dimension>{});
00112 }
00113
00114 };
00115
00116
00117 /** Create the array data of V from a tuple of initializer
00118
00119     If there is only 1 initializer, this is a scalar initialization of a
00120     vector and the value is expanded to all the vector elements first.
00121 */
00122 template <typename V, typename Tuple>
00123 auto expand(Tuple t) {
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 }
00131
00132 }
00133 }
00134 }
00135
00136 /*
00137     # Some Emacs stuff:
00138     ### Local Variables:
00139     ###   ispell-local-dictionary: "american"
00140     ###   eval: (flyspell-prog-mode)
00141     ### End:
00142 */
00143
00144 #endif // TRISYCL_SYCL_DETAIL_ARRAY_TUPLE_HELPERS_HPP

```

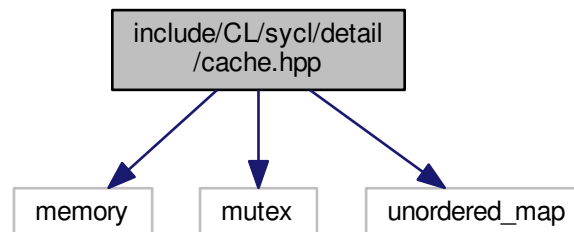
11.29 include/CL/sycl/detail/cache.hpp File Reference

```

#include <memory>
#include <mutex>
#include <unordered_map>

```

Include dependency graph for cache.hpp:



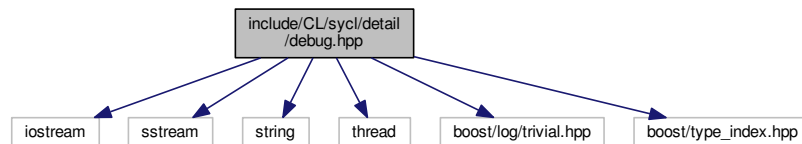

```

00018 namespace detail {
00019
00020
00021 /** A simple thread safe cache mechanism to cache std::shared_ptr of
00022     values indexed by keys
00023
00024     Since internally only std::weak_ptr are stored, this does not
00025     prevent object deletion but it is up to the programmer not to use
00026     this cache to retrieve deleted objects.
00027 */
00028 template <typename Key, typename Value>
00029 class cache {
00030
00031 public:
00032
00033     /// The type of the keys used to indexed the cache
00034     using key_type = Key;
00035
00036     /// The base type of the values stored in the cache
00037     using value_type = Value;
00038
00039 private:
00040
00041     /// The caching storage
00042     std::unordered_map<key_type, std::weak_ptr<value_type>> c;
00043
00044     /// To make the cache thread-safe
00045     std::mutex m;
00046
00047 public:
00048
00049     /** Get a value stored in the cache if present or insert by calling
00050         a generator function
00051
00052         \param[in] k is the key used to retrieve the value
00053
00054         \param[in] create_element is the function to be called if the
00055         key is not found in the cache to generate a value which is
00056         inserted for the key. This function has to produce a value
00057         convertible to a shared_ptr
00058
00059         \return a shared_ptr to the value retrieved or inserted
00060     */
00061     template <typename Functor>
00062     std::shared_ptr<value_type> get_or_register(const key_type &k,
00063                                              Functor &&create_element) {
00064         std::lock_guard<std::mutex> lg { m };
00065
00066         auto i = c.find(k);
00067         if (i != c.end())
00068             // Return the found element
00069             return std::shared_ptr<value_type>{ i->second };
00070
00071         // Otherwise create and insert a new element
00072         std::shared_ptr<value_type> e { create_element() };
00073         c.insert({ k, e });
00074         return e;
00075     }
00076
00077
00078     /** Remove an entry from the cache
00079
00080         \param[in] k is the key associated to the value to remove from
00081         the cache
00082     */
00083     void remove(const key_type &k) {
00084         std::lock_guard<std::mutex> lg { m };
00085         c.erase(k);
00086     }
00087 };
00088
00089
00090 }
00091
00092 }
00093
00094 /*
00095     # Some Emacs stuff:
00096     ### Local Variables:
00097     ### ispell-local-dictionary: "american"
00098     ### eval: (flyspell-prog-mode)
00099     ### End:
00100 */
00101
00102 #endif // TRISYCL_SYCL_DEVICE_CACHE_HPP

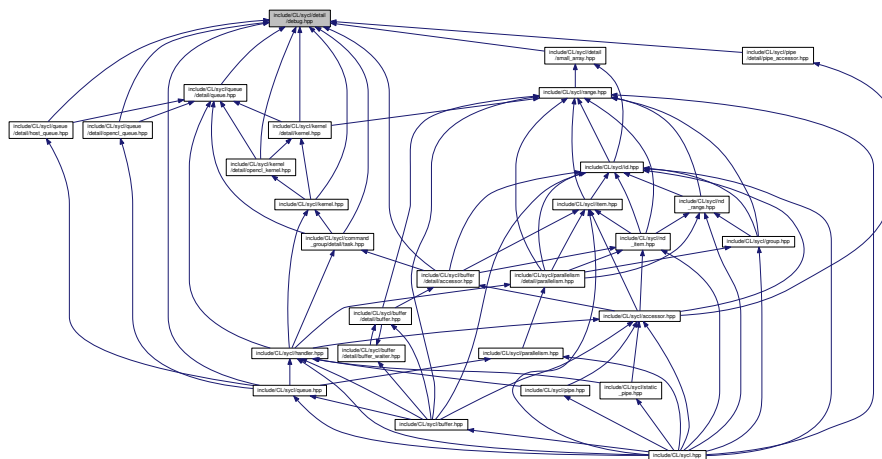
```

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 ostream 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

```

00001 #ifndef TRISYCL_SYCL_DETAIL_DEBUG_HPP
00002 #define TRISYCL_SYCL_DETAIL_DEBUG_HPP
00003
00004 /** \file Track constructor/destructor invocations and trace kernel execution
00005
00006     Define the TRISYCL_DEBUG CPP flag to have an output.
00007
00008     To use it in some class C, make C inherit from debug<C>.
00009
00010     Ronan at Keryell point FR
00011
00012     This file is distributed under the University of Illinois Open Source
00013     License. See LICENSE.TXT for details.
00014 */
00015
00016 #include <iostream>
00017
00018 // The common debug and trace infrastructure
00019 #if defined(TRISYCL_DEBUG) || defined(TRISYCL_TRACE_KERNEL)
00020 #include <sstream>
00021 #include <string>
00022 #include <thread>
00023
00024 #include <boost/log/trivial.hpp>
00025 #include <boost/type_index.hpp>
00026
00027 // To be able to construct string literals like "blah"s
00028 using namespace std::string_literals;
00029
00030 /** Dump a debug message in a formatted way.
00031
00032     Use an intermediate ostream because there are issues with
00033     BOOST_LOG_TRIVIAL to display C strings
00034 */
00035 #define TRISYCL_INTERNAL_DUMP(expression) do { \
00036     std::ostringstream s; \
00037     s << expression; \
00038     BOOST_LOG_TRIVIAL(debug) << s.str(); \
00039 } while(0)
00040 #endif
00041
00042 #ifndef TRISYCL_DEBUG
00043 #define TRISYCL_DUMP(expression) TRISYCL_INTERNAL_DUMP(expression)
00044
00045 /// Same as TRISYCL_DUMP() but with thread id first
00046 #define TRISYCL_DUMP_T(expression) \
00047     TRISYCL_DUMP("Thread " << std::hex \
00048                 << std::this_thread::get_id() << ": " << expression) \
00049 #else
00050 #define TRISYCL_DUMP(expression) do { } while(0)
00051 #define TRISYCL_DUMP_T(expression) do { } while(0)
00052 #endif
00053
00054 namespace cl {
00055 namespace sycl {
00056 namespace detail {
00057
00058 /** \addtogroup debug_trace Debugging and tracing support
00059     @{
00060 */
00061
00062 /** Class used to trace the construction, copy-construction,
00063     move-construction and destruction of classes that inherit from it
00064
00065     \param T is the real type name to be used in the debug output.
00066 */
00067 template <typename T>
00068 struct debug {
00069     // To trace the execution of the conSTRUCTORs and deSTRUCTORs
00070     #ifdef TRISYCL_DEBUG_STRUCTORS
00071     /// Trace the construction with the compiler-dependent mangled named
00072     debug() {
00073         TRISYCL_DUMP("Constructor of "
00074                     << boost::typeindex::type_id<T>().pretty_name()
00075                     << " " << (void*) this);
00076     }
00077
00078
00079     /** Trace the copy construction with the compiler-dependent mangled
00080         named
00081
00082         Only add this constructor if T has itself the same constructor,
00083         otherwise it may prevent the synthesis of default copy
00084         constructor and assignment.

```

```

00085  */
00086  template <typename U = T>
00087  debug(debug const &,
00088        /* Use intermediate U type to have the type dependent for
00089         enable_if to work
00090
00091         \todo Use is_copy_constructible_v when moving to C++17 */
00092         std::enable_if_t<std::is_copy_constructible<U>::value> * = 0) {
00093      TRISYCL_DUMP("Copy of " << boost::typeindex::type_id<T>().pretty_name()
00094                  << " " << (void*) this);
00095  }
00096
00097
00098  /** Trace the move construction with the compiler-dependent mangled
00099      named
00100
00101      Only add this constructor if T has itself the same constructor,
00102      otherwise it may prevent the synthesis of default move
00103      constructor and move assignment.
00104  */
00105  template <typename U = T>
00106  debug(debug &&,
00107        /* Use intermediate U type to have the type dependent for
00108         enable_if to work
00109
00110         \todo Use is_move_constructible_v when moving to C++17 */
00111         std::enable_if_t<std::is_move_constructible<U>::value> * = 0) {
00112      TRISYCL_DUMP("Move of " << boost::typeindex::type_id<T>().pretty_name()
00113                  << " " << (void*) this);
00114  }
00115
00116
00117  /// Trace the destruction with the compiler-dependent mangled named
00118  ~debug() {
00119      TRISYCL_DUMP("~ Destructor of "
00120                  << boost::typeindex::type_id<T>().pretty_name()
00121                  << " " << (void*) this);
00122  }
00123 #endif
00124 };
00125
00126
00127 /** Wrap a kernel functor in some tracing messages to have start/stop
00128     information when TRISYCL_TRACE_KERNEL macro is defined */
00129 template <typename KernelName, typename Functor>
00130 auto trace_kernel(const Functor &f) {
00131     #ifdef TRISYCL_TRACE_KERNEL
00132         // Inject tracing message around the kernel
00133         return [=] {
00134             /* Since the class KernelName may just be declared and not really
00135              defined, just use it through a class pointer to have
00136              typeid().name() not complaining */
00137             TRISYCL_INTERNAL_DUMP(
00138                 "Kernel started "
00139                 << boost::typeindex::type_id<KernelName *>().pretty_name());
00140             f();
00141             TRISYCL_INTERNAL_DUMP(
00142                 "Kernel stopped "
00143                 << boost::typeindex::type_id<KernelName *>().pretty_name());
00144         };
00145     #else
00146         // Identity by default
00147         return f;
00148     #endif
00149 }
00150
00151
00152 /** Class used to display a vector-like type of classes that inherit from
00153     it
00154
00155     \param T is the real type name to be used in the debug output.
00156
00157     Calling the display() method dump the values on std::cout
00158  */
00159 template <typename T>
00160 struct display_vector {
00161
00162     /// To debug and test
00163     void display() const {
00164         #ifdef TRISYCL_DEBUG
00165             std::cout << boost::typeindex::type_id<T>().pretty_name() << ":";
00166         #endif
00167         // Get a pointer to the real object
00168         for (auto e : *static_cast<const T *>(this))
00169             std::cout << " " << e;
00170         std::cout << std::endl;
00171     }

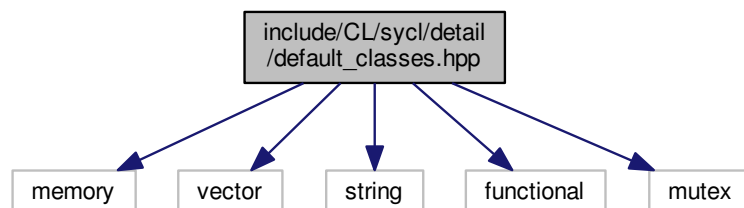
```

```
00172
00173 };
00174
00175 /// @} End the debug_trace Doxygen group
00176
00177 }
00178 }
00179 }
00180
00181 /*
00182     # Some Emacs stuff:
00183     ### Local Variables:
00184     ### ispell-local-dictionary: "american"
00185     ### eval: (flyspell-prog-mode)
00186     ### End:
00187 */
00188
00189 #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:



11.34 default_classes.hpp

```

00001 #ifndef TRISYCL_SYCL_DETAIL_DEFAULT_CLASSES_HPP
00002 #define TRISYCL_SYCL_DETAIL_DEFAULT_CLASSES_HPP
00003
00004 /** \file The OpenCL SYCL default classes to use from the STL according to
00005     section 3.2 of SYCL 1.2 specification
00006
00007     Ronan at Keryell point FR
00008
00009     This file is distributed under the University of Illinois Open Source
00010     License. See LICENSE.TXT for details.
00011 */
00012
00013 /** \addtogroup defaults Manage default configuration and types
00014     @{
00015 */
00016
00017 #ifndef CL_SYCL_NO_STD_VECTOR
00018 /** The vector type to be used as SYCL vector
00019     */
00020 #include <memory>
00021 #include <vector>
00022 namespace cl {
00023 namespace sycl {
00024
00025     template <class T, class Alloc = std::allocator<T>>
00026     using vector_class = std::vector<T, Alloc>;
00027
00028 }
00029 }
00030 #endif
00031
00032
00033 #ifndef CL_SYCL_NO_STD_STRING
00034 /** The string type to be used as SYCL string
00035     */
00036 #include <string>
00037 namespace cl {
00038 namespace sycl {
00039
00040     using string_class = std::string;
00041
00042 }
00043 }
00044 #endif
00045
00046
00047 #ifndef CL_SYCL_NO_STD_FUNCTION
00048 /** The functional type to be used as SYCL function
00049     */
00050 #include <functional>
00051 namespace cl {
00052 namespace sycl {
00053
00054     template <class R, class... ArgTypes>
00055     using function_class = std::function<R(ArgTypes...)>;
00056
00057 }
00058 }
00059 #endif
00060
00061
00062 #ifndef CL_SYCL_NO_STD_MUTEX
00063 /** The mutex type to be used as SYCL mutex
00064     */
00065 #include <mutex>
00066 namespace cl {
00067 namespace sycl {
00068
00069     using mutex_class = std::mutex;
00070
00071 }
00072 }
00073 #endif
00074
00075
00076 #ifndef CL_SYCL_NO_STD_UNIQUE_PTR
00077 /** The unique pointer type to be used as SYCL unique pointer
00078     */
00079 #include <memory>
00080 namespace cl {
00081 namespace sycl {
00082
00083     template <class T, class D = std::default_delete<T>>
00084     using unique_ptr_class = std::unique_ptr<T[], D>;

```

```

00085
00086 }
00087 }
00088 #endif
00089
00090
00091 #ifndef CL_SYCL_NO_STD_SHARED_PTR
00092 /** The shared pointer type to be used as SYCL shared pointer
00093  */
00094 #include <memory>
00095 namespace cl {
00096 namespace sycl {
00097
00098 template <class T>
00099 using shared_ptr_class = std::shared_ptr<T>;
00100
00101 }
00102 }
00103 #endif
00104
00105
00106 #ifndef CL_SYCL_NO_STD_WEAK_PTR
00107 /** The weak pointer type to be used as SYCL weak pointer
00108  */
00109 #include <memory>
00110 namespace cl {
00111 namespace sycl {
00112
00113 template <class T>
00114 using weak_ptr_class = std::weak_ptr<T>;
00115
00116 }
00117 }
00118 #endif
00119
00120 /// @} End the defaults Doxygen group
00121
00122 /*
00123  # Some Emacs stuff:
00124  ### Local Variables:
00125  ### ispell-local-dictionary: "american"
00126  ### eval: (flyspell-prog-mode)
00127  ### End:
00128 */
00129
00130 #endif // TRISYCL_SYCL_DETAIL_DEFAULT_CLASSES_HPP

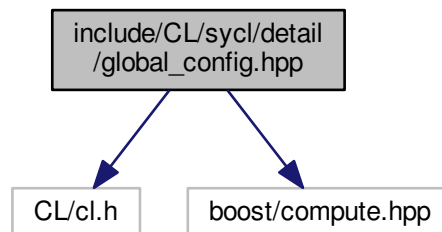
```

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:



[illegible]

- #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.

```
00001 #ifndef TRISYCL_SYCL_DETAIL_GLOBAL_CONFIG_HPP
```

```

00002 #define TRISYCL_SYCL_DETAIL_GLOBAL_CONFIG_HPP
00003
00004 /** \file The OpenCL SYCL details on the global triSYCL configuration
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 /** \addtogroup defaults Manage default configuration and types
00013     @{
00014 */
00015
00016 // The following symbols can be set to implement a different version
00017 #ifndef CL_SYCL_LANGUAGE_VERSION
00018 /// This implement SYCL 2.2
00019 #define CL_SYCL_LANGUAGE_VERSION 220
00020 #endif
00021
00022 #ifndef CL_TRISYCL_LANGUAGE_VERSION
00023 /// This implement triSYCL 2.2
00024 #define CL_TRISYCL_LANGUAGE_VERSION 220
00025 #endif
00026
00027 /// This source is compiled by a single source compiler
00028 #define __SYCL_SINGLE_SOURCE__
00029
00030
00031 /** Define TRISYCL_OPENCL to add OpenCL
00032
00033     triSYCL can indeed work without OpenCL if only host support is needed.
00034 */
00035 #ifdef TRISYCL_OPENCL
00036
00037 // SYCL interoperation API with OpenCL requires some OpenCL C types:
00038 #if defined(__APPLE__)
00039 #include <OpenCL/cl.h>
00040 #else
00041 #include <CL/cl.h>
00042 #endif
00043 // But the triSYCL OpenCL implementation is actually based on Boost.Compute
00044 #include <boost/compute.hpp>
00045 /// A macro to keep some stuff in OpenCL mode
00046 #define TRISYCL_SKIP_OPENCL(x) x
00047 #else
00048 /// A macro to skip stuff when not supporting OpenCL
00049 #define TRISYCL_SKIP_OPENCL(x)
00050 #endif
00051
00052 /** Allow the asynchronous implementation of tasks */
00053 #ifndef TRISYCL_ASYNC
00054 /** Use asynchronous tasks by default.
00055
00056     Is set to 0, the functors are executed synchronously.
00057 */
00058 #define TRISYCL_ASYNC 0
00059 #endif
00060
00061 /// @} End the defaults Doxygen group
00062
00063 /*
00064     # Some Emacs stuff:
00065     ### Local Variables:
00066     ### ispell-local-dictionary: "american"
00067     ### eval: (flyspell-prog-mode)
00068     ### End:
00069 */
00070
00071 #endif // TRISYCL_SYCL_DETAIL_GLOBAL_CONFIG_HPP

```

11.37 include/CL/sycl/detail/linear_id.hpp File Reference

```
#include <cstdint>
```

```
graph TD; A["include/CL/sycl/detail  
/linear_id.hpp"] --> B["cstddef"]
```

- `cl`
The vector type to be used as SYCL vector.
- `cl::sycl`
- `cl::sycl::detail`

- `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

```

00001 #ifndef TRISYCL_SYCL_DETAIL_LINEAR_ID_HPP
00002 #define TRISYCL_SYCL_DETAIL_LINEAR_ID_HPP
00003
00004 /** \file Compute linearized array access
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 #include <cstdint>
00013
00014 namespace cl {
00015 namespace sycl {
00016 namespace detail {
00017
00018 /** \addtogroup helpers Some helpers for the implementation
00019     @{
00020 */
00021
00022 /** Compute a linearized array access used in the OpenCL 2 world
00023
00024     Typically for the get_global_linear_id() and get_local_linear_id()
00025     functions.
00026 */
00027 template <typename Range, typename Id>
00028 size_t constexpr inline linear_id(Range range, Id id, Id offset = {}) {
00029     auto dims = std::distance(std::begin(range), std::end(range));
00030
00031     size_t linear_id = 0;
00032     /* A good compiler should unroll this and do partial evaluation to
00033        remove the first multiplication by 0 of this Horner evaluation and
00034        remove the 0 offset evaluation */
00035     for (int i = dims - 1; i >= 0; --i)
00036         linear_id = linear_id*range[i] + id[i] - offset[i];
00037
00038     return linear_id;
00039 }
00040
00041
00042 /// @} End the helpers Doxygen group
00043
00044 }
00045 }
00046 }
00047
00048 /*
00049     # Some Emacs stuff:
00050     ### Local Variables:
00051     ### ispell-local-dictionary: "american"
00052     ### eval: (flyspell-prog-mode)
00053     ### End:
00054 */
00055
00056 #endif // TRISYCL_SYCL_DETAIL_LINEAR_ID_HPP

```

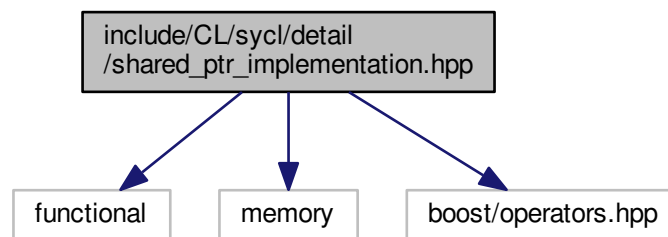
11.39 include/CL/sycl/detail/shared_ptr_implementation.hpp File Reference

```

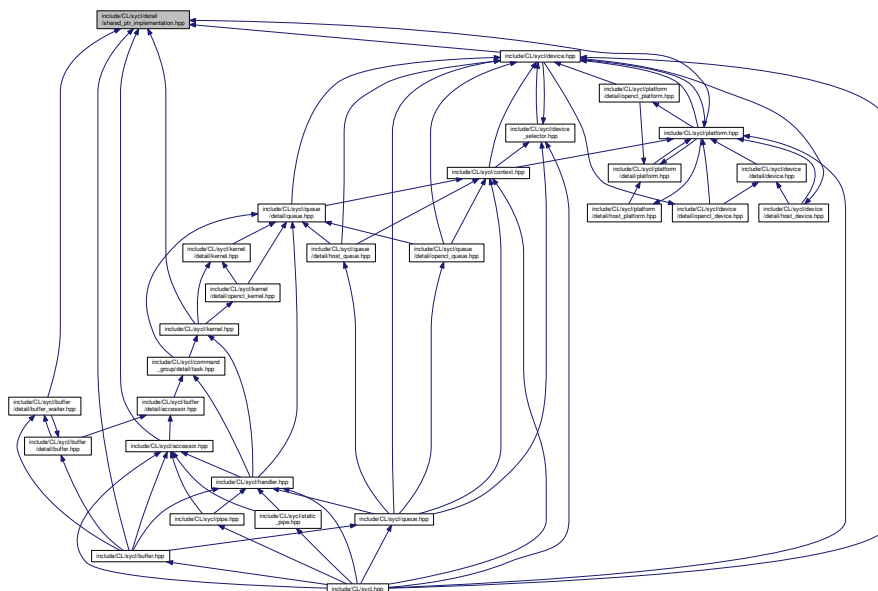
#include <functional>
#include <memory>
#include <boost/operators.hpp>

```

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

```

00001 #ifndef TRISYCL_SYCL_DETAIL_SHARED_PTR_IMPLEMENTATION_HPP
00002 #define TRISYCL_SYCL_DETAIL_SHARED_PTR_IMPLEMENTATION_HPP
00003
00004 /** \file Mix-in to add an implementation as shared_ptr with total
00005     ordering and hashing so that the class can be used with algorithms
00006     and in (un)ordered containers
00007
00008     Ronan at Keryell point FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 #include <functional>
00015 #include <memory>
00016
00017 #include <boost/operators.hpp>
00018
00019 namespace cl {
00020 namespace sycl {
00021 namespace detail {
00022
00023 /** Provide an implementation as shared_ptr with total ordering and
00024     hashing to be used with algorithms and in (un)ordered containers
00025
00026     To be used, a Parent class wanting an Implementation needs to
00027     inherit from.
00028
00029     The implementation ends up in a member really named
00030     "implementation".
00031
00032     \code
00033     public detail::shared_ptr_implementation<Parent, Implementation>
00034     \endcode
00035
00036     and also inject in std namespace a specialization for
00037     \code hash<Parent> \endcode
00038 */
00039 template <typename Parent, typename Implementation>
00040 struct shared_ptr_implementation : public boost::totally_ordered<Parent> {
00041
00042     /// The implementation forward everything to this... implementation
00043     std::shared_ptr<Implementation> implementation;
00044
00045 public:
00046
00047     /// The implementation directly as a shared pointer
00048     shared_ptr_implementation(std::shared_ptr<Implementation> i)
00049         : implementation { i } {}
00050
00051
00052     /// The implementation takes the ownership from a raw pointer
00053     shared_ptr_implementation(Implementation *i) : implementation { i } {}
00054
00055
00056     /// Keep all other constructors to have usual shared_ptr behaviour
00057     shared_ptr_implementation() = default;
00058
00059
00060     /** Equality operator
00061
00062     This is generalized by boost::equality_comparable from
00063     boost::totally_ordered to implement the equality comparable
00064     concept
00065     */
00066     bool operator ==(const Parent &other) const {
00067         return implementation == other.implementation;
00068     }
00069
00070
00071     /** Inferior operator
00072
00073     This is generalized by boost::less_than_comparable from
00074     boost::totally_ordered to implement the equality comparable
00075     concept
00076
00077     \todo Add this to the spec
00078     */
00079     bool operator <(const Parent &other) const {
00080         return implementation < other.implementation;
00081     }
00082
00083
00084     /// Forward the hashing for unordered containers to the implementation

```

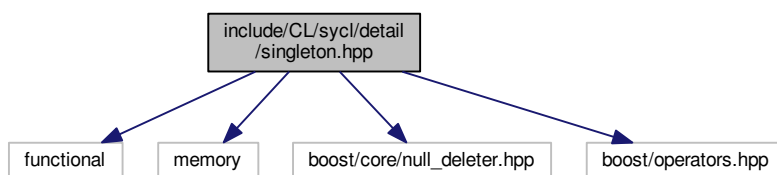


```
00085     auto hash() const {
00086         return std::hash<decltype(implementation)>{}(implementation);
00087     }
00088 };
00089 };
00090
00091 }
00092 }
00093 }
00094
00095 /*
00096     # Some Emacs stuff:
00097     ### Local Variables:
00098     ###   ispell-local-dictionary: "american"
00099     ###   eval: (flyspell-prog-mode)
00100     ### End:
00101 */
00102
00103 #endif // TRISYCL_SYCL_DETAIL_SHARED_PTR_IMPLEMENTATION_HPP
```

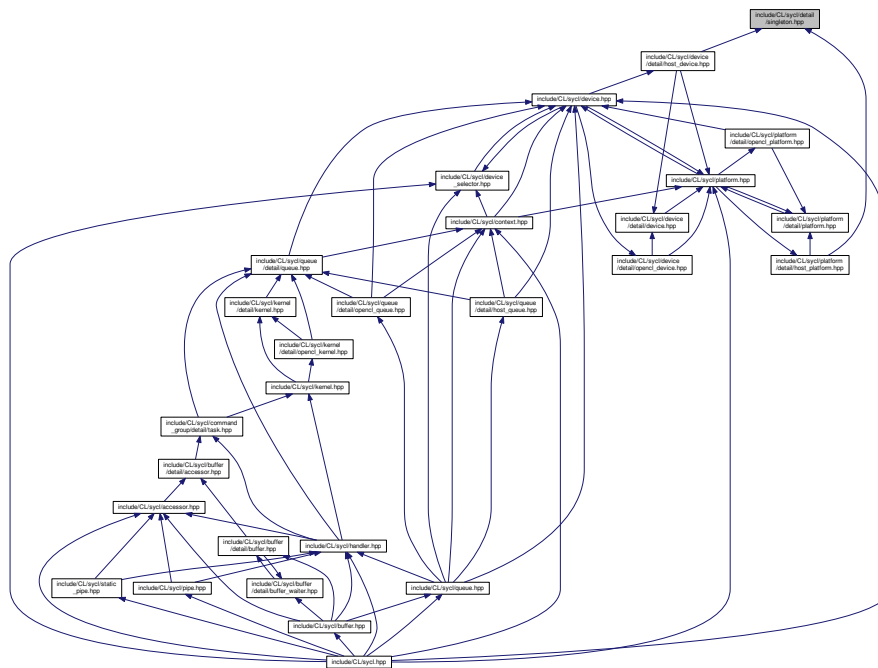
11.41 include/CL/sycl/detail/singleton.hpp File Reference

```
#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:



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

```

00001 #ifndef TRISYCL_SYCL_DETAIL_SINGLETON_HPP
00002 #define TRISYCL_SYCL_DETAIL_SINGLETON_HPP
00003
00004 /** \file Mix-in to add a singleton implementation with an instance() method
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 #include <functional>
00013 #include <memory>
00014
00015 #include <boost/core/null_deleter.hpp>
00016 #include <boost/operators.hpp>
00017
00018

```

```

00019 namespace cl {
00020 namespace sycl {
00021 namespace detail {
00022
00023 /// Provide a singleton factory
00024 template <typename T>
00025 struct singleton {
00026
00027     /// Get a singleton instance of T
00028     static std::shared_ptr<T> instance() {
00029         // C++11 guarantees 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     }
00038 };
00039 };
00040
00041 }
00042 }
00043 }
00044
00045 /*
00046     # Some Emacs stuff:
00047     ### Local Variables:
00048     ### ispell-local-dictionary: "american"
00049     ### eval: (flyspell-prog-mode)
00050     ### End:
00051 */
00052
00053 #endif // TRISYCL_SYCL_DETAIL_SINGLETON_HPP

```

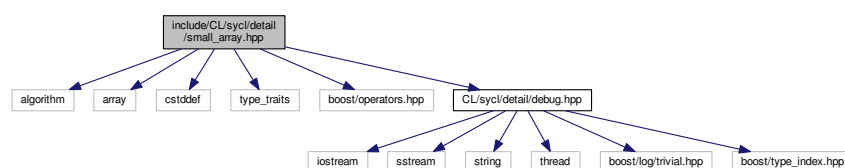
11.43 include/CL/sycl/detail/small_array.hpp File Reference

```

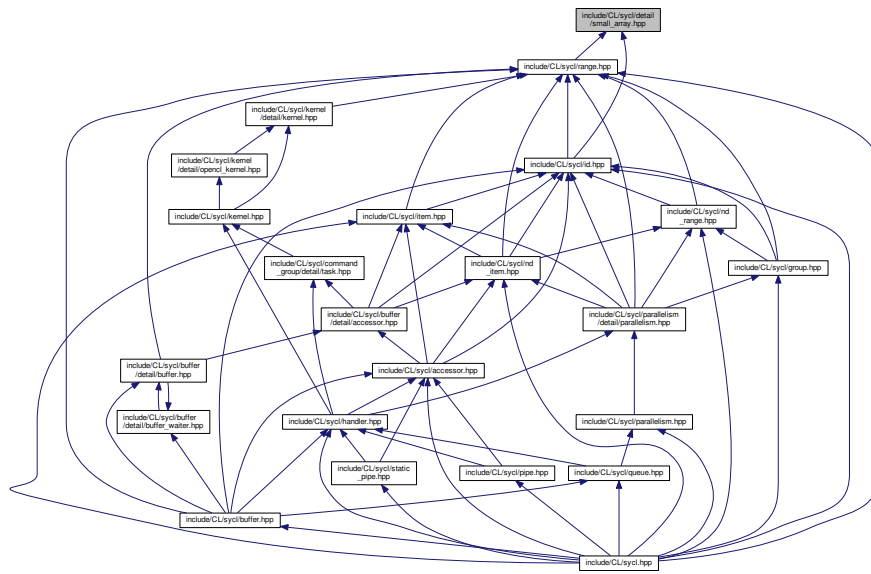
#include <algorithm>
#include <array>
#include <cstdint>
#include <type_traits>
#include <boost/operators.hpp>
#include "CL/sycl/detail/debug.hpp"

```

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

```

00001 #ifndef TRISYCL_SYCL_DETAIL_SMALL_ARRAY_HPP
00002 #define TRISYCL_SYCL_DETAIL_SMALL_ARRAY_HPP
00003
00004 /** \file This is a small array class to build range<>, id<>, etc.
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 #include <algorithm>
00013 #include <array>
00014 #include <cstdint>
00015 #include <type_traits>
00016
00017 #include <boost/operators.hpp>
00018
00019 #include "CL/sycl/detail/debug.hpp"
00020
00021
00022 namespace cl {
00023 namespace sycl {
00024 namespace detail {
00025
00026 /** \addtogroup helpers Some helpers for the implementation
00027     @{
00028 */
00029
00030
00031 /** Helper macro to declare a vector operation with the given side-effect
00032     operator */
00033 #define TRISYCL_BOOST_OPERATOR_VECTOR_OP(op) \
00034     FinalType operator op(const FinalType &rhs) { \
00035         for (std::size_t i = 0; i != Dims; ++i) \
00036             (*this)[i] op rhs[i]; \
00037         return *this; \
00038     }
00039
00040
00041 /** Define a multi-dimensional index, used for example to locate a work
00042     item or a buffer element
00043
00044     Unfortunately, even if std::array is an aggregate class allowing
00045     native list initialization, it is no longer an aggregate if we derive
00046     from an aggregate. Thus we have to redeclare the constructors.
00047
00048     \param BasicType is the type element, such as int
00049
00050     \param Dims is the dimension number, typically between 1 and 3
00051
00052     \param FinalType is the final type, such as range<> or id<>, so that
00053     boost::operator can return the right type
00054
00055     \param EnableArgsConstructor adds a constructors from Dims variadic
00056     elements when true. It is false by default.
00057
00058     std::array<> provides the collection concept, with .size(), == and !=
00059     too.
00060 */
00061 template <typename BasicType,
00062           typename FinalType,
00063           std::size_t Dims,
00064           bool EnableArgsConstructor = false>
00065 struct small_array : std::array<BasicType, Dims>,
00066 // To have all the usual arithmetic operations on this type
00067 boost::euclidean_ring_operators<FinalType>,
00068 // Bitwise operations
00069 boost::bitwise<FinalType>,
00070 // Shift operations
00071 boost::shiftable<FinalType>,
00072 // Already provided by array<> lexicographically:
00073 // boost::equality_comparable<FinalType>,
00074 // boost::less_than_comparable<FinalType>,
00075 // Add a display() method
00076 detail::display_vector<FinalType> {
00077
00078     /// \todo add this Boost::multi_array or STL concept to the
00079     /// specification?
00080     static const auto dimensionality = Dims;
00081
00082     /* Note that constexpr size() from the underlying std::array provides
00083        the same functionality */
00084     static const size_t dimension = Dims;

```

```

00085     using element_type = BasicType;
00086
00087
00088     /** A constructor from another array
00089
00090         Make it explicit to avoid spurious range<> constructions from int *
00091         for example
00092     */
00093     template <typename SourceType>
00094     small_array(const SourceType src[Dims]) {
00095         // (*this)[0] is the first element of the underlying array
00096         std::copy_n(src, Dims, &(*this)[0]);
00097     }
00098
00099
00100     /// A constructor from another small_array of the same size
00101     template <typename SourceBasicType,
00102              typename SourceFinalType,
00103              bool SourceEnableArgsConstructor>
00104     small_array(const small_array<SourceBasicType,
00105                               SourceFinalType,
00106                               Dims,
00107                               SourceEnableArgsConstructor> &src) {
00108         std::copy_n(&src[0], Dims, &(*this)[0]);
00109     }
00110
00111
00112     /** Initialize the array from a list of elements
00113
00114         Strangely, even when using the array constructors, the
00115         initialization of the aggregate is not available. So recreate an
00116         equivalent here.
00117
00118         Since there are inherited types that defines some constructors with
00119         some conflicts, make it optional here, according to
00120         EnableArgsConstructor template parameter.
00121     */
00122     template <typename... Types,
00123              // Just to make enable_if depend of the template and work
00124              bool Depend = true,
00125              typename = typename std::enable_if_t<EnableArgsConstructor
00126                                                  && Depend>>
00127     small_array(const Types &... args)
00128         : std::array<BasicType, Dims> {
00129         // Allow a loss of precision in initialization with the static_cast
00130         { static_cast<BasicType>(args)... }
00131     }
00132     {
00133         static_assert(sizeof...(args) == Dims,
00134                       "The number of initializing elements should match "
00135                       "the dimension");
00136     }
00137
00138
00139     /// Construct a small_array from a std::array
00140     template <typename SourceBasicType>
00141     small_array(const std::array<SourceBasicType, Dims> &src)
00142         : std::array<BasicType, Dims>(src) {}
00143
00144
00145     /// Keep other constructors from the underlying std::array
00146     using std::array<BasicType, Dims>::array;
00147
00148     /// Keep the synthesized constructors
00149     small_array() = default;
00150
00151     /// Return the element of the array
00152     auto get(std::size_t index) const {
00153         return (*this)[index];
00154     }
00155
00156     /* Implement minimal methods boost::euclidean_ring_operators needs to
00157        generate everything */
00158     /// Add + like operations on the id<> and others
00159     TRISYCL_BOOST_OPERATOR_VECTOR_OP(+=)
00160
00161     /// Add - like operations on the id<> and others
00162     TRISYCL_BOOST_OPERATOR_VECTOR_OP(-=)
00163
00164     /// Add * like operations on the id<> and others
00165     TRISYCL_BOOST_OPERATOR_VECTOR_OP(*=)
00166
00167     /// Add / like operations on the id<> and others
00168     TRISYCL_BOOST_OPERATOR_VECTOR_OP(/=)
00169
00170     /// Add % like operations on the id<> and others
00171     TRISYCL_BOOST_OPERATOR_VECTOR_OP(%=)

```

```

00172
00173 /// Add << like operations on the id<> and others
00174 TRISYCL_BOOST_OPERATOR_VECTOR_OP(<<=)
00175
00176 /// Add >> like operations on the id<> and others
00177 TRISYCL_BOOST_OPERATOR_VECTOR_OP(>>=)
00178
00179 /// Add & like operations on the id<> and others
00180 TRISYCL_BOOST_OPERATOR_VECTOR_OP(&=)
00181
00182 /// Add ^ like operations on the id<> and others
00183 TRISYCL_BOOST_OPERATOR_VECTOR_OP(^=)
00184
00185 /// Add | like operations on the id<> and others
00186 TRISYCL_BOOST_OPERATOR_VECTOR_OP(|=)
00187
00188
00189 /** Since the boost::operator work on the small_array, add an implicit
00190     conversion to produce the expected type */
00191 operator FinalType () {
00192     return *static_cast<FinalType *>(this);
00193 }
00194
00195 };
00196
00197
00198 /** A small array of 1, 2 or 3 elements with the implicit constructors */
00199 template <typename BasicType, typename FinalType, std::size_t Dims>
00200 struct small_array_123 : small_array<BasicType, FinalType, Dims> {
00201     static_assert(1 <= Dims && Dims <= 3,
00202         "Dimensions are between 1 and 3");
00203 };
00204
00205
00206 /** Use some specializations so that some function overloads can be
00207     determined according to some implicit constructors and to have an
00208     implicit conversion from/to BasicType (such as an int typically) if
00209     dims = 1
00210 */
00211 template <typename BasicType, typename FinalType>
00212 struct small_array_123<BasicType, FinalType, 1>
00213 : public small_array<BasicType, FinalType, 1> {
00214     /// A 1-D constructor to have implicit conversion from from 1 integer
00215     /// and automatic inference of the dimensionality
00216     small_array_123(BasicType x) {
00217         (*this)[0] = x;
00218     }
00219
00220
00221     /// Keep other constructors
00222     small_array_123() = default;
00223
00224     using small_array<BasicType, FinalType, 1>::small_array;
00225
00226     /** Conversion so that an for example an id<1> can basically be used
00227         like an integer */
00228     operator BasicType() const {
00229         return (*this)[0];
00230     }
00231 };
00232
00233
00234 template <typename BasicType, typename FinalType>
00235 struct small_array_123<BasicType, FinalType, 2>
00236 : public small_array<BasicType, FinalType, 2> {
00237     /// A 2-D constructor to have implicit conversion from from 2 integers
00238     /// and automatic inference of the dimensionality
00239     small_array_123(BasicType x, BasicType y) {
00240         (*this)[0] = x;
00241         (*this)[1] = y;
00242     }
00243
00244
00245     /** Broadcasting constructor initializing all the elements with the
00246         same value
00247
00248         \todo Add to the specification of the range, id...
00249     */
00250     explicit small_array_123(BasicType e) : small_array_123 { e, e } { }
00251
00252
00253     /// Keep other constructors
00254     small_array_123() = default;
00255
00256     using small_array<BasicType, FinalType, 2>::small_array;
00257 };
00258

```

```

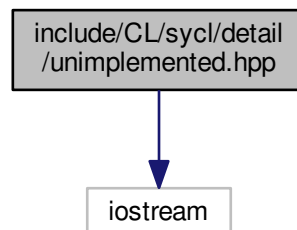
00259
00260 template <typename BasicType, typename FinalType>
00261 struct small_array_123<BasicType, FinalType, 3>
00262 : public small_array<BasicType, FinalType, 3> {
00263     /// A 3-D constructor to have implicit conversion from 3 integers
00264     /// and automatic inference of the dimensionality
00265     small_array_123(BasicType x, BasicType y, BasicType z) {
00266         (*this)[0] = x;
00267         (*this)[1] = y;
00268         (*this)[2] = z;
00269     }
00270
00271
00272     /** Broadcasting constructor initializing all the elements with the
00273         same value
00274
00275         \todo Add to the specification of the range, id...
00276     */
00277     explicit small_array_123(BasicType e) : small_array_123 { e, e, e } { }
00278
00279
00280     /// Keep other constructors
00281     small_array_123() = default;
00282
00283     using small_array<BasicType, FinalType, 3>::small_array;
00284 };
00285
00286 /// @} End the helpers Doxygen group
00287
00288 }
00289 }
00290 }
00291
00292 /*
00293     # Some Emacs stuff:
00294     ### Local Variables:
00295     ### ispell-local-dictionary: "american"
00296     ### eval: (flyspell-prog-mode)
00297     ### End:
00298 */
00299
00300 #endif // TRISYCL_SYCL_DETAIL_SMALL_ARRAY_HPP

```

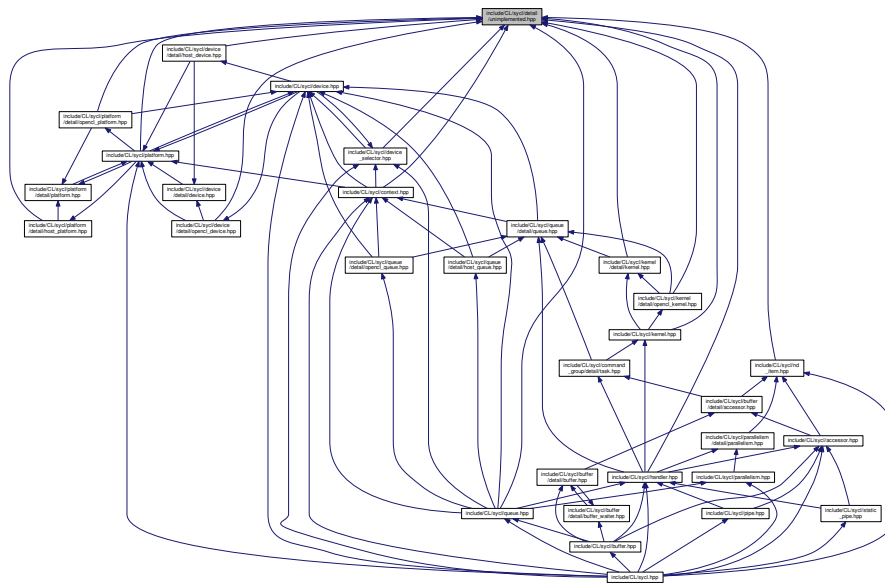
11.45 include/CL/sycl/detail/unimplemented.hpp File Reference

#include <iostream>

Include dependency graph for unimplemented.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- `cl`
The vector type to be used as SYCL vector.
- `cl::sycl`
- `cl::sycl::detail`

Functions

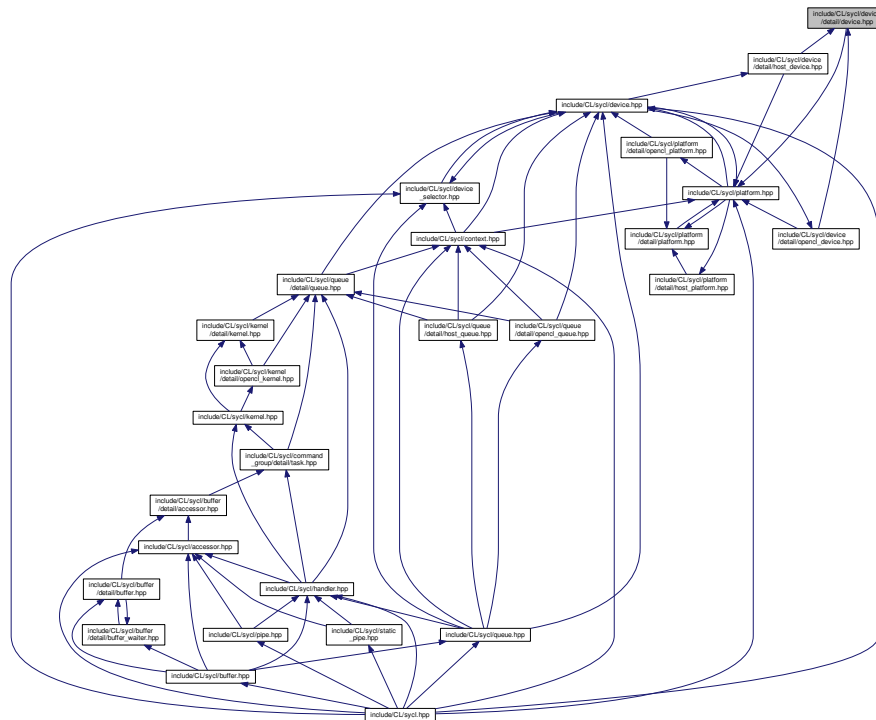
- `void cl::sycl::detail::unimplemented()`
Display an "unimplemented" message.

11.46 unimplemented.hpp

```

00001 #ifndef TRISYCL_SYCL_DETAIL_UNIMPLEMENTED_HPP
00002 #define TRISYCL_SYCL_DETAIL_UNIMPLEMENTED_HPP
00003
00004 /** \file Deal with unimplemented features
00005     Ronan at Keryell point FR
00006
00007     This file is distributed under the University of Illinois Open Source
00008     License. See LICENSE.TXT for details.
00009 */
00010
00011 #include <iostream>
00012
00013 namespace cl {
00014 namespace sycl {
00015 namespace detail {
00016
00017 /** \addtogroup helpers Some helpers for the implementation
00018     @{
00019 */
00020
00021 /** Display an "unimplemented" message
00022
00023     Can be changed to call assert(0) or whatever.
  
```


This graph shows which files directly or indirectly include this file:



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

```

00001 #ifndef TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_HPP
00002 #define TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_HPP
00003
00004 /** \file The OpenCL SYCL abstract device
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 #include "CL/sycl/detail/default_classes.hpp"
00013
00014 #include "CL/sycl/platform.hpp"
00015

```

```

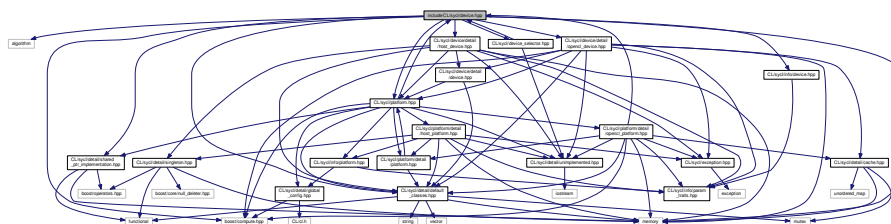
00016 namespace cl {
00017 namespace sycl {
00018 namespace detail {
00019
00020 /** \addtogroup execution Platforms, contexts, devices and queues
00021     @{
00022 */
00023
00024 /// An abstract class representing various models of SYCL devices
00025 class device {
00026
00027 public:
00028
00029 #ifdef TRISYCL_OPENCL
00030     /// Return the cl_device_id of the underlying OpenCL platform
00031     virtual cl_device_id get() const = 0;
00032 #endif
00033
00034
00035     /// Return true if the device is a SYCL host device
00036     virtual bool is_host() const = 0;
00037
00038
00039     /// Return true if the device is an OpenCL CPU device
00040     virtual bool is_cpu() const = 0;
00041
00042
00043     /// Return true if the device is an OpenCL GPU device
00044     virtual bool is_gpu() const = 0;
00045
00046
00047     /// Return true if the device is an OpenCL accelerator device
00048     virtual bool is_accelerator() const = 0;
00049
00050
00051     /// Return the platform of device
00052     virtual cl::sycl::platform get_platform() const = 0;
00053
00054
00055     /// Query the device for OpenCL info::device info
00056     /** \todo virtual cannot be templated
00057     template <typename T>
00058     virtual T get_info(info::device param) const = 0;
00059     */
00060
00061
00062     /// Specify whether a specific extension is supported on the device.
00063     virtual bool has_extension(const string_class &extension) const = 0;
00064
00065
00066     // Virtual to call the real destructor
00067     virtual ~device() {}
00068 };
00069 };
00070
00071 /// @} to end the execution Doxygen group
00072
00073 }
00074 }
00075 }
00076
00077 /**
00078     # Some Emacs stuff:
00079     ### Local Variables:
00080     ### ispell-local-dictionary: "american"
00081     ### eval: (flyspell-prog-mode)
00082     ### End:
00083 */
00084
00085 #endif // TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_HPP

```

11.49 include/CL/sycl/device.hpp File Reference

```
#include <algorithm>
```

Include dependency graph for device.hpp:

[illegible]

- 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

```

00001 #ifndef TRISYCL_SYCL_DEVICE_HPP
00002 #define TRISYCL_SYCL_DEVICE_HPP
00003
00004 /** \file The OpenCL SYCL device
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 #include <algorithm>
00013 #include <memory>
00014
00015 #ifdef TRISYCL_OPENCL
00016 #include <boost/compute.hpp>
00017 #endif
00018
00019 #include "CL/sycl/detail/default_classes.hpp"
00020
00021 #include "CL/sycl/detail/shared_ptr_implementation.hpp"
00022 #include "CL/sycl/device/detail/host_device.hpp"
00023 #ifdef TRISYCL_OPENCL
00024 #include "CL/sycl/device/detail/opencl_device.hpp"
00025 #endif
00026 #include "CL/sycl/info/device.hpp"
00027 #include "CL/sycl/device_selector.hpp"
00028 #include "CL/sycl/platform.hpp"
00029
00030 namespace cl {
00031 namespace sycl {
00032
00033 class device_selector;
00034 class platform;
00035
00036 /** \addtogroup execution Platforms, contexts, devices and queues
00037     @{
00038 */
00039
00040 /// SYCL device
00041 class device
00042 {
00043     /* Use the underlying device implementation that can be shared in the
00044        SYCL model */
00044     : public detail::shared_ptr_implementation<device, detail::device> {
00045
00046     // The type encapsulating the implementation
00047     using implementation_t =
00048         detail::shared_ptr_implementation<device, detail::device>
00049     ;
00050 public:
00051
00052     // Make the implementation member directly accessible in this class
00053     using implementation_t::implementation;
00054
00055     /// The default constructor uses the SYCL host device
00056     device() : implementation_t {
00057         detail::host_device::instance() } {}
00058
00059 #ifdef TRISYCL_OPENCL
00060     /** Construct a device class instance using cl_device_id of the
00061         OpenCL device
00062
00063         Return synchronous errors via the SYCL exception class.
00064
00065         Retain a reference to the OpenCL device and if this device was
00066         an OpenCL subdevice the device should be released by the caller
00067         when it is no longer needed.
00068     */

```

```

00069     device(cl_device_id device_id)
00070     : device { boost::compute::device { device_id } } {}
00071
00072
00073     /** Construct a device class instance using a boost::compute::device
00074
00075         This is a triSYCL extension for boost::compute interoperation.
00076
00077         Return synchronous errors via the SYCL exception class.
00078     */
00079     device(const boost::compute::device &d)
00080     : implementation_t { detail::openccl_device::instance(d)
00081     } {}
00082 #endif
00083
00084     /** Construct a device class instance using the device selector
00085         provided
00086
00087         Return errors via C++ exception class.
00088
00089         \todo Make it non-explicit in the specification?
00090     */
00091     explicit device(const device_selector &ds) {
00092         auto devices = device::get_devices();
00093         if (devices.empty())
00094             // \todo Put a SYCL exception
00095             throw std::domain_error("No device at all! Internal error...");
00096
00097         /* Find the device with the best score according to the given
00098            device_selector */
00099         auto max = std::max_element(devices.cbegin(), devices.cend(),
00100                                     [&] (const device &d1, const device &d2) {
00101                                         return ds(d1) < ds(d2);
00102                                     });
00103         if (ds(*max) < 0)
00104             // \todo Put a SYCL exception
00105             throw std::domain_error("No device selected because no positive "
00106                                     "device_selector score found");
00107
00108         // Create the current device as a shared copy of the selected one
00109         implementation = max->implementation;
00110     }
00111
00112
00113 #ifndef TRISYCL_OPENCL
00114     /** Return the cl_device_id of the underlying OpenCL platform
00115
00116         Return synchronous errors via the SYCL exception class.
00117
00118         Retain a reference to the returned cl_device_id object. Caller
00119         should release it when finished.
00120
00121         In the case where this is the SYCL host device it will throw an
00122         exception.
00123     */
00124     cl_device_id get() const {
00125         return implementation->get();
00126     }
00127 #endif
00128
00129
00130     /// Return true if the device is the SYCL host device
00131     bool is_host() const {
00132         return implementation->is_host();
00133     }
00134
00135
00136     /// Return true if the device is an OpenCL CPU device
00137     bool is_cpu() const {
00138         return implementation->is_cpu();
00139     }
00140
00141
00142     /// Return true if the device is an OpenCL GPU device
00143     bool is_gpu() const {
00144         return implementation->is_gpu();
00145     }
00146
00147
00148     /// Return true if the device is an OpenCL accelerator device
00149     bool is_accelerator() const {
00150         return implementation->is_accelerator();
00151     }
00152
00153
00154

```

```

00155  /** Return the device_type of a device
00156
00157      \todo Present in Boost.Compute, to be added to the specification
00158  */
00159  info::device_type type() const {
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  }
00172
00173
00174  /** Return the platform of device
00175
00176      Return synchronous errors via the SYCL exception class.
00177  */
00178  platform get_platform() const {
00179      return implementation->get_platform();
00180  }
00181
00182
00183  /** Return a list of all available devices
00184
00185      Return synchronous errors via SYCL exception classes.
00186  */
00187  static vector_class<device>
00188  get_devices(info::device_type device_type =
00189  info::device_type::all)
00189  __attribute__((weak));
00190
00191
00192  /** Query the device for OpenCL info::device info
00193
00194      Return synchronous errors via the SYCL exception class.
00195
00196      \todo
00197  */
00198  template <typename T>
00199  T get_info(info::device param) const {
00200      //return implementation->get_info<Param>(param);
00201  }
00202
00203
00204  /** Query the device for OpenCL info::device info
00205
00206      Return synchronous errors via the SYCL exception class.
00207
00208      \todo
00209  */
00210  template <info::device Param>
00211  auto get_info() const {
00212      // Forward to the version where the info parameter is not a template
00213      //return get_info<typename info::param_traits_t<info::device, Param>>(Param);
00214  }
00215
00216
00217  /// Test if a specific extension is supported on the device
00218  bool has_extension(const string_class &extension) const {
00219      return implementation->has_extension(extension);
00220  }
00221
00222
00223  #ifndef XYZTRISYCL_OPENCL
00224  /** Partition the device into sub devices based upon the properties
00225      provided
00226
00227      Return synchronous errors via SYCL exception classes.
00228
00229      \todo
00230  */
00231  vector_class<device>
00232  create_sub_devices(info::device_partition_type partition_type,
00233      info::device_partition_property partition_property,
00234      info::device_affinity_domain affinity_domain) const {
00235      return implementation->create_sub_devices(partition_type,
00236          partition_property,
00237          affinity_domain);
00238  }
00239  #endif
00240

```



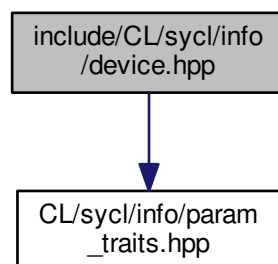
```

00241 };
00242
00243 /// @} to end the Doxygen group
00244
00245 }
00246 }
00247
00248
00249 /* Inject a custom specialization of std::hash to have the buffer
00250    usable into an unordered associative container
00251
00252    \todo Add this to the spec
00253 */
00254 namespace std {
00255
00256 template <> struct hash<cl::sycl::device> {
00257
00258     auto operator()(const cl::sycl::device &d) const {
00259         // Forward the hashing to the implementation
00260         return d.hash();
00261     }
00262 };
00263 };
00264
00265 }
00266
00267 /*
00268     # Some Emacs stuff:
00269     ### Local Variables:
00270     ### ispell-local-dictionary: "american"
00271     ### eval: (flyspell-prog-mode)
00272     ### End:
00273 */
00274
00275 #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:



- 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_width_char`, `cl::sycl::info::device::native_vector_width_short`, `cl::sycl::info::device::native_vector_width_int`,
`cl::sycl::info::device::native_vector_width_long_long`, `cl::sycl::info::device::native_vector_width_float`, `cl::sycl::info::device::native_vector_width_double`, `cl::sycl::info::device::native_vector_width_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_width`, `cl::sycl::info::device::image3d_max_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

```

00001 #ifndef TRISYCL_SYCL_INFO_DEVICE_HPP
00002 #define TRISYCL_SYCL_INFO_DEVICE_HPP
00003
00004 /** \file The OpenCL SYCL device information parameters
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 #include "CL/sycl/info/param_traits.hpp"
00013
00014 namespace cl {
00015 namespace sycl {
00016
00017 /** \addtogroup execution Platforms, contexts, devices and queues
00018     @{
00019 */
00020
00021 namespace info {
00022
00023 /** Type of devices
00024
00025     To be used either to define a device type or to select more
00026     broadly a kind of device
00027
00028     \todo To be moved in the specification from platform to device
00029
00030     \todo Add opencl to the specification
00031
00032     \todo there is no accelerator_selector and custom_accelerator
00033 */
00034 enum class device_type : unsigned int {
00035     cpu,
00036     gpu,
00037     accelerator,
00038     custom,
00039     defaults,
00040     host,
00041     opencl,
00042     all
00043 };
00044
00045
00046 /** Device information descriptors
00047
00048     From specs/latex/headers/deviceInfo.h in the specification
00049
00050     \todo Should be unsigned int?
00051 */
00052 enum class device : int {
00053     device_type,
00054     vendor_id,
00055     max_compute_units,
00056     max_work_item_dimensions,
00057     max_work_item_sizes,
00058     max_work_group_size,
00059     preferred_vector_width_char,
00060     preferred_vector_width_short,
00061     preferred_vector_width_int,
00062     preferred_vector_width_long_long,
00063     preferred_vector_width_float,
00064     preferred_vector_width_double,
00065     preferred_vector_width_half,
00066     native_vector_width_char,
00067     native_vector_width_short,
00068     native_vector_width_int,
00069     native_vector_width_long_long,
00070     native_vector_width_float,
00071     native_vector_width_double,
00072     native_vector_width_half,
00073     max_clock_frequency,

```

```

00074     address_bits,
00075     max_mem_alloc_size,
00076     image_support,
00077     max_read_image_args,
00078     max_write_image_args,
00079     image2d_max_height,
00080     image2d_max_width,
00081     image3d_max_height,
00082     image3d_max_width,
00083     image3d_max_depth,
00084     image_max_buffer_size,
00085     image_max_array_size,
00086     max_samplers,
00087     max_parameter_size,
00088     mem_base_addr_align,
00089     single_fp_config,
00090     double_fp_config,
00091     global_mem_cache_type,
00092     global_mem_cache_line_size,
00093     global_mem_cache_size,
00094     global_mem_size,
00095     max_constant_buffer_size,
00096     max_constant_args,
00097     local_mem_type,
00098     local_mem_size,
00099     error_correction_support,
00100     host_unified_memory,
00101     profiling_timer_resolution,
00102     endian_little,
00103     is_available,
00104     is_compiler_available,
00105     is_linker_available,
00106     execution_capabilities,
00107     queue_properties,
00108     built_in_kernels,
00109     platform,
00110     name,
00111     vendor,
00112     driver_version,
00113     profile,
00114     device_version,
00115     opencl_version,
00116     extensions,
00117     printf_buffer_size,
00118     preferred_interop_user_sync,
00119     parent_device,
00120     partition_max_sub_devices,
00121     partition_properties,
00122     partition_affinity_domain,
00123     partition_type,
00124     reference_count
00125 };
00126
00127 enum class device_partition_property : int {
00128     unsupported,
00129     partition_equally,
00130     partition_by_counts,
00131     partition_by_affinity_domain,
00132     partition_affinity_domain_next_partitionable
00133 };
00134
00135 enum class device_affinity_domain : int {
00136     unsupported,
00137     numa,
00138     L4_cache,
00139     L3_cache,
00140     L2_cache,
00141     next_partitionable
00142 };
00143
00144 enum class device_partition_type : int {
00145     no_partition,
00146     numa,
00147     L4_cache,
00148     L3_cache,
00149     L2_cache,
00150     L1_cache
00151 };
00152
00153 enum class local_mem_type : int {
00154     none,
00155     local,
00156     global
00157 };
00158
00159 enum class fp_config : int {
00160     denorm,

```

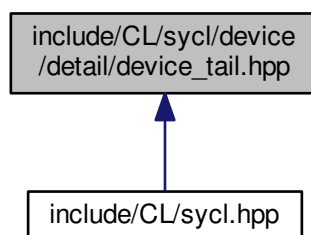
```

00161     inf_nan,
00162     round_to_nearest,
00163     round_to_zero,
00164     round_to_inf,
00165     fma,
00166     correctly_rounded_divide_sqrt,
00167     soft_float
00168 };
00169
00170 enum class global_mem_cache_type : int {
00171     none,
00172     read_only,
00173     write_only
00174 };
00175
00176 enum class device_execution_capabilities : unsigned int {
00177     exec_kernel,
00178     exec_native_kernel
00179 };
00180
00181
00182 using device_fp_config = unsigned int;
00183 using device_exec_capabilities = unsigned int;
00184 using device_queue_properties = unsigned int;
00185
00186
00187 /** Query the return type for get_info() on context stuff
00188
00189     \todo To be implemented, return always void.
00190 */
00191 TRISYCL_INFO_PARAM_TRAITS_ANY_T(info::device, void)
00192 }
00193 }
00194 }
00195 }
00196
00197 /*
00198     # Some Emacs stuff:
00199     ### Local Variables:
00200     ### ispell-local-dictionary: "american"
00201     ### eval: (flyspell-prog-mode)
00202     ### End:
00203 */
00204
00205 #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

```

00001 #ifndef TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_TAIL_HPP
00002 #define TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_TAIL_HPP
00003
00004 /** \file The ending part of of OpenCL SYCL device
00005
00006     This is here to break a dependence between device and device_selector
00007
00008     Ronan at Keryell point FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 namespace cl {
00015 namespace sycl {
00016
00017 /** \addtogroup execution Platforms, contexts, devices and queues
00018     @{
00019 */
00020
00021 /** Return a list of all available devices
00022
00023     Return synchronous errors via SYCL exception classes.
00024 */
00025 vector_class<device>
00026 device::get_devices(info::device_type device_type) {
00027     // Start with the default device
00028     vector_class<device> devices = { {} };
00029
00030 #ifdef TRISYCL_OPENCL
00031     // Then add all the OpenCL devices
00032     for (const auto &d : boost::compute::system::devices())
00033         devices.emplace_back(d);
00034 #endif
00035
00036     // The selected devices
00037     vector_class<device> sd;
00038     device_type_selector s { device_type };
00039
00040     // Return the devices with the good criterion according to the selector
00041     std::copy_if(devices.begin(), devices.end(), std::back_inserter(sd),
00042                 [&](const device &e ) { return s(e) >= 0; });
00043     return sd;
00044 }
00045
00046 /// @} to end the Doxygen group
00047
00048 }
00049 }
00050
00051 /**
00052     # Some Emacs stuff:
00053     ### Local Variables:
00054     ### ispell-local-dictionary: "american"
00055     ### eval: (flyspell-prog-mode)
00056     ### End:
00057 */
00058
00059 #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"

```

[illegible][illegible]

- class `cl::sycl::detail::host_device`
SYCL host device.

- `cl`
The vector type to be used as SYCL vector.
- `cl::sycl`
- `cl::sycl::detail`

11.56 host_device.hpp

```

00001 #ifndef TRISYCL_SYCL_DEVICE_DETAIL_HOST_DEVICE_HPP
00002 #define TRISYCL_SYCL_DEVICE_DETAIL_HOST_DEVICE_HPP
00003
00004 /** \file The OpenCL SYCL host device implementation
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 #include <memory>
00013
00014 #include "CL/sycl/detail/default_classes.hpp"
00015
00016 #include "CL/sycl/detail/singleton.hpp"
00017 #include "CL/sycl/detail/unimplemented.hpp"
00018 #include "CL/sycl/device/detail/device.hpp"
00019 #include "CL/sycl/exception.hpp"
00020 #include "CL/sycl/info/param_traits.hpp"
00021 #include "CL/sycl/platform.hpp"
00022
00023 namespace cl {
00024 namespace sycl {
00025 namespace detail {
00026
00027 /** SYCL host device
00028
00029     \todo The implementation is quite minimal for now. :-)
00030 */
00031 class host_device : public detail::device,
00032                   public detail::singleton<host_device> {
00033 public:
00034
00035 #ifdef TRISYCL_OPENCL
00036     /** Return the cl_device_id of the underlying OpenCL platform
00037
00038         This throws an error since there is no OpenCL device associated
00039         to the host device.
00040     */
00041     cl_device_id get() const override {
00042         throw non_cl_error("The host device has no OpenCL device");
00043     }
00044 #endif
00045
00046     /// Return true since the device is a SYCL host device
00047     bool is_host() const override {
00048         return true;
00049     }
00050
00051     /// Return false since the host device is not an OpenCL CPU device
00052     bool is_cpu() const override {
00053         return false;
00054     }
00055
00056     /// Return false since the host device is not an OpenCL GPU device
00057     bool is_gpu() const override {
00058         return false;
00059     }
00060
00061     /// Return false since the host device is not an OpenCL accelerator device
00062     bool is_accelerator() const override {
00063         return false;
00064     }
00065
00066     /** Return the platform of device
00067
00068         Return synchronous errors via the SYCL exception class.
00069
00070         \todo To be implemented
00071     */
00072     platform get_platform() const override {
00073         detail::unimplemented();
00074         return {};
00075     }
00076
00077 #if 0
00078     /** Query the device for OpenCL info::device info

```



```

graph TD
    CL_sycl_hpp[include/CL/sycl.hpp]
    CL_sycl_device_hpp[include/CL/sycl/device.hpp]
    CL_sycl_device_detail_opengl_device_hpp[include/CL/sycl/device/detail/opengl_device.hpp]
    CL_sycl_platform_hpp[include/CL/sycl/platform.hpp]
    CL_sycl_platform_detail_opengl_platform_hpp[include/CL/sycl/platform/detail/opengl_platform.hpp]
    CL_sycl_device_detail_device_hpp[include/CL/sycl/device/detail/device.hpp]
    CL_sycl_device_detail_host_device_hpp[include/CL/sycl/device/detail/host_device.hpp]
    CL_sycl_context_hpp[include/CL/sycl/context.hpp]
    CL_sycl_queue_hpp[include/CL/sycl/queue.hpp]
    CL_sycl_queue_detail_opengl_queue_hpp[include/CL/sycl/queue/detail/opengl_queue.hpp]
    CL_sycl_queue_detail_host_queue_hpp[include/CL/sycl/queue/detail/host_queue.hpp]
    CL_sycl_kernel_hpp[include/CL/sycl/kernel.hpp]
    CL_sycl_kernel_detail_opengl_kernel_hpp[include/CL/sycl/kernel/detail/opengl_kernel.hpp]
    CL_sycl_kernel_detail_host_kernel_hpp[include/CL/sycl/kernel/detail/host_kernel.hpp]
    CL_sycl_command_group_detail_task_hpp[include/CL/sycl/command_group/detail/task.hpp]
    CL_sycl_buffer_hpp[include/CL/sycl/buffer.hpp]
    CL_sycl_buffer_detail_accessor_hpp[include/CL/sycl/buffer/detail/accessor.hpp]
    CL_sycl_handler_hpp[include/CL/sycl/handler.hpp]
    CL_sycl_static_pipe_hpp[include/CL/sycl/static_pipe.hpp]
    CL_sycl_pipe_hpp[include/CL/sycl/pipe.hpp]
    CL_sycl_buffer_detail_buffer_hpp[include/CL/sycl/buffer/detail/buffer.hpp]
    CL_sycl_buffer_detail_waiter_hpp[include/CL/sycl/buffer/detail/waiter.hpp]
    CL_sycl_queue_detail_queue_hpp[include/CL/sycl/queue/detail/queue.hpp]
    CL_sycl_selector_hpp[include/CL/sycl/device_selector.hpp]

    CL_sycl_hpp --> CL_sycl_device_hpp
    CL_sycl_hpp --> CL_sycl_platform_hpp
    CL_sycl_hpp --> CL_sycl_queue_hpp
    CL_sycl_hpp --> CL_sycl_kernel_hpp
    CL_sycl_hpp --> CL_sycl_command_group_detail_task_hpp
    CL_sycl_hpp --> CL_sycl_buffer_hpp
    CL_sycl_hpp --> CL_sycl_handler_hpp
    CL_sycl_hpp --> CL_sycl_static_pipe_hpp
    CL_sycl_hpp --> CL_sycl_pipe_hpp
    CL_sycl_hpp --> CL_sycl_selector_hpp
    CL_sycl_hpp --> CL_sycl_device_detail_opengl_device_hpp
    CL_sycl_hpp --> CL_sycl_device_detail_device_hpp
    CL_sycl_hpp --> CL_sycl_device_detail_host_device_hpp
    CL_sycl_hpp --> CL_sycl_context_hpp
    CL_sycl_hpp --> CL_sycl_queue_detail_opengl_queue_hpp
    CL_sycl_hpp --> CL_sycl_queue_detail_host_queue_hpp
    CL_sycl_hpp --> CL_sycl_kernel_detail_opengl_kernel_hpp
    CL_sycl_hpp --> CL_sycl_kernel_detail_host_kernel_hpp
    CL_sycl_device_hpp --> CL_sycl_device_detail_opengl_device_hpp
    CL_sycl_device_hpp --> CL_sycl_device_detail_device_hpp
    CL_sycl_device_hpp --> CL_sycl_device_detail_host_device_hpp
    CL_sycl_platform_hpp --> CL_sycl_platform_detail_opengl_platform_hpp
    CL_sycl_platform_hpp --> CL_sycl_queue_detail_opengl_queue_hpp
    CL_sycl_platform_hpp --> CL_sycl_queue_detail_host_queue_hpp
    CL_sycl_platform_hpp --> CL_sycl_kernel_detail_opengl_kernel_hpp
    CL_sycl_platform_hpp --> CL_sycl_kernel_detail_host_kernel_hpp
    CL_sycl_queue_hpp --> CL_sycl_queue_detail_opengl_queue_hpp
    CL_sycl_queue_hpp --> CL_sycl_queue_detail_host_queue_hpp
    CL_sycl_kernel_hpp --> CL_sycl_kernel_detail_opengl_kernel_hpp
    CL_sycl_kernel_hpp --> CL_sycl_kernel_detail_host_kernel_hpp
    CL_sycl_command_group_detail_task_hpp --> CL_sycl_buffer_detail_accessor_hpp
    CL_sycl_buffer_hpp --> CL_sycl_buffer_detail_accessor_hpp
    CL_sycl_handler_hpp --> CL_sycl_buffer_detail_accessor_hpp
    CL_sycl_handler_hpp --> CL_sycl_static_pipe_hpp
    CL_sycl_handler_hpp --> CL_sycl_pipe_hpp
    CL_sycl_buffer_detail_accessor_hpp --> CL_sycl_buffer_detail_buffer_hpp
    CL_sycl_buffer_detail_accessor_hpp --> CL_sycl_buffer_detail_waiter_hpp
    CL_sycl_buffer_detail_accessor_hpp --> CL_sycl_queue_detail_opengl_queue_hpp
    CL_sycl_buffer_detail_accessor_hpp --> CL_sycl_queue_detail_host_queue_hpp
    CL_sycl_buffer_detail_accessor_hpp --> CL_sycl_kernel_detail_opengl_kernel_hpp
    CL_sycl_buffer_detail_accessor_hpp --> CL_sycl_kernel_detail_host_kernel_hpp
    CL_sycl_buffer_detail_accessor_hpp --> CL_sycl_selector_hpp
    CL_sycl_buffer_detail_waiter_hpp --> CL_sycl_queue_detail_opengl_queue_hpp
    CL_sycl_buffer_detail_waiter_hpp --> CL_sycl_queue_detail_host_queue_hpp
    CL_sycl_buffer_detail_waiter_hpp --> CL_sycl_kernel_detail_opengl_kernel_hpp
    CL_sycl_buffer_detail_waiter_hpp --> CL_sycl_kernel_detail_host_kernel_hpp
    CL_sycl_buffer_detail_waiter_hpp --> CL_sycl_selector_hpp
    CL_sycl_queue_detail_opengl_queue_hpp --> CL_sycl_selector_hpp
    CL_sycl_queue_detail_host_queue_hpp --> CL_sycl_selector_hpp
    CL_sycl_kernel_detail_opengl_kernel_hpp --> CL_sycl_selector_hpp
    CL_sycl_kernel_detail_host_kernel_hpp --> CL_sycl_selector_hpp
    CL_sycl_selector_hpp --> CL_sycl_device_detail_opengl_device_hpp
    CL_sycl_selector_hpp --> CL_sycl_device_detail_device_hpp
    CL_sycl_selector_hpp --> CL_sycl_device_detail_host_device_hpp
  
```

- class `cl::sycl::detail::opencl_device`
SYCL OpenCL device.

- `cl`

The vector type to be used as SYCL vector.

- `cl::sycl`
- `cl::sycl::detail`

- `detail::cache< cl_device_id, detail::opengl_device > opengl_device::cache` `cl::sycl::detail::__attribute__` ←
((weak))

11.58 opcnl_device.hpp

```

00001 #ifndef TRISYCL_SYCL_DEVICE_DETAIL_OPENCL_DEVICE_HPP
00002 #define TRISYCL_SYCL_DEVICE_DETAIL_OPENCL_DEVICE_HPP
00003
00004 /** \file The SYCL OpenCL device implementation
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 #include <memory>
00013
00014 #include <boost/compute.hpp>
00015
00016 #include "CL/sycl/detail/default_classes.hpp"
00017
00018 #include "CL/sycl/detail/cache.hpp"
00019 #include "CL/sycl/detail/unimplemented.hpp"
00020 #include "CL/sycl/device/detail/device.hpp"
00021 #include "CL/sycl/exception.hpp"
00022 #include "CL/sycl/info/param_traits.hpp"
00023 #include "CL/sycl/platform.hpp"
00024
00025 namespace cl {
00026 namespace sycl {
00027 namespace detail {
00028
00029 /// SYCL OpenCL device
00030 class opcnl_device : public detail::device {
00031
00032     /// Use the Boost Compute abstraction of the OpenCL device
00033     boost::compute::device d;
00034
00035     /** A cache to always return the same alive device for a given
00036         OpenCL device
00037
00038         C++11 guaranties the static construction is thread-safe
00039     */
00040     static detail::cache<cl_device_id, detail::opcnl_device>
    cache;
00041
00042 public:
00043
00044     /// Return the cl_device_id of the underlying OpenCL device
00045     cl_device_id get() const override {
00046         return d.id();
00047     }
00048
00049
00050     /// Return false since an OpenCL device is not the SYCL host device
00051     bool is_host() const override {
00052         return false;
00053     }
00054
00055
00056     /// Test if the OpenCL is a CPU device
00057     bool is_cpu() const override {
00058         return d.type() == boost::compute::device::cpu;
00059     }
00060
00061
00062     /// Test if the OpenCL is a GPU device
00063     bool is_gpu() const override {
00064         return d.type() == boost::compute::device::gpu;
00065     }
00066
00067
00068     /// Test if the OpenCL is an accelerator device
00069     bool is_accelerator() const override {
00070         return d.type() == boost::compute::device::accelerator;
00071     }
00072
00073
00074     /** Return the platform of device
00075
00076         Return synchronous errors via the SYCL exception class.
00077
00078         \todo To be implemented
00079     */
00080     cl::sycl::platform get_platform() const override {
00081         detail::unimplemented();
00082         return {};
00083     }

```

```

00084
00085 #if 0
00086  /** Query the device for OpenCL info::device info
00087
00088      Return synchronous errors via the SYCL exception class.
00089
00090      \todo To be implemented
00091  */
00092  template <info::device Param>
00093  typename info::param_traits<info::device, Param>::type
00094  get_info() const override {
00095      detail::unimplemented();
00096      return {};
00097  }
00098 #endif
00099
00100  /** Specify whether a specific extension is supported on the device.
00101
00102      \todo To be implemented
00103  */
00104  bool has_extension(const string_class &extension) const override {
00105      detail::unimplemented();
00106      return {};
00107  }
00108
00109
00110  ///// Get a singleton instance of the opengl_device
00111  static std::shared_ptr<opengl_device>
00112  instance(const boost::compute::device &d) {
00113      return cache.get_or_register(d.id(),
00114                                  [&] { return new opengl_device { d }; });
00115  }
00116
00117 private:
00118
00119  /// Only the instance factory can built it
00120  opengl_device(const boost::compute::device &d) : d { d } {}
00121
00122 public:
00123
00124  /// Unregister from the cache on destruction
00125  ~opengl_device() override {
00126      cache.remove(d.id());
00127  }
00128
00129 };
00130
00131  /** Allocate the cache here but since this is a pure-header library,
00132      use a weak symbol so that only one remains when SYCL headers are
00133      used in different compilation units of a program
00134  */
00135  detail::cache<cl_device_id, detail::opengl_device>
00136  opengl_device::cache
00137  __attribute__((weak));
00138 }
00139 }
00140 }
00141
00142  /*
00143      # Some Emacs stuff:
00144      ### Local Variables:
00145      ### ispell-local-dictionary: "american"
00146      ### eval: (flyspell-prog-mode)
00147      ### End:
00148  */
00149
00150 #endif // TRISYCL_SYCL_DEVICE_DETAIL_OPENGL_DEVICE_HPP

```

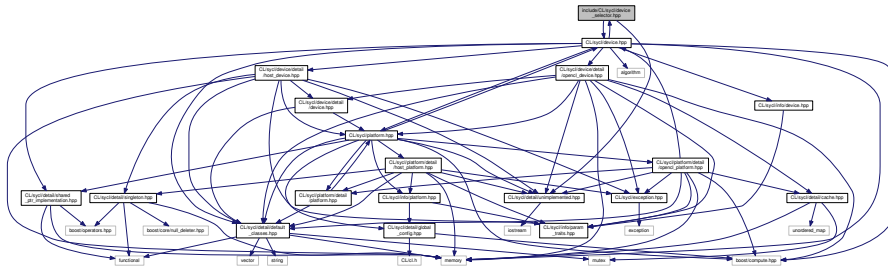
11.59 include/CL/sycl/device_selector.hpp File Reference

```

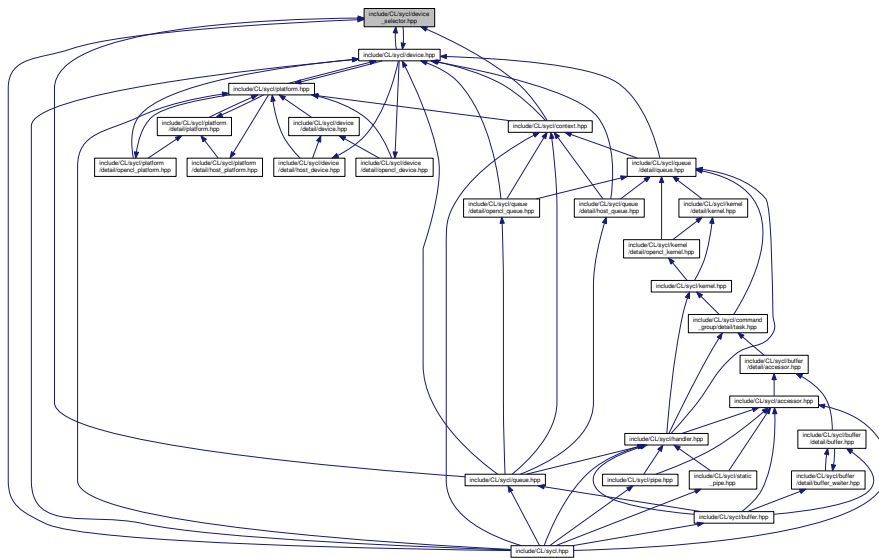
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"

```

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

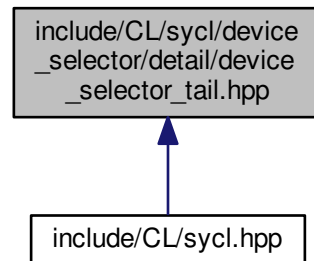
```

00001 #ifndef TRISYCL_SYCL_DEVICE_SELECTOR_HPP
00002 #define TRISYCL_SYCL_DEVICE_SELECTOR_HPP
00003
00004 /** \file The OpenCL SYCL device_selector
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 #include "CL/sycl/detail/unimplemented.hpp"
00013 #include "CL/sycl/device.hpp"
00014
00015 namespace cl {
00016 namespace sycl {
00017
00018 /** \addtogroup execution Platforms, contexts, devices and queues
00019     @{
00020 */
00021
00022 /** The SYCL heuristics to select a device
00023
00024     The device with the highest score is selected
00025 */
00026 class device_selector {
00027
00028 public:
00029
00030     /** Returns a selected device using the functor operator defined in
00031         sub-classes operator()(const device &dev)
00032
00033         \todo Remove this from specification
00034     */
00035     void /* device */ select_device() const {
00036         // return {};
00037     }
00038
00039
00040     /** This pure virtual operator allows the customization of device
00041         selection.
00042
00043         It defines the behavior of the device_selector functor called by
00044         the SYCL runtime on device selection. It returns a "score" for each
00045         device in the system and the highest rated device will be used
00046         by the SYCL runtime.
00047     */
00048     virtual int operator()(const device &dev) const = 0;
00049
00050
00051     /// Virtual destructor so the final destructor can be called if any
00052     virtual ~device_selector() {}
00053
00054 };
00055
00056 /// @} to end the execution Doxygen group
00057
00058 }
00059 }
00060
00061 /**
00062     # Some Emacs stuff:
00063     ### Local Variables:
00064     ### ispell-local-dictionary: "american"
00065     ### eval: (flyspell-prog-mode)
00066     ### End:
00067 */
00068
00069 #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:



Classes

- class [cl::sycl::device_type_selector](#)
A device selector by device_type. [More...](#)
- class [cl::sycl::device_type_name_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_type_name_selector](#)< [info::device_type::defaults](#) >
Devices selected by heuristics of the system.
- using [cl::sycl::gpu_selector](#) = [device_type_name_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_type_name_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_type_name_selector](#)< [info::device_type::host](#) >
Selects the SYCL host CPU device that does not require an OpenCL runtime.

11.62 device_selector_tail.hpp

```

00001 #ifndef TRISYCL_SYCL_DEVICE_SELECTOR_DETAIL_DEVICE_SELECTOR_TAIL_HPP
00002 #define TRISYCL_SYCL_DEVICE_SELECTOR_DETAIL_DEVICE_SELECTOR_TAIL_HPP
00003
00004 /** \file The ending part of of the OpenCL SYCL device_selector
00005
00006     This is here to break a dependence between device and device_selector
00007
00008     \todo Implement lacking SYCL 2.2 selectors
00009
00010     Ronan at Keryell point FR
00011
00012     This file is distributed under the University of Illinois Open Source
00013     License. See LICENSE.TXT for details.
00014 */
00015
00016 namespace cl {
00017 namespace sycl {
00018
00019 /** \addtogroup execution Platforms, contexts, devices and queues
00020     @{
00021 */
00022
00023
00024 /** A device selector by device_type
00025
00026     \todo To be added to the specification
00027 */
00028 class device_type_selector : public device_selector {
00029 private:
00030
00031     /// The device_type to select
00032     info::device_type device_type;
00033
00034     /** Cache the default device to select with the default device
00035         selector.
00036
00037         This is the host device at construction time and remains as is
00038         if there is no openCL device */
00039     device default_device;
00040
00041 public:
00042
00043     device_type_selector(info::device_type device_type)
00044         : device_type { device_type } {
00045         // The default device selection heuristic
00046         if (device_type == info::device_type::defaults) {
00047             auto devices = device::get_devices(
00048                 info::device_type::opencl);
00049             /* If there is an OpenCL device, pick the first one as the
00050                default device, other wise it is the host device */
00051             if (!devices.empty())
00052                 default_device = devices[0];
00053         }
00054     }
00055
00056     // To select only the requested device_type
00057     int operator()(const device &dev) const override {
00058         if (device_type == info::device_type::all)
00059             // All devices fit all
00060             return 1;
00061
00062         if (device_type == info::device_type::defaults)
00063             // Only select the default device
00064             return dev == default_device ? 1 : -1;
00065
00066         if (device_type == info::device_type::opencl)
00067             // For now, any non host device is an OpenCL device
00068             return dev.is_host() ? -1 : 1;
00069
00070         return dev.type() == device_type ? 1 : -1;
00071     }
00072 };
00073
00074
00075 /** Select a device by template device_type parameter
00076
00077     \todo To be added to the specification
00078 */
00079
00080 template <info::device_type DeviceType>
00081 class device_type_name_selector : public
00082     device_type_selector {

```

```

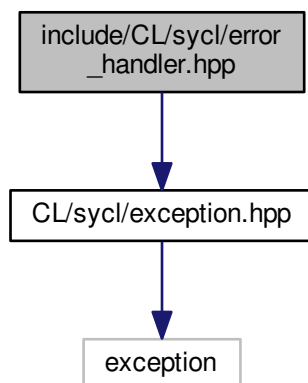
00083 public:
00084
00085     device_type_name_selector() : device_type_selector {
00086         DeviceType {} {}
00087     };
00088
00089 /** Devices selected by heuristics of the system
00090     If no OpenCL device is found then it defaults to the SYCL host device.
00091 */
00092 using default_selector =
00093     device_type_name_selector<info::device_type::defaults>;
00094
00095 /** Select devices according to device type info::device::device_type::gpu
00096     from all the available OpenCL devices.
00097     If no OpenCL GPU device is found the selector fails.
00098     Select the best GPU, if any.
00099 */
00100 using gpu_selector =
00101     device_type_name_selector<info::device_type::gpu>;
00102
00103 /** Select devices according to device type info::device::device_type::cpu
00104     from all the available devices and heuristics
00105     If no OpenCL CPU device is found the selector fails.
00106 */
00107 using cpu_selector =
00108     device_type_name_selector<info::device_type::cpu>;
00109
00110 /** Selects the SYCL host CPU device that does not require an OpenCL
00111     runtime
00112 */
00113 using host_selector =
00114     device_type_name_selector<info::device_type::host>;
00115
00116 // @} to end the execution Doxygen group
00117 }
00118 }
00119
00120 /*
00121     # Some Emacs stuff:
00122     ### Local Variables:
00123     ###   ispell-local-dictionary: "american"
00124     ###   eval: (flyspell-prog-mode)
00125     ### End:
00126 */
00127 #endif // TRISYCL_SYCL_DEVICE_SELECTOR_DETAIL_DEVICE_SELECTOR_TAIL_HPP

```

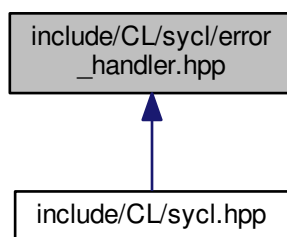
11.63 include/CL/sycl/error_handler.hpp File Reference

```
#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

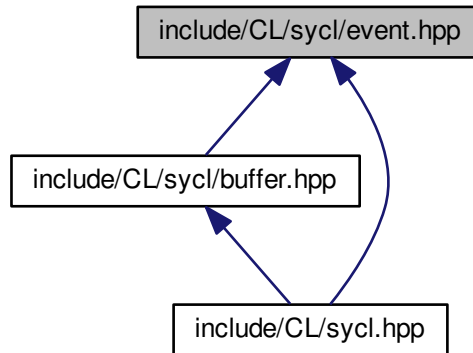
```

00001 #ifndef TRISYCL_SYCL_ERROR_HANDLER_HPP
00002 #define TRISYCL_SYCL_ERROR_HANDLER_HPP
00003
00004 /** \file The OpenCL SYCL error_handler
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 #include "CL/sycl/exception.hpp"
00013
00014 namespace cl {
00015 namespace sycl {
00016
00017 /** \addtogroup error_handling Error handling
00018     @{
00019 */
00020
00021 /** \todo Refactor when updating to latest specification
00022 namespace trisycl {
00023     // Create a default error handler to be used when nothing is specified
00024     struct default_error_handler;
00025 }
00026
00027 /** User supplied error handler to call a user-provided function when an
00028     error happens from a SYCL object that was constructed with this error
00029     handler
00030 */
00031 struct error_handler {
00032     /** The method to define to be called in the case of an error
00033
00034         \todo Add "virtual void" to the specification
00035     */
00036     virtual void report_error(exception &error) = 0;
00037
00038     /** Add a default_handler to be used by default
00039
00040         \todo add this concept to the specification?
00041     */
00042     static trisycl::default_error_handler
00043     default_handler;
00044 };
00045
00046 namespace trisycl {
00047     struct default_error_handler : error_handler {
00048     void report_error(exception &) override {
00049     }
00050     };
00051
00052 // \todo finish initialization
00053 //error_handler::default_handler = nullptr;
00054
00055 /** @} End the error_handling Doxygen group
00056 */
00057 }
00058
00059 # Some Emacs stuff:
00060 ### Local Variables:
00061 ### ispell-local-dictionary: "american"
00062 ### eval: (flyspell-prog-mode)
00063 ### End:
00064 */
00065 #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

```

00001 #ifndef TRISYCL_SYCL_EVENT_HPP
00002 #define TRISYCL_SYCL_EVENT_HPP
00003
00004 /** \file The event class
00005
00006     Ronan at keryell dot FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011 namespace cl {
00012 namespace sycl {
00013
00014 class event {
00015
00016 public:
00017     event() = default;
00018
00019
00020
00021 /** \todo To be implemented */
00022 #if 0
00023     explicit event(cl_event clEvent);
00024
  
```

```

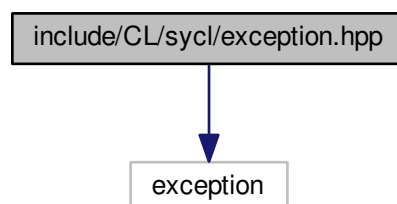
00025     event(const event & rhs);
00026
00027     cl_event get();
00028
00029     vector_class<event> get_wait_list();
00030
00031     void wait();
00032
00033     static void wait(const vector_class<event> &eventList);
00034
00035     void wait_and_throw();
00036
00037     static void wait_and_throw(const vector_class<event> &eventList);
00038
00039     template <info::event param>
00040     typename param_traits<info::event, param>::type get_info() const;
00041
00042     template <info::event_profiling param>
00043     typename param_traits<info::event_profiling,
00044                           param>::type get_profiling_info() const;
00045 #endif
00046 };
00047
00048 }
00049 }
00050
00051 /*
00052     # Some Emacs stuff:
00053     ### Local Variables:
00054     ###   ispell-local-dictionary: "american"
00055     ###   eval: (flyspell-prog-mode)
00056     ### End:
00057 */
00058
00059 #endif // TRISYCL_SYCL_EVENT_HPP

```

11.67 include/CL/sycl/exception.hpp File Reference

#include <exception>

Include dependency graph for exception.hpp:



- 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 enqueueing 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

```

00001 #ifndef TRISYCL_SYCL_EXCEPTION_HPP
00002 #define TRISYCL_SYCL_EXCEPTION_HPP
00003
00004 /** \file The OpenCL SYCL exception
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 #include <exception>
00013
00014 namespace cl {
00015 namespace sycl {
00016
00017 /** \addtogroup error_handling Error handling
00018     @{
00019 */
00020
00021
00022 /** A shared pointer to an exception as in C++ specification
00023
00024     \todo Do we need this instead of reusing directly the one from C++11?
00025 */
00026 using exception_ptr = std::exception_ptr;
00027
00028
00029 /** Exception list to store several exceptions
00030
00031     \todo Do we need to define it in SYCL or can we rely on plain C++17 one?
00032 */
00033 struct exception_list : std::vector<exception_ptr> {
00034     using std::vector<exception_ptr>::vector;
00035 };
00036

```



```

00037 using async_handler = function_class<void, exception_list>
00038 ;
00039
00040 /// Encapsulate a SYCL error information
00041 class exception {
00042     /// The error message to return
00043     string_class message;
00044 public:
00045     /// Construct an exception with a message for internal use
00046     exception(const string_class &message) : message { message } {}
00047
00048     /// Returns a descriptive string for the error, if available
00049     string_class what() const {
00050         return message;
00051     }
00052
00053     /** Returns the context that caused the error
00054
00055         Returns nullptr if not a buffer error.
00056
00057         \todo Cannot return nullptr. Use optional? Use a specific exception type?
00058     */
00059     //context get_context()
00060 };
00061
00062 /// Returns the OpenCL error code encapsulated in the exception
00063 class cl_exception : public exception {
00064 #ifdef TRISYCL_OPENCL
00065     /// The OpenCL error code to return
00066     cl_int cl_code;
00067 public:
00068     /** Construct an exception with a message and OpenCL error code for
00069         internal use */
00070     cl_exception(const string_class &message, cl_int cl_code)
00071         : exception { message }, cl_code { cl_code } {}
00072
00073     // thrown as a result of an OpenCL API error code
00074     cl_int get_cl_code() const {
00075         return cl_code;
00076     }
00077 #endif
00078 };
00079
00080 /// An error stored in an exception_list for asynchronous errors
00081 struct async_exception : exception {
00082     using exception::exception;
00083 };
00084
00085 class runtime_error : public exception {
00086     using exception::exception;
00087 };
00088
00089 /// Error that occurred before or while enqueueing the SYCL kernel
00090 class kernel_error : public runtime_error {
00091     using runtime_error::runtime_error;
00092 };
00093
00094 /// Error regarding the cl::sycl::accessor objects defined
00095 class accessor_error : public runtime_error {
00096     using runtime_error::runtime_error;
00097 };
00098
00099 /// Error regarding the cl::sycl::nd_range specified for the SYCL kernel
00100 class nd_range_error : public runtime_error {
00101     using runtime_error::runtime_error;
00102 };
00103
00104 /// Error regarding associated cl::sycl::event objects
00105 class event_error : public runtime_error {

```

```

00123     using runtime_error::runtime_error;
00124 };
00125
00126
00127 /** Error regarding parameters to the SYCL kernel, it may apply to any
00128     captured parameters to the kernel lambda
00129 */
00130 class invalid_parameter_error : public runtime_error {
00131     using runtime_error::runtime_error;
00132 };
00133
00134
00135 /// The SYCL device will trigger this exception on error
00136 class device_error : public exception {
00137     using exception::exception;
00138 };
00139
00140
00141 /// Error while compiling the SYCL kernel to a SYCL device
00142 class compile_program_error : public device_error {
00143     using device_error::device_error;
00144 };
00145
00146
00147 /// Error while linking the SYCL kernel to a SYCL device
00148 class link_program_error : public device_error {
00149     using device_error::device_error;
00150 };
00151
00152
00153 /// Error regarding any memory objects being used inside the kernel
00154 class invalid_object_error : public device_error {
00155     using device_error::device_error;
00156 };
00157
00158
00159 /// Error on memory allocation on the SYCL device for a SYCL kernel
00160 class memory_allocation_error : public device_error {
00161     using device_error::device_error;
00162 };
00163
00164
00165 /// A failing pipe error will trigger this exception on error
00166 class pipe_error : public runtime_error {
00167     using runtime_error::runtime_error;
00168 };
00169
00170
00171 /// The SYCL platform will trigger this exception on error
00172 class platform_error : public device_error {
00173     using device_error::device_error;
00174 };
00175
00176
00177 /** The SYCL runtime will trigger this error if there is an error when
00178     profiling info is enabled
00179 */
00180 class profiling_error : public device_error {
00181     using device_error::device_error;
00182 };
00183
00184
00185 /** Exception thrown when an optional feature or extension is used in
00186     a kernel but its not available on the device the SYCL kernel is
00187     being enqueued on
00188 */
00189 class feature_not_supported : public device_error {
00190     using device_error::device_error;
00191 };
00192
00193
00194 /** Exception for an OpenCL operation requested in a non OpenCL area
00195
00196     \todo Add to the specification
00197
00198     \todo Clean implementation
00199
00200     \todo Exceptions are named error in C++
00201 */
00202 class non_cl_error : public runtime_error {
00203     using runtime_error::runtime_error;
00204 };
00205
00206
00207 /// @} End the error_handling Doxygen group
00208
00209 }

```

```

00210 }
00211
00212 /*
00213  # Some Emacs stuff:
00214  ### Local Variables:
00215  ### ispell-local-dictionary: "american"
00216  ### eval: (flyspell-prog-mode)
00217  ### End:
00218 */
00219
00220 #endif // TRISYCL_SYCL_EXCEPTION_HPP

```

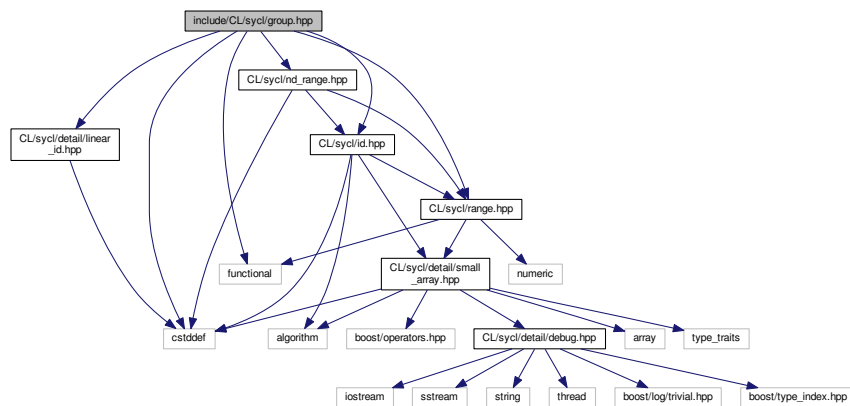
11.69 include/CL/sycl/group.hpp File Reference

```

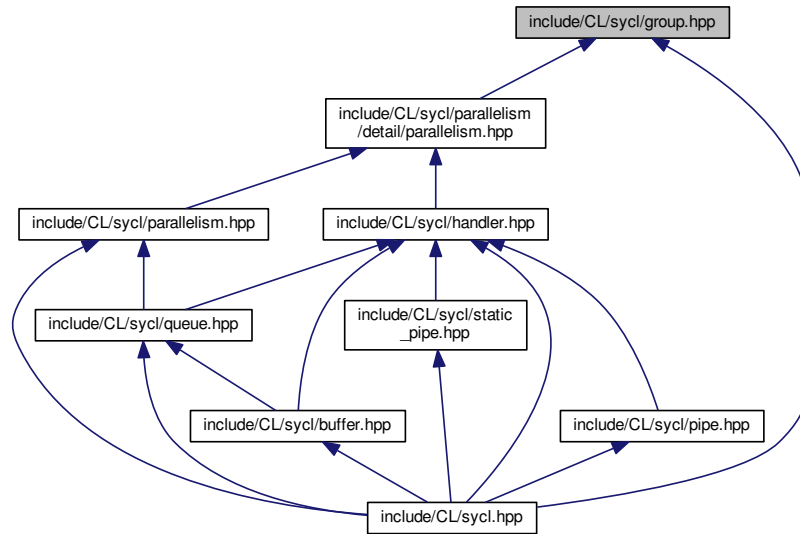
#include <cstdint>
#include <functional>
#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"

```

Include dependency graph for group.hpp:



This graph shows which files directly or indirectly include this file:



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

```

00001 #ifndef TRISYCL_SYCL_GROUP_HPP
00002 #define TRISYCL_SYCL_GROUP_HPP
00003
00004 /** \file The OpenCL SYCL nd_item<>
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 #include <cstdint>
00013 #include <functional>
00014
00015 #include "CL/sycl/detail/linear_id.hpp"
00016 #include "CL/sycl/id.hpp"
00017 #include "CL/sycl/nd_range.hpp"
00018 #include "CL/sycl/range.hpp"
00019
00020 namespace cl {
00021 namespace sycl {
00022
00023 template <std::size_t dims = 1>
00024 struct group;
00025
00026 namespace detail {
00027
00028 template <std::size_t Dimensions = 1, typename ParallelForFuncor>
00029 void parallel_for_workitem(const group<Dimensions> &g,
00030                           ParallelForFuncor f);
00031
00032 }
00033
00034 /** \addtogroup parallelism Expressing parallelism through kernels
00035     @{
00036 */
00037
00038 /** A group index used in a parallel_for_workitem to specify a work_group
00039     */
00040 template <std::size_t dims>
00041 struct group {
00042     /// \todo add this Boost::multi_array or STL concept to the
00043     /// specification?
00044     static constexpr auto dimensionality = dims;
00045 private:
00046     /// The coordinate of the group item
00047     id<dims> group_id;
00048     /// Keep a reference on the nd_range to serve potential query on it
00049     nd_range<dims> ndr;
00050 public:
00051     /** Create a group from an nd_range<> with a 0 id<>
00052         \todo This should be private since it is only used by the trisycl
00053         implementation
00054     */
00055     group(const nd_range<dims> &ndr) : ndr { ndr } {}
00056
00057     /** Create a group from an id and a nd_range<>
00058         \todo This should be private somehow, but it is used by the
00059         validation infrastructure
00060     */
00061     group(const id<dims> &i, const nd_range<dims> &ndr) :
00062         group_id { i }, ndr { ndr } {}
00063
00064     /** To be able to copy and assign group, use default constructors too
00065         \todo Make most of them protected, reserved to implementation
00066     */
00067     group() = default;
00068
00069     /** Return an id representing the index of the group within the nd_range
00070         for every dimension
00071     */
00072     id<dims> get() const { return group_id; }
00073
00074
00075
00076
00077
00078
00079
00080
00081
00082
00083
00084

```

```

00085
00086 /// Return the index of the group in the given dimension
00087 size_t get(int dimension) const { return get()[dimension]; }
00088
00089
00090 /** Return the index of the group in the given dimension within the
00091     nd_range<>
00092
00093     \todo In this implementation it is not const because the group<> is
00094     written in the parallel_for iterators. To fix according to the
00095     specification
00096 */
00097 auto &operator[](int dimension) {
00098     return group_id[dimension];
00099 }
00100
00101
00102 /** Return a range<> representing the dimensions of the current
00103     group
00104
00105     This local range may have been provided by the programmer, or chosen
00106     by the runtime.
00107
00108     \todo Fix this comment and the specification
00109 */
00110 range<dims> get_group_range() const {
00111     return get_nd_range().get_group();
00112 }
00113
00114
00115 /// Return element dimension from the constituent group range
00116 size_t get_group_range(int dimension) const {
00117     return get_group_range()[dimension];
00118 }
00119
00120
00121 /// Get the local range for this work_group
00122 range<dims> get_global_range() const { return get_nd_range().get_global(); }
00123
00124
00125 /// Return element dimension from the constituent global range
00126 size_t get_global_range(int dimension) const {
00127     return get_global_range()[dimension];
00128 }
00129
00130
00131 /** Get the local range for this work_group
00132
00133     \todo Add to the specification
00134 */
00135 range<dims> get_local_range() const { return get_nd_range().get_local(); }
00136
00137
00138 /** Return element dimension from the constituent local range
00139
00140     \todo Add to the specification
00141 */
00142 size_t get_local_range(int dimension) const {
00143     return get_local_range()[dimension];
00144 }
00145
00146
00147 /** Get the offset of the NDRange
00148
00149     \todo Add to the specification
00150 */
00151 id<dims> get_offset() const { return get_nd_range().get_offset(); }
00152
00153
00154 /** Get the offset of the NDRange
00155
00156     \todo Add to the specification
00157 */
00158 size_t get_offset(int dimension) const { return get_offset()[dimension]; }
00159
00160
00161 /// \todo Also provide this access to the current nd_range
00162 nd_range<dims> get_nd_range() const { return ndr; }
00163
00164
00165 /** Get a linearized version of the group ID
00166
00167     */
00168 size_t get_linear() const {
00169     return detail::linear_id(get_group_range(), get());
00170 }
00171

```

```

00172
00173  /** Loop on the work-items inside a work-group
00174
00175  \todo Add this method in the specification
00176  */
00177  void parallel_for_work_item(std::function<void(
nd_item<dimensionality>)> f)
00178  const {
00179      detail::parallel_for_workitem(*this, f);
00180  }
00181
00182
00183  /** Loop on the work-items inside a work-group
00184
00185  \todo Add this method in the specification
00186  */
00187  void parallel_for_work_item(std::function<void(
item<dimensionality>)> f)
00188  const {
00189      auto item_adapter = [=] (nd_item<dimensionality> ndi) {
00190          item<dimensionality> i = ndi.get_item();
00191          f(i);
00192      };
00193      detail::parallel_for_workitem(*this, item_adapter);
00194  }
00195
00196 };
00197
00198 /// @} End the parallelism Doxygen group
00199
00200 }
00201 }
00202
00203 /*
00204  # Some Emacs stuff:
00205  ### Local Variables:
00206  ### ispell-local-dictionary: "american"
00207  ### eval: (flyspell-prog-mode)
00208  ### End:
00209  */
00210
00211 #endif // TRISYCL_SYCL_GROUP_HPP

```

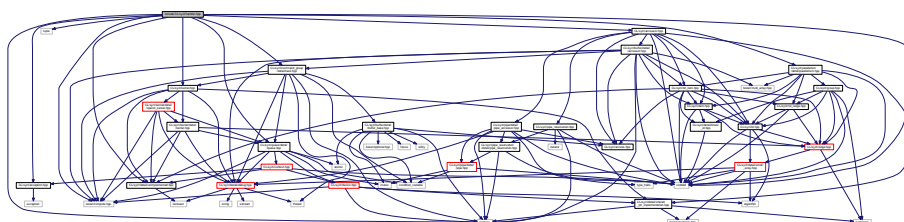
11.71 include/CL/sycl/handler.hpp File Reference

```

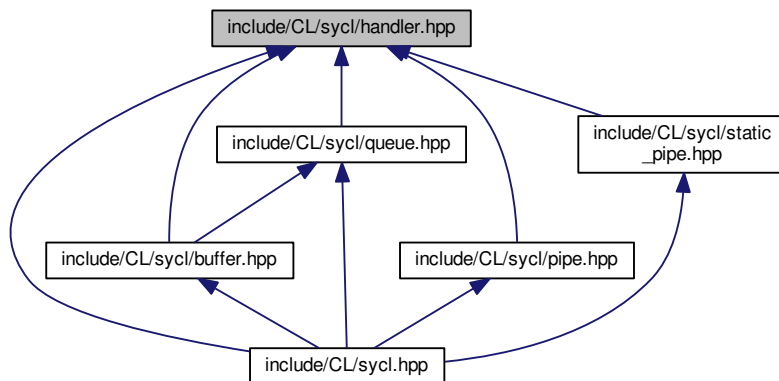
#include <cstdint>
#include <memory>
#include <tuple>
#include <boost/compute.hpp>
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/kernel.hpp"
#include "CL/sycl/parallelism/detail/parallelism.hpp"
#include "CL/sycl/queue/detail/queue.hpp"

```

Include dependency graph for handler.hpp:



This graph shows which files directly or indirectly include this file:



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

<i>global_size</i>	is the full size of the <code>range<></code>
<i>N</i>	dimensionality of the iteration space
<i>f</i>	is the kernel functor to execute
<i>KernelName</i>	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

```
00001 #ifndef TRISYCL_SYCL_HANDLER_HPP
00002 #define TRISYCL_SYCL_HANDLER_HPP
00003
00004 /** \file The OpenCL SYCL command group handler
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 #include <cstdint>
00013 #include <memory>
00014 #include <tuple>
00015
00016 #ifdef TRISYCL_OPENCL
00017 #include <boost/compute.hpp>
00018 #endif
00019
00020 #include "CL/sycl/accessor.hpp"
00021 #include "CL/sycl/command_group/detail/task.hpp"
00022 #include "CL/sycl/detail/unimplemented.hpp"
00023 #include "CL/sycl/exception.hpp"
00024 #include "CL/sycl/kernel.hpp"
00025 #include "CL/sycl/parallelism/detail/parallelism.hpp"
00026 #include "CL/sycl/queue/detail/queue.hpp"
00027
00028 namespace cl {
00029 namespace sycl {
00030
00031 /** \addtogroup execution Platforms, contexts, devices and queues
00032     @{
00033 */
00034
00035 /** Command group handler class
00036
00037     A command group handler object can only be constructed by the SYCL runtime.
00038
00039     All of the accessors defined in the command group scope take as a
```

```

00040     parameter an instance of the command group handler and all the kernel
00041     invocation functions are methods of this class.
00042 */
00043 class handler {
00044 public:
00045     /** Attach the task and accessors to it.
00046     */
00047     std::shared_ptr<detail::task> task;
00048
00049     /* Create a command group handler from the queue detail
00050
00051     The queue detail is used to track kernel completion.
00052
00053     Note that this is an implementation dependent constructor. Normal
00054     users cannot construct handler from scratch.
00055
00056     \todo Make this constructor private
00057     */
00058     handler(const std::shared_ptr<detail::queue> &q) {
00059         // Create a new task for this command_group
00060         task = std::make_shared<detail::task>(q);
00061     }
00062
00063 #ifdef TRISYCL_OPENCL
00064     /** Set kernel arg for an OpenCL kernel which is used through the
00065     SYCL/OpenCL interop interface
00066
00067     The index value specifies which parameter of the OpenCL kernel is
00068     being set and the accessor object, which OpenCL buffer or image is
00069     going to be given as kernel argument.
00070
00071     \todo Update the specification to use a ref && to the accessor instead?
00072
00073     \todo It is not that clean to have set_arg() associated to a
00074     command handler. Rethink the specification?
00075
00076     \todo It seems more logical to have these methods on kernel instead
00077     */
00078     template <typename DataType,
00079               std::size_t Dimensions,
00080               access::mode Mode,
00081               access::target Target = access::target::global_buffer>
00082     void set_arg(int arg_index,
00083                 accessor<DataType, Dimensions, Mode, Target>
00084                 acc_obj) {
00085         /* Before running the kernel, make sure the cl_mem behind this
00086         accessor is up-to-date on the device if needed and pass it to
00087         the kernel.
00088
00089         Explicitly capture task by copy instead of having this captured
00090         by reference and task by reference by side effect */
00091         task->add_prelude([=, task = task] {
00092             acc_obj.implementation->copy_in_cl_buffer();
00093             task->get_kernel().get_boost_compute()
00094                 .set_arg(arg_index, acc_obj.implementation->get_cl_buffer());
00095         });
00096         /* After running the kernel, make sure the cl_mem behind this
00097         accessor is up-to-date on the host if needed */
00098         task->add_postlude([=] {
00099             acc_obj.implementation->copy_back_cl_buffer();
00100         });
00101     }
00102
00103     /** Set kernel args for an OpenCL kernel which is used through the
00104     SYCL/OpenCL interoperability interface
00105
00106     The index value specifies which parameter of the OpenCL kernel is
00107     being set and the accessor object, which OpenCL buffer or image is
00108     going to be given as kernel argument.
00109
00110     \todo It is not that clean to have set_arg() associated to a
00111     command handler. Rethink the specification?
00112
00113     \todo To be implemented
00114     */
00115     template <typename T>
00116     void set_arg(int arg_index, T scalar_value) {
00117         detail::unimplemented();
00118     }
00119
00120
00121
00122
00123
00124

```

```

00125 private:
00126
00127     /// Helper to individually call set_arg() for each argument
00128     template <std::size_t... Is, typename... Ts>
00129     void dispatch_set_arg(std::index_sequence<Is...>, Ts&&... args) {
00130         /// Use an intermediate tuple to ease individual argument access
00131         auto &&t = std::make_tuple(std::forward<Ts>(args)...);
00132         /// Dispatch individual set_arg() for each argument
00133         auto just_to_evaluate = {
00134             0 /*< At least 1 element to deal with empty set_args() *//,
00135             ( set_arg(Is, std::forward<Ts>(std::get<Is>(t))), 0)...
00136         };
00137         /// Remove the warning about unused variable
00138         static_cast<void>(just_to_evaluate);
00139     }
00140
00141 public:
00142
00143     /** Set all kernel args for an OpenCL kernel which is used through the
00144         SYCL/OpenCL interop interface
00145
00146         \todo Update the specification to add this function according to
00147             https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15978 proposal
00148     */
00149     template <typename... Ts>
00150     void set_args(Ts&&... args) {
00151         /* Construct a set of increasing argument index to be able to call
00152            the real set_arg */
00153         dispatch_set_arg(std::make_index_sequence<sizeof...(Ts)>{},
00154             std::forward<Ts>(args)...);
00155     }
00156 #endif
00157
00158
00159     /** Kernel invocation method of a kernel defined as a lambda or
00160         functor. If it is a lambda function or the functor type is globally
00161         visible there is no need for the developer to provide a kernel name type
00162         (typename KernelName) for it, as described in 3.5.3
00163
00164         SYCL single_task launches a computation without parallelism at
00165         launch time.
00166
00167         \param F specify the kernel to be launched as a single_task
00168
00169         \param KernelName is a class type that defines the name to be used for
00170         the underlying kernel
00171     */
00172     template <typename KernelName = std::nullptr_t>
00173     void single_task(std::function<void(void)> F) {
00174         task->schedule(detail::trace_kernel<KernelName>(F));
00175     }
00176
00177
00178     /** SYCL parallel_for launches a data parallel computation with
00179         parallelism specified at launch time by a range<>
00180
00181         Kernel invocation method of a kernel defined as a lambda or functor,
00182         for the specified range and given an id or item for indexing in the
00183         indexing space defined by range.
00184
00185         If it is a lambda function or the if the functor type is globally
00186         visible there is no need for the developer to provide a kernel name
00187         type (typename KernelName) for it, as described in detail in 3.5.3
00188
00189         \param global_size is the full size of the range<>
00190
00191         \param N dimensionality of the iteration space
00192
00193         \param f is the kernel functor to execute
00194
00195         \param KernelName is a class type that defines the name to be used
00196         for the underlying kernel
00197
00198         Unfortunately, to have implicit conversion to work on the range, the
00199         function can not be templated, so instantiate it for all the
00200         dimensions
00201     */
00202     #define TRISYCL_parallel_for_functor_GLOBAL(N)
00203     template <typename KernelName = std::nullptr_t,
00204         typename ParallelForFunctor>
00205     void parallel_for(range<N> global_size,
00206         ParallelForFunctor f) {
00207         task->schedule(detail::trace_kernel<KernelName>([=] {
00208             detail::parallel_for(global_size, f);
00209         }));
00210     }
00211

```

```

00212 TRISYCL_parallel_for_functor_GLOBAL(1)
00213 TRISYCL_parallel_for_functor_GLOBAL(2)
00214 TRISYCL_parallel_for_functor_GLOBAL(3)
00215
00216
00217 /** Kernel invocation method of a kernel defined as a lambda or functor,
00218     for the specified range and offset and given an id or item for
00219     indexing in the indexing space defined by range
00220
00221     If it is a lambda function or the if the functor type is globally
00222     visible there is no need for the developer to provide a kernel name
00223     type (typename KernelName) for it, as described in detail in 3.5.3
00224
00225     \param global_size is the global size of the range<>
00226
00227     \param offset is the offset to be add to the id<> during iteration
00228
00229     \param f is the kernel functor to execute
00230
00231     \param ParallelForFunctor is the kernel functor type
00232
00233     \param KernelName is a class type that defines the name to be used for
00234     the underlying kernel
00235
00236     Unfortunately, to have implicit conversion to work on the range, the
00237     function can not be templated, so instantiate it for all the
00238     dimensions
00239 */
00240 #define TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(N)
00241     template <typename KernelName = std::nullptr_t,
00242             typename ParallelForFunctor>
00243     void parallel_for(range<N> global_size,
00244                     id<N> offset,
00245                     ParallelForFunctor f) {
00246         task->schedule(detail::trace_kernel<KernelName>{[=] {
00247             detail::parallel_for_global_offset(global_size,
00248                                             offset,
00249                                             f);
00250         }});
00251     }
00252
00253 TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(1)
00254 TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(2)
00255 TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(3)
00256
00257
00258 /** Kernel invocation method of a kernel defined as a lambda or functor,
00259     for the specified nd_range and given an nd_item for indexing in the
00260     indexing space defined by the nd_range
00261
00262     If it is a lambda function or the if the functor type is globally
00263     visible there is no need for the developer to provide a kernel name
00264     type (typename KernelName) for it, as described in detail in 3.5.3
00265
00266     \param r defines the iteration space with the work-group layout and
00267     offset
00268
00269     \param Dimensions dimensionality of the iteration space
00270
00271     \param f is the kernel functor to execute
00272
00273     \param ParallelForFunctor is the kernel functor type
00274
00275     \param KernelName is a class type that defines the name to be used for
00276     the underlying kernel
00277 */
00278 template <typename KernelName,
00279         std::size_t Dimensions,
00280         typename ParallelForFunctor>
00281 void parallel_for(nd_range<Dimensions> r, ParallelForFunctor f) {
00282     task->schedule(detail::trace_kernel<KernelName>{[=] {
00283         detail::parallel_for(r, f);
00284     }});
00285 }
00286
00287
00288 /** Hierarchical kernel invocation method of a kernel defined as a
00289     lambda encoding the body of each work-group to launch
00290
00291     May contain multiple kernel built-in parallel_for_work_item
00292     functions representing the execution on each work-item.
00293
00294     Launch num_work_groups work-groups of runtime-defined
00295     size. Described in detail in 3.5.3.
00296
00297     \param r defines the iteration space with the work-group layout and
00298     offset

```

```

00299
00300     \param Dimensions dimensionality of the iteration space
00301
00302     \param f is the kernel functor to execute
00303
00304     \param ParallelForFunctor is the kernel functor type
00305
00306     \param KernelName is a class type that defines the name to be used for
00307     the underlying kernel
00308 */
00309 template <typename KernelName = std::nullptr_t,
00310           std::size_t Dimensions = 1,
00311           typename ParallelForFunctor>
00312 void parallel_for_work_group(nd_range<Dimensions> r,
00313                             ParallelForFunctor f) {
00314     task->schedule(detail::trace_kernel<KernelName>{[=] {
00315         detail::parallel_for_workgroup(r, f); }});
00316 }
00317
00318
00319 /** Kernel invocation method of a kernel defined as pointer to a kernel
00320     object, described in detail in 3.5.3
00321
00322     \todo Add in the spec a version taking a kernel and a functor,
00323     to have host fall-back
00324
00325     \todo To be implemented
00326 */
00327 void single_task(kernel syclKernel) {
00328     detail::unimplemented();
00329 }
00330
00331
00332 /** Kernel invocation method of a kernel defined as a kernel object,
00333     for the specified range and given an id or item for indexing in
00334     the indexing space defined by range, described in detail in
00335     5.4.
00336
00337     \todo Add in the spec a version taking a kernel and a functor,
00338     to have host fall-back
00339 */
00340 #define TRISYCL_ParallelForKernel_RANGE(N)
00341 void parallel_for(range<N> num_work_items,
00342                  kernel sycl_kernel) {
00343     /* For now just use the usual host task system to schedule
00344        manually the OpenCL kernels instead of using OpenCL event-based
00345        scheduling
00346
00347        \todo Move the tracing inside the kernel implementation
00348
00349        \todo Simplify this 2 step ugly interface
00350     */
00351     task->set_kernel(sycl_kernel.implementation);
00352     /* Use an intermediate variable to capture task by copy because
00353        otherwise "this" is captured by reference and havoc with task
00354        just accessing the dead "this". Nasty bug to find... */
00355     task->schedule(detail::trace_kernel<kernel>{[=, t = task] {
00356         sycl_kernel.implementation->parallel_for(t, t->get_queue(),
00357                                                    num_work_items); }});
00358 }
00359
00360 /* Do not use a template parameter since otherwise the parallel_for
00361     functor is selected instead of this one
00362
00363     \todo Clean this
00364 */
00365 TRISYCL_ParallelForKernel_RANGE(1)
00366 TRISYCL_ParallelForKernel_RANGE(2)
00367 TRISYCL_ParallelForKernel_RANGE(3)
00368 #undef TRISYCL_ParallelForKernel_RANGE
00369
00370 /** Kernel invocation method of a kernel defined as pointer to a kernel
00371     object, for the specified nd_range and given an nd_item for indexing
00372     in the indexing space defined by the nd_range, described in detail
00373     in 3.5.3
00374
00375     \todo Add in the spec a version taking a kernel and a functor,
00376     to have host fall-back
00377
00378     \todo To be implemented
00379 */
00380 template <std::size_t Dimensions = 1>
00381 void parallel_for(nd_range<Dimensions>, kernel syclKernel) {
00382     detail::unimplemented();
00383 }
00384
00385 };

```

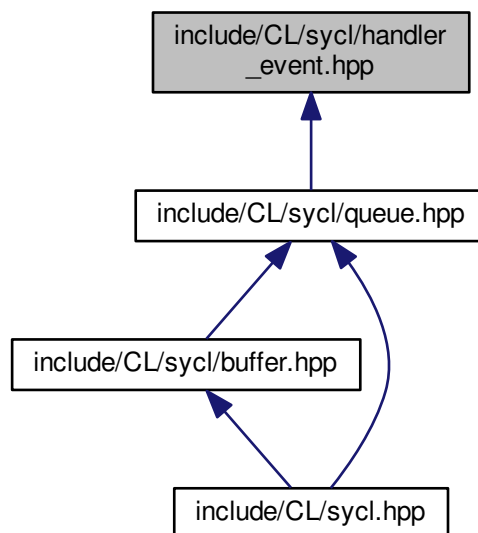
```

00386
00387 namespace detail {
00388
00389 /** Register a buffer as used by a task
00390
00391     This is a proxy function to avoid complicated type recursion.
00392 */
00393 static std::shared_ptr<detail::task>
00394 add_buffer_to_task(handler *command_group_handler,
00395                  std::shared_ptr<detail::buffer_base> b,
00396                  bool is_write_mode) {
00397     command_group_handler->task->add_buffer(b, is_write_mode);
00398     return command_group_handler->task;
00399 }
00400
00401 }
00402
00403 /// @} End the execution Doxygen group
00404
00405 }
00406 }
00407
00408 /*
00409     # Some Emacs stuff:
00410     ### Local Variables:
00411     ### ispell-local-dictionary: "american"
00412     ### eval: (flyspell-prog-mode)
00413     ### End:
00414 */
00415
00416 #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:



Classes

- class [handler_event](#)
Handler event.

11.74 handler_event.hpp

```

00001 #ifndef TRISYCL_SYCL_HANDLER_EVENT_HPP
00002 #define TRISYCL_SYCL_HANDLER_EVENT_HPP
00003
00004 /** \file The handler event
00005
00006     Implement parallel constructions to launch kernels
00007
00008     Ronan at keryell dot FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 /** \todo To be implemented */
00015 /** Handler event
00016
00017     \todo To be implemented
00018 */
00019 class handler_event {
00020 /*
00021 public:
00022     event get_kernel() const;
00023     event get_complete() const;
00024     event get_end() const;
00025 */
00026 };
00027
00028
00029 /*
00030 # Some Emacs stuff:
00031 ### Local Variables:
00032 ### ispell-local-dictionary: "american"
00033 ### eval: (flyspell-prog-mode)
00034 ### End:
00035 */
00036
00037 #endif // TRISYCL_SYCL_HANDLER_EVENT_HPP

```

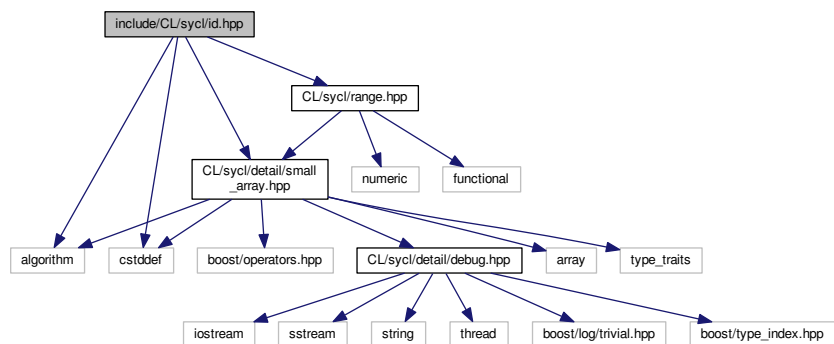
11.75 include/CL/sycl/id.hpp File Reference

```

#include <algorithm>
#include <cstdint>
#include "CL/sycl/detail/small_array.hpp"
#include "CL/sycl/range.hpp"

```

Include dependency graph for id.hpp:




```

graph TD
    CLibSycl_hpp[include/CLibSycl.hpp]
    CLibSycl_item_hpp[include/CL/sycl/item.hpp]
    CLibSycl_ind_range_hpp[include/CL/sycl/ind_range.hpp]
    CLibSycl_ind_item_hpp[include/CL/sycl/ind_item.hpp]
    CLibSycl_group_hpp[include/CL/sycl/group.hpp]
    CLibSycl_parallelism_detail_hpp[include/CL/sycl/parallelism/detail/parallelism.hpp]
    CLibSycl_parallelism_hpp[include/CL/sycl/parallelism.hpp]
    CLibSycl_buffer_detail_hpp[include/CL/sycl/buffer/detail/buffer.hpp]
    CLibSycl_queue_hpp[include/CL/sycl/queue.hpp]
    CLibSycl_pipe_hpp[include/CL/sycl/pipe.hpp]
    CLibSycl_handler_hpp[include/CL/sycl/handler.hpp]
    CLibSycl_accesssor_hpp[include/CL/sycl/accesssor.hpp]
    CLibSycl_buffer_hpp[include/CL/sycl/buffer.hpp]
    CLibSycl_static_pipe_hpp[include/CL/sycl/static_pipe.hpp]
    CLibSycl_detail_buffer_waiter_hpp[include/CL/sycl/buffer/detail/buffer_waiter.hpp]

    CLibSycl_hpp --> CLibSycl_item_hpp
    CLibSycl_hpp --> CLibSycl_ind_range_hpp
    CLibSycl_hpp --> CLibSycl_ind_item_hpp
    CLibSycl_hpp --> CLibSycl_group_hpp
    CLibSycl_hpp --> CLibSycl_parallelism_detail_hpp
    CLibSycl_hpp --> CLibSycl_parallelism_hpp
    CLibSycl_hpp --> CLibSycl_buffer_detail_hpp
    CLibSycl_hpp --> CLibSycl_queue_hpp
    CLibSycl_hpp --> CLibSycl_pipe_hpp
    CLibSycl_hpp --> CLibSycl_handler_hpp
    CLibSycl_hpp --> CLibSycl_accesssor_hpp
    CLibSycl_hpp --> CLibSycl_buffer_hpp
    CLibSycl_hpp --> CLibSycl_static_pipe_hpp
    CLibSycl_hpp --> CLibSycl_detail_buffer_waiter_hpp

    CLibSycl_item_hpp --> CLibSycl_ind_range_hpp
    CLibSycl_item_hpp --> CLibSycl_ind_item_hpp
    CLibSycl_ind_range_hpp --> CLibSycl_ind_item_hpp
    CLibSycl_group_hpp --> CLibSycl_parallelism_detail_hpp
    CLibSycl_parallelism_detail_hpp --> CLibSycl_parallelism_hpp
    CLibSycl_parallelism_hpp --> CLibSycl_buffer_detail_hpp
    CLibSycl_buffer_detail_hpp --> CLibSycl_queue_hpp
    CLibSycl_queue_hpp --> CLibSycl_pipe_hpp
    CLibSycl_pipe_hpp --> CLibSycl_handler_hpp
    CLibSycl_handler_hpp --> CLibSycl_accesssor_hpp
    CLibSycl_accesssor_hpp --> CLibSycl_buffer_hpp
    CLibSycl_buffer_hpp --> CLibSycl_static_pipe_hpp
    CLibSycl_static_pipe_hpp --> CLibSycl_detail_buffer_waiter_hpp
    CLibSycl_detail_buffer_waiter_hpp --> CLibSycl_queue_hpp
  
```

- 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...](#)

- `cl`
The vector type to be used as SYCL vector.
- `cl::sycl`

- 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

```

00001 #ifndef TRISYCL_SYCL_ID_HPP
00002 #define TRISYCL_SYCL_ID_HPP
00003
00004 /** \file The OpenCL SYCL id<>
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 #include <algorithm>
00013 #include <cstdint>
00014
00015 #include "CL/sycl/detail/small_array.hpp"
00016 #include "CL/sycl/range.hpp"
00017
00018 namespace cl {
00019 namespace sycl {
00020
00021 template <std::size_t dims> class item;
00022
00023 /** \addtogroup parallelism Expressing parallelism through kernels
00024     @{
00025 */
00026
00027 /** Define a multi-dimensional index, used for example to locate a work
00028     item
00029 */
00030 template <std::size_t dims = 1>
00031 class id : public detail::small_array_123<std::size_t, id<dims>, dims> {
00032
00033 public:
00034
00035     // Inherit from all the constructors
00036     using detail::small_array_123<std::size_t,
00037                                   id<dims>,
00038                                   dims>::small_array_123;
00039
00040
00041     /// Construct an id from the dimensions of a range
00042     id(const range<dims> &range_size)
00043     /** Use the fact we have a constructor of a small_array from a another
00044         kind of small_array
00045         */
00046     : detail::small_array_123<std::size_t, id<dims>, dims> { range_size } {}
00047
00048
00049     /// Construct an id from an item global_id
00050     id(const item<dims> &rhs)
00051     : detail::small_array_123<std::size_t, id<dims>, dims>
00052     { rhs.get() }
00053     {}
00054
00055     /// Keep other constructors
00056     id() = default;
00057 };
00058
00059
00060
00061 /** Implement a make_id to construct an id<> of the right dimension with
00062     implicit conversion from an initializer list for example.
00063
00064     Cannot use a template on the number of dimensions because the implicit
00065     conversion would not be tried. */
00066 inline auto make_id(id<1> i) { return i; }
00067 inline auto make_id(id<2> i) { return i; }
00068 inline auto make_id(id<3> i) { return i; }
00069
00070
00071 /** Construct an id<> from a function call with arguments, like
00072     make_id(1, 2, 3) */
00073 template<typename... BasicType>
00074 auto make_id(BasicType... Args) {
00075     // Call constructor directly to allow narrowing
00076     return id<sizeof...(Args)>(Args...);
00077 }
00078
00079 /// @} End the parallelism Doxygen group
00080
00081 }
00082 }
00083
00084 /*

```

```

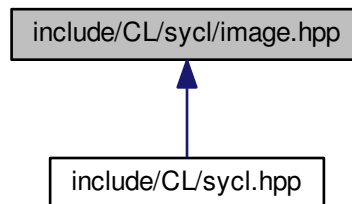
00085      # Some Emacs stuff:
00086      ### Local Variables:
00087      ### ispell-local-dictionary: "american"
00088      ### eval: (flyspell-prog-mode)
00089      ### End:
00090  */
00091
00092 #endif // TRISYCL_SYCL_ID_HPP

```

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.

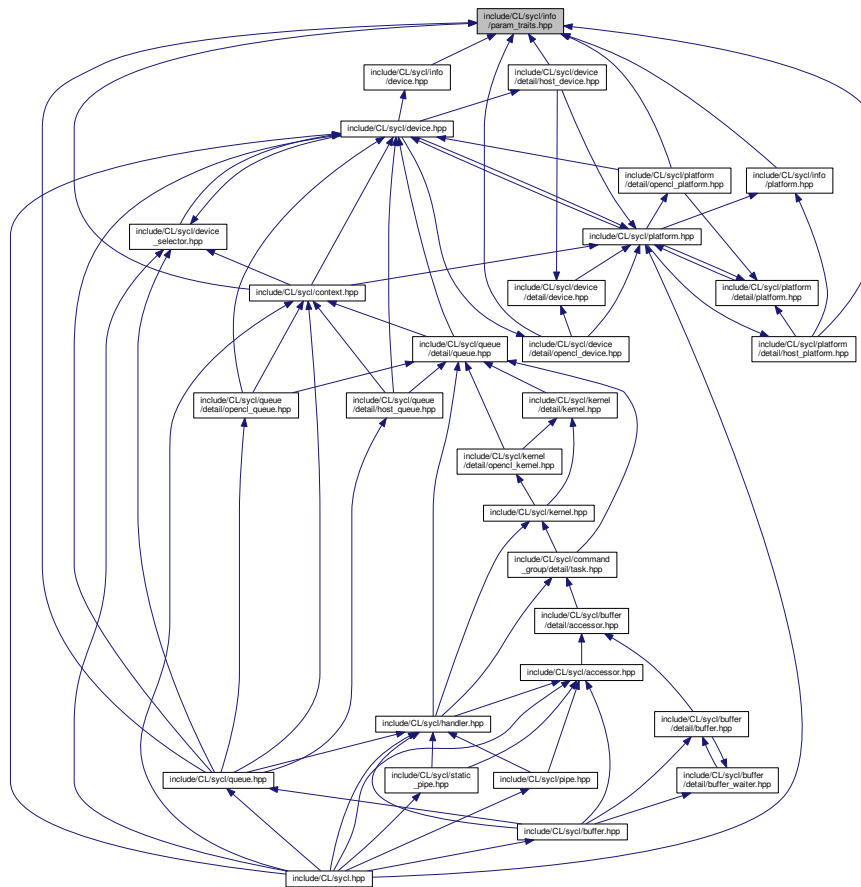
Definition in file [image.hpp](#).

11.78 image.hpp

```
00001 #ifndef TRISYCL_SYCL_IMAGE_HPP
00002 #define TRISYCL_SYCL_IMAGE_HPP
00003
00004 /** \file
00005
00006     OpenCL SYCL image class
00007
00008     Ronan at Keryell point FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 namespace cl {
00015 namespace sycl {
00016
00017 /** \addtogroup data
00018
00019     @{
00020 */
00021
00022 /// \todo implement image
00023 template <std::size_t dimensions> struct image;
00024
00025
00026 /// @} End the data Doxygen group
00027
00028
00029 }
00030 }
00031
00032 /*
00033     # Some Emacs stuff:
00034     ### Local Variables:
00035     ### ispell-local-dictionary: "american"
00036     ### eval: (flyspell-prog-mode)
00037     ### End:
00038 */
00039
00040 #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:



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
```

```

00035 */
00036 #define TRISYCL_INFO_PARAM_TRAITS(VALUE, RETURN_TYPE)      \
00037     template <>                                           \
00038     struct param_traits<decltype(VALUE), VALUE> {         \
00039         using type = RETURN_TYPE;                         \
00040     };                                                     \
00041 }                                                         \
00042 }                                                         \
00043 }                                                         \
00044 }                                                         \
00045 }                                                         \
00046 /*
00047     # Some Emacs stuff:
00048     ### Local Variables:
00049     ### ispell-local-dictionary: "american"
00050     ### eval: (flyspell-prog-mode)
00051     ### End:
00052 */
00053
00054 #endif // TRISYCL_SYCL_INFO_PARAM_TRAITS_HPP

```

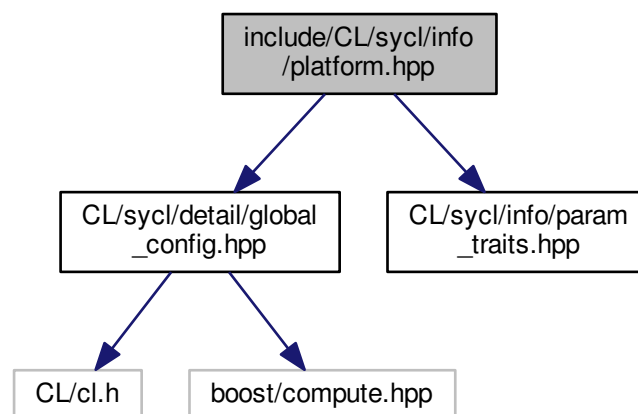
11.81 include/CL/sycl/info/platform.hpp File Reference

```

#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/info/param_traits.hpp"

```

Include dependency graph for platform.hpp:



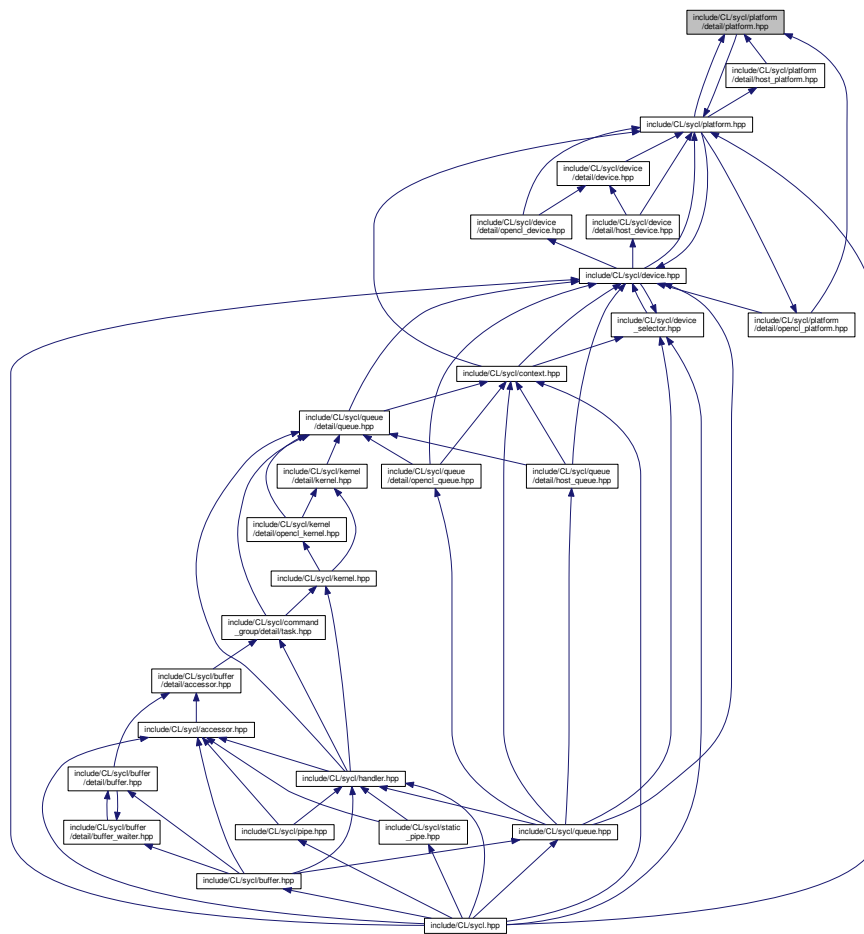
11.82 platform.hpp

```

00001 #ifndef TRISYCL_SYCL_INFO_PLATFORM_HPP
00002 #define TRISYCL_SYCL_INFO_PLATFORM_HPP
00003
00004 /** \file The OpenCL SYCL platform information parameters
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 #include "CL/sycl/detail/global_config.hpp"
00013 #include "CL/sycl/info/param_traits.hpp"
00014
00015 namespace cl {
00016 namespace sycl {
00017
00018 /** \addtogroup execution Platforms, contexts, devices and queues
00019     @{
00020 */
00021 namespace info {
00022
00023 /** Platform information descriptors
00024
00025     A SYCL platform can be queried for all of the following information
00026     using the get_info function.
00027
00028     In this implementation, the values are mapped to OpenCL values to
00029     avoid further remapping later when OpenCL is used
00030 */
00031 enum class platform : unsigned int {
00032     /** Returns the profile name (as a string_class) supported by the
00033         implementation.
00034
00035         Can be either FULL PROFILE or EMBEDDED PROFILE.
00036     */
00037     profile TRISYCL_SKIP_OPENCL(= CL_PLATFORM_PROFILE),
00038
00039     /** Returns the OpenCL software driver version string in the form major
00040         number.minor number (as a string_class)
00041     */
00042     version TRISYCL_SKIP_OPENCL(= CL_PLATFORM_VERSION),
00043
00044     /** Returns the name of the platform (as a string_class)
00045     */
00046     name TRISYCL_SKIP_OPENCL(= CL_PLATFORM_NAME),
00047
00048     /** Returns the string provided by the platform vendor (as a string_class)
00049     */
00050     vendor TRISYCL_SKIP_OPENCL(= CL_PLATFORM_VENDOR),
00051
00052     /** Returns a space-separated list of extension names supported by the
00053         platform (as a string_class)
00054     */
00055     extensions TRISYCL_SKIP_OPENCL(= CL_PLATFORM_EXTENSIONS),
00056
00057     #if CL_SYCL_LANGUAGE_VERSION >= 220 && defined(CL_VERSION_2_1)
00058     /** Returns the resolution of the host timer in nanoseconds as used by
00059         clGetDeviceAndHostTimer
00060     */
00061     host_timer_resolution
00062         TRISYCL_SKIP_OPENCL(= CL_PLATFORM_HOST_TIMER_RESOLUTION)
00063     #endif
00064 };
00065
00066 /** Query the return type for get_info() on platform parameter type
00067
00068     This defines the meta-function
00069     \code
00070     param_traits<info::platform x, string_class>::type == string_class
00071     \endcode
00072
00073     for all x, which means that get_info() returns always a string_class
00074     when asked about platform info.
00075 */
00076 TRISYCL_INFO_PARAM_TRAITS_ANY_T(info::platform,
00077     string_class)
00078
00079 #if CL_SYCL_LANGUAGE_VERSION >= 220 && defined(CL_VERSION_2_1)
00080 /// get_info<host_timer_resolution>() return a cl_ulong
00081 #ifdef TRISYCL_OPENCL
00082 TRISYCL_INFO_PARAM_TRAITS(info::platform::host_timer_resolution, cl_ulong)
00083 #else

```


This graph shows which files directly or indirectly include this file:



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

```
00001 #ifndef TRISYCL_SYCL_PLATFORM_DETAIL_PLATFORM_HPP
00002 #define TRISYCL_SYCL_PLATFORM_DETAIL_PLATFORM_HPP
00003
00004 /** \file The OpenCL SYCL abstract platform
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 #include "CL/sycl/detail/default_classes.hpp"
00013
00014 #include "CL/sycl/platform.hpp"
00015
00016 namespace cl {
00017 namespace sycl {
00018 namespace detail {
00019
00020 /** \addtogroup execution Platforms, contexts, devices and queues
00021     @{
00022 */
00023
00024 /// An abstract class representing various models of SYCL platforms
00025 class platform {
00026
00027 public:
00028
00029 #ifdef TRISYCL_OPENCL
00030     /// Return the cl_platform_id of the underlying OpenCL platform
00031     virtual cl_platform_id get() const = 0;
00032 #endif
00033
00034     /// Return true if the platform is a SYCL host platform
00035     virtual bool is_host() const = 0;
00036
00037     /// Query the platform for OpenCL string info::platform info
00038     virtual string_class get_info_string(info::platform param) const
00039     = 0;
00040
00041     /// Specify whether a specific extension is supported on the platform.
00042     virtual bool has_extension(const string_class &extension) const = 0;
00043
00044     // Virtual to call the real destructor
00045     virtual ~platform() {}
00046
00047 };
00048
00049 /// @} to end the execution Doxygen group
00050
00051
00052
00053
00054
00055
00056
00057
00058 /*
00059     # Some Emacs stuff:
00060     ### Local Variables:
00061     ###   ispell-local-dictionary: "american"
00062     ###   eval: (flyspell-prog-mode)
00063     ### End:
00064 */
00065
00066 #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"

```

[illegible]

- 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

```

00001 #ifndef TRISYCL_SYCL_PLATFORM_HPP
00002 #define TRISYCL_SYCL_PLATFORM_HPP
00003
00004 /** \file The OpenCL SYCL platform
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 #ifdef TRISYCL_OPENCL
00013 #include <boost/compute.hpp>
00014 #endif
00015
00016 #include "CL/sycl/detail/default_classes.hpp"
00017 #include "CL/sycl/detail/global_config.hpp"
00018
00019 #include "CL/sycl/detail/shared_ptr_implementation.hpp"
00020 #include "CL/sycl/detail/unimplemented.hpp"
00021 #include "CL/sycl/device.hpp"
00022 #include "CL/sycl/platform/detail/host_platform.hpp"
00023 #ifdef TRISYCL_OPENCL
00024 #include "CL/sycl/platform/detail/opengl_platform.hpp"
00025 #endif
00026 #include "CL/sycl/platform/detail/platform.hpp"
00027 #include "CL/sycl/info/platform.hpp"
00028
00029 namespace cl {
00030 namespace sycl {
00031
00032     class device_selector;
00033     class device;
00034
00035     /** \addtogroup execution Platforms, contexts, devices and queues
00036         @{
00037     */
00038
00039     /** Abstract the OpenCL platform
00040
00041         \todo triSYCL Implementation
00042     */
00043     class platform
00044     /* Use the underlying platform implementation that can be shared in the
00045        SYCL model */
00046     : public detail::shared_ptr_implementation<platform, detail::platform> {
00047
00048     // The type encapsulating the implementation
00049     using implementation_t =
00050         detail::shared_ptr_implementation<platform, detail::platform>
00051     ;
00052
00053     // Make the implementation member directly accessible in this class
00054     using implementation_t::implementation;
00055
00056     public:
00057
00058     /** Default constructor for platform which is the host platform
00059
00060         Returns errors via the SYCL exception class.
00061     */
00062     platform() : implementation_t {
00063         detail::host_platform::instance() } {}
00064
00065 #ifdef TRISYCL_OPENCL
00066     /** Construct a platform class instance using cl_platform_id of the
00067         OpenCL device
00068     */

```

```

00069         Return synchronous errors via the SYCL exception class.
00070
00071         Retain a reference to the OpenCL platform.
00072     */
00073     platform(cl_platform_id platform_id)
00074         : platform { boost::compute::platform { platform_id } } {}
00075
00076
00077     /** Construct a platform class instance using a boost::compute::platform
00078
00079         This is a triSYCL extension for boost::compute interoperation.
00080
00081         Return synchronous errors via the SYCL exception class.
00082     */
00083     platform(const boost::compute::platform &p)
00084         : implementation_t { detail::opengl_platform::instance
00085         (p) } {}
00086 #endif
00087
00088     /** Construct a platform object from the device selected by a device
00089         selector of the user's choice
00090
00091         Returns errors via the SYCL exception class.
00092     */
00093     explicit platform(const device_selector &dev_selector) {
00094         detail::unimplemented();
00095     }
00096
00097
00098 #ifndef TRISYCL_OPENGL
00099     /** Returns the cl_platform_id of the underlying OpenCL platform
00100
00101         If the platform is not a valid OpenCL platform, for example if it is
00102         the SYCL host, an exception is thrown
00103
00104         \todo Define a SYCL exception for this
00105     */
00106     cl_platform_id get() const {
00107         return implementation->get();
00108     }
00109 #endif
00110
00111
00112     /// Get the list of all the platforms available to the application
00113     static vector_class<platform> get_platforms() {
00114         // Start with the default platform
00115         vector_class<platform> platforms { {} };
00116
00117 #ifndef TRISYCL_OPENGL
00118         // Then add all the OpenCL platforms
00119         for (const auto &d : boost::compute::system::platforms())
00120             platforms.emplace_back(d);
00121 #endif
00122
00123         return platforms;
00124     }
00125
00126 #if 0
00127     /** Returns all the available devices for this platform, of type device
00128         type, which is defaulted to info::device_type::all
00129
00130         By default returns all the devices.
00131
00132         \todo To be implemented
00133     */
00134     vector_class<device>
00135     get_devices(info::device_type device_type =
00136     info::device_type::all) const {
00137         detail::unimplemented();
00138         return {};
00139     }
00140 #endif
00141
00142     /** Get the OpenCL information about the requested parameter
00143
00144         \todo Add to the specification
00145     */
00146     template <typename ReturnT>
00147     ReturnT get_info(info::platform param) const {
00148         // Only strings are needed here
00149         return implementation->get_info_string(param);
00150     }
00151
00152
00153     /// Get the OpenCL information about the requested template parameter

```

```

00154     template <info::platform Param>
00155     typename info::param_traits<info::platform, Param>::type
00156     get_info() const {
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,
                                Param>::type>(Param);
00162     }
00163 }
00164
00165
00166     /// Test if an extension is available on the platform
00167     bool has_extension(const string_class &extension) const {
00168         return implementation->has_extension(extension);
00169     }
00170
00171
00172     /// Test if this platform is a host platform
00173     bool is_host() const {
00174         return implementation->is_host();
00175     }
00176
00177 };
00178
00179 /// @} to end the execution Doxygen group
00180
00181 }
00182 }
00183
00184
00185 /* Inject a custom specialization of std::hash to have the buffer
00186    usable into an unordered associative container
00187
00188    \todo Add this to the spec
00189 */
00190 namespace std {
00191
00192     template <> struct hash<cl::sycl::platform> {
00193
00194         auto operator()(const cl::sycl::platform &p) const {
00195             // Forward the hashing to the implementation
00196             return p.hash();
00197         }
00198
00199     };
00200
00201 }
00202
00203 /*
00204     # Some Emacs stuff:
00205     ### Local Variables:
00206     ###   ispell-local-dictionary: "american"
00207     ###   eval: (flyspell-prog-mode)
00208     ### End:
00209 */
00210
00211 #endif // TRISYCL_SYCL_PLATFORM_HPP

```

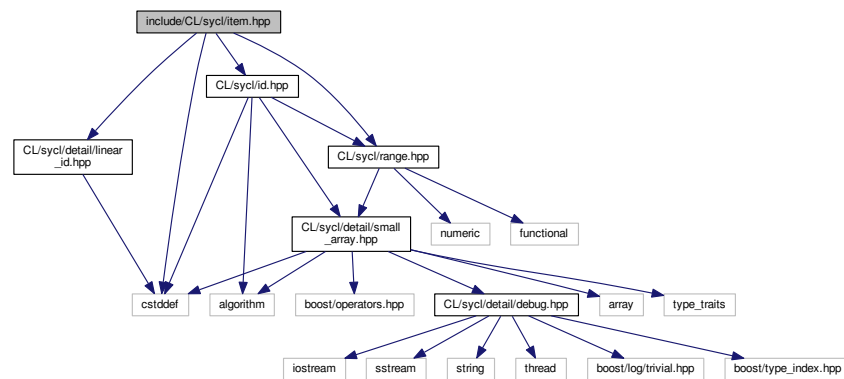
11.87 include/CL/sycl/item.hpp File Reference

```

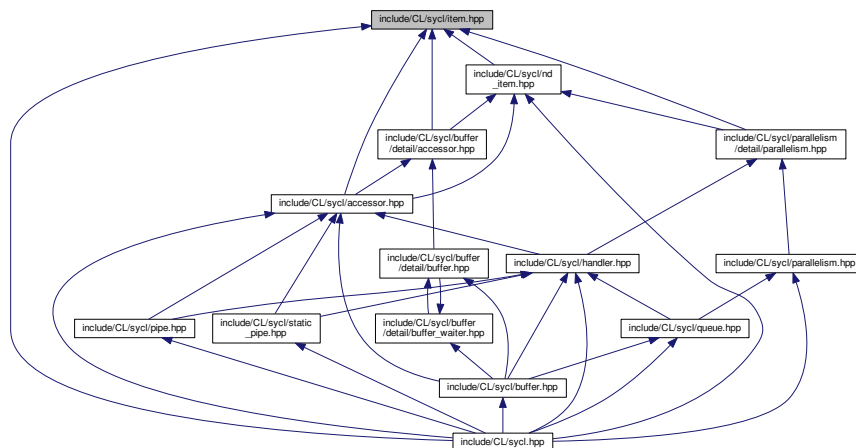
#include <cstdint>
#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/range.hpp"

```


Include dependency graph for item.hpp:



This graph shows which files directly or indirectly include this file:



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

```

00001 #ifndef TRISYCL_SYCL_ITEM_HPP
00002 #define TRISYCL_SYCL_ITEM_HPP
00003
00004 /** \file The OpenCL SYCL item<>
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 #include <cstdint>
00013
00014 #include "CL/sycl/detail/linear_id.hpp"
00015 #include "CL/sycl/id.hpp"
00016 #include "CL/sycl/range.hpp"
00017
00018 namespace cl {
00019 namespace sycl {
00020
00021 /** \addtogroup parallelism Expressing parallelism through kernels
00022     @{
00023 */
00024
00025 /** A SYCL item stores information on a work-item with some more context
00026     such as the definition range and offset.
00027 */
00028 template <std::size_t dims = 1>
00029 class item {
00030
00031 public:
00032     /// \todo add this Boost::multi_array or STL concept to the
00033     /// specification?
00034     static constexpr auto dimensionality = dims;
00035
00036 private:
00037     range<dims> global_range;
00038     id<dims> global_index;
00039     id<dims> offset;
00040
00041 public:
00042     /** Create an item from a local size and an optional offset
00043
00044         This constructor is used by the trisycl implementation and the
00045         non-regression testing.
00046     */
00047     item(range<dims> global_size,
00048          id<dims> global_index,
00049          id<dims> offset = {}) :
00050         global_range { global_size },
00051         global_index { global_index },
00052         offset { offset }
00053     {}
00054
00055     /** To be able to copy and assign item, use default constructors too
00056
00057         \todo Make most of them protected, reserved to implementation
00058     */
00059     item() = default;
00060
00061     /** Return the constituent local or global id<> representing the
00062         work-item's position in the iteration space
00063     */
00064     id<dims> get() const { return global_index; }
00065
00066     /** Return the requested dimension of the constituent id<> representing
00067         the work-item's position in the iteration space
00068     */
00069     size_t get(int dimension) const { return get()[dimension]; }
00070
00071     /** Return the constituent id<> l-value representing the work-item's
00072         position in the iteration space in the given dimension
00073     */
00074     auto &operator[](int dimension) { return global_index[dimension]; }
00075
00076     /** Returns a range<> representing the dimensions of the range of

```

```

00085     possible values of the item
00086     */
00087     range<dims> get_range() const { return global_range; }
00088
00089
00090     /** Returns an id<> representing the n-dimensional offset provided to
00091         the parallel_for and that is added by the runtime to the global-ID
00092         of each work-item, if this item represents a global range
00093
00094         For an item representing a local range of where no offset was passed
00095         this will always return an id of all 0 values.
00096     */
00097     id<dims> get_offset() const { return offset; }
00098
00099
00100     /** Return the linearized ID in the item's range
00101
00102         Computed as the flattened ID after the offset is subtracted.
00103     */
00104     size_t get_linear_id() const {
00105         return detail::linear_id(get_range(), get(),
00106             get_offset());
00107     }
00108
00109     /** For the implementation, need to set the global index
00110
00111         \todo Move to private and add friends
00112     */
00113     void set(id<dims> Index) { global_index = Index; }
00114
00115
00116     /// Display the value for debugging and validation purpose
00117     void display() const {
00118         global_range.display();
00119         global_index.display();
00120         offset.display();
00121     }
00122 };
00123
00124
00125 /// @} End the parallelism Doxygen group
00126
00127 }
00128 }
00129
00130 /*
00131     # Some Emacs stuff:
00132     ### Local Variables:
00133     ### ispell-local-dictionary: "american"
00134     ### eval: (flyspell-prog-mode)
00135     ### End:
00136 */
00137
00138 #endif // TRISYCL_SYCL_ITEM_HPP

```

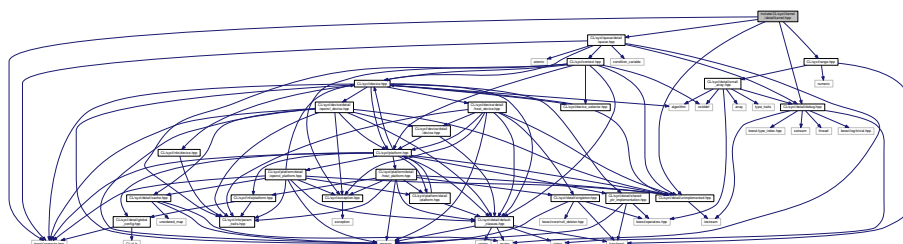
11.89 include/CL/sycl/kernel/detail/kernel.hpp File Reference

```

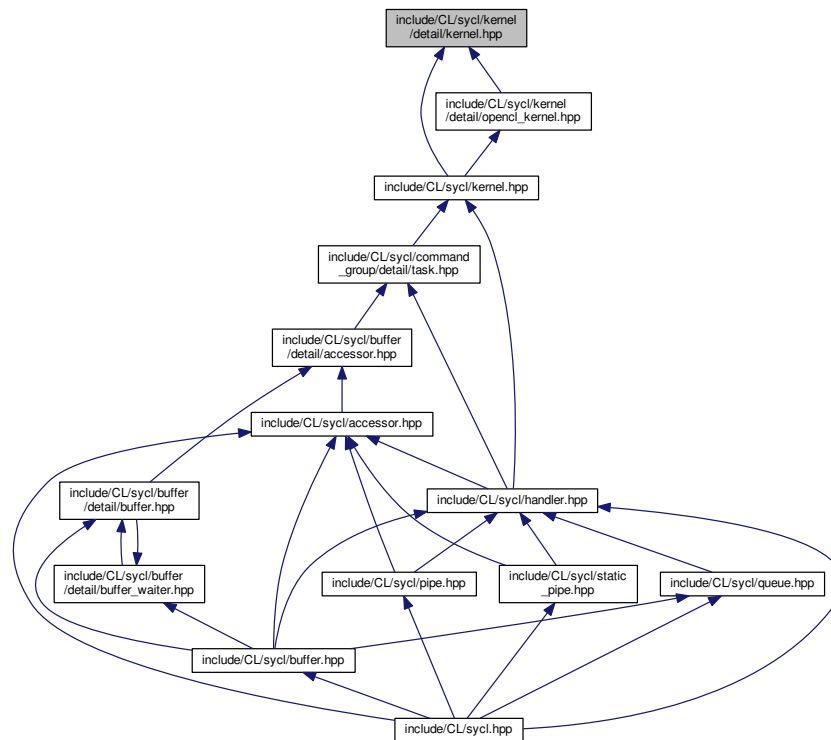
#include <boost/compute.hpp>
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/queue/detail/queue.hpp"
#include "CL/sycl/range.hpp"

```

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

```
00001 #ifndef TRISYCL_SYCL_KERNEL_DETAIL_KERNEL_HPP
00002 #define TRISYCL_SYCL_KERNEL_DETAIL_KERNEL_HPP
00003
00004 /** \file The OpenCL SYCL kernel
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 #ifdef TRISYCL_OPENCL
00013 #include <boost/compute.hpp>
00014 #endif
00015
00016 #include "CL/sycl/detail/debug.hpp"
00017 #include "CL/sycl/detail/unimplemented.hpp"
00018 // #include "CL/sycl/info/kernel.hpp"
00019 #include "CL/sycl/queue/detail/queue.hpp"
00020 #include "CL/sycl/range.hpp"
00021
00022 namespace cl {
00023 namespace sycl {
00024 namespace detail {
00025
00026 /** \addtogroup execution Platforms, contexts, devices and queues
00027     @{
00028 */
00029
00030 /// Abstract SYCL kernel
00031 class kernel : detail::debug<detail::kernel> {
00032
00033 public:
00034
00035 #ifdef TRISYCL_OPENCL
00036     /** Return the OpenCL kernel object for this kernel
00037
00038         Retains a reference to the returned cl_kernel object. Caller
00039         should release it when finished.
00040     */
00041     virtual cl_kernel get() const = 0;
00042
00043
00044     /** Return the Boost.Compute OpenCL kernel object for this kernel
00045
00046         This is an extension.
00047     */
00048     virtual boost::compute::kernel get_boost_compute() const = 0;
00049 #endif
```

```

00050
00051
00052  /** Launch a kernel with a range<>
00053
00054      Do not use a template since it does not work with virtual functions
00055
00056      \todo Think to a cleaner solution
00057  */
00058  #define TRISYCL_ParallelForKernel_RANGE(N) \
00059      virtual void parallel_for(std::shared_ptr<detail::task> task, std::shared_ptr<detail::queue> q, \
00060                              const range<N> &num_work_items) = 0;
00061
00062  TRISYCL_ParallelForKernel_RANGE(1)
00063  TRISYCL_ParallelForKernel_RANGE(2)
00064  TRISYCL_ParallelForKernel_RANGE(3)
00065  #undef TRISYCL_ParallelForKernel_RANGE
00066
00067
00068  /// Return the context that this kernel is defined for
00069  //virtual context get_context() const;
00070
00071  /// Return the program that this kernel is part of
00072  //virtual program get_program() const;
00073
00074  // Virtual to call the real destructor
00075  virtual ~kernel() {}
00076
00077 };
00078
00079 /// @} End the execution Doxygen group
00080
00081 }
00082 }
00083 }
00084
00085 /*
00086  # Some Emacs stuff:
00087  ### Local Variables:
00088  ###  ispell-local-dictionary: "american"
00089  ###  eval: (flyspell-prog-mode)
00090  ###  End:
00091  */
00092
00093 #endif // TRISYCL_SYCL_DETAIL_KERNEL_KERNEL_HPP

```

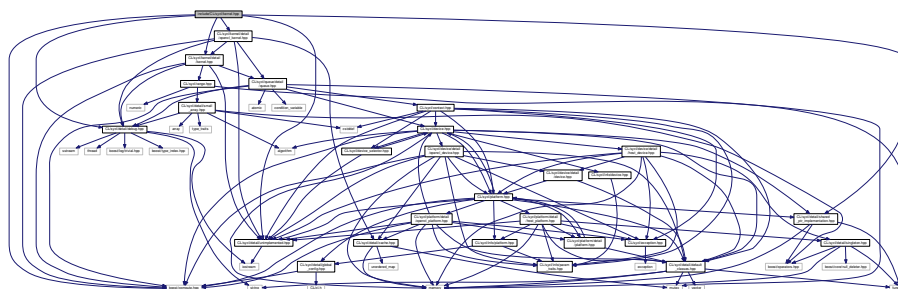
11.91 include/CL/sycl/kernel.hpp File Reference

```

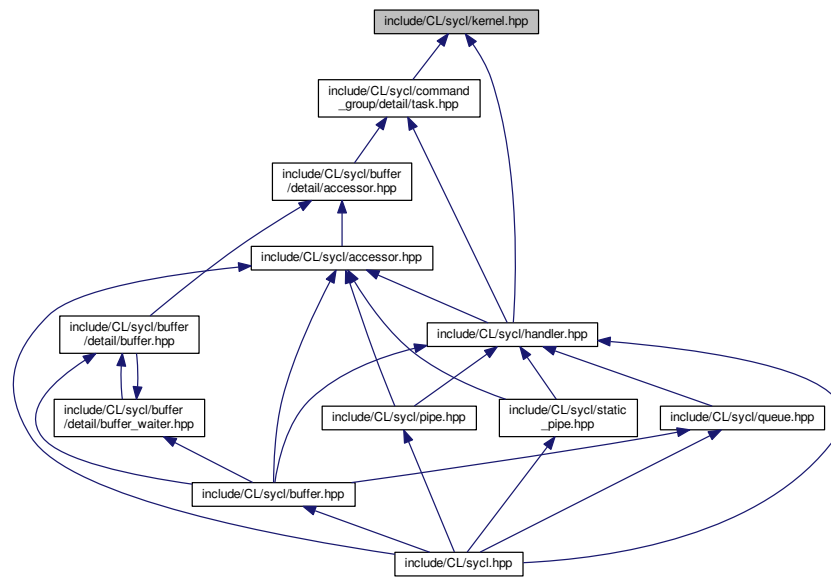
#include <boost/compute.hpp>
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/kernel/detail/kernel.hpp"
#include "CL/sycl/kernel/detail/opencl_kernel.hpp"

```

Include dependency graph for kernel.hpp:



This graph shows which files directly or indirectly include this file:



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

```

00001 #ifndef TRISYCL_SYCL_KERNEL_HPP
00002 #define TRISYCL_SYCL_KERNEL_HPP
00003
00004 /** \file The OpenCL SYCL kernel
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 #ifdef TRISYCL_OPENCL
00013 #include <boost/compute.hpp>
00014 #endif
00015
00016 #include "CL/sycl/detail/debug.hpp"
00017 #include "CL/sycl/detail/shared_ptr_implementation.hpp"
00018 #include "CL/sycl/detail/unimplemented.hpp"
00019 // #include "CL/sycl/info/kernel.hpp"

```

```

00020 #include "CL/sycl/kernel/detail/kernel.hpp"
00021 #ifdef TRISYCL_OPENCL
00022 #include "CL/sycl/kernel/detail/opencv_kernel.hpp"
00023 #endif
00024
00025 namespace cl {
00026 namespace sycl {
00027
00028 /** \addtogroup execution Platforms, contexts, devices and queues
00029     @{
00030 */
00031
00032 /** SYCL kernel
00033
00034     \todo To be implemented
00035
00036     \todo Check specification
00037 */
00038 class kernel
00039     /* Use the underlying kernel implementation that can be shared in
00040        the SYCL model */
00041     : public detail::shared_ptr_implementation<kernel, detail::kernel> {
00042
00043     // The type encapsulating the implementation
00044     using implementation_t =
00045         detail::shared_ptr_implementation<kernel, detail::kernel>
00046     ;
00047
00048     // Make the implementation member directly accessible in this class
00049     using implementation_t::implementation;
00050
00051     // The handler class uses the implementation
00052     friend class handler;
00053
00054     public:
00055
00056     /** The default object is not valid because there is no program or
00057         \code cl_kernel \endcode associated with it */
00058     kernel() = delete;
00059
00060 #ifdef TRISYCL_OPENCL
00061     /** Constructor for SYCL kernel class given an OpenCL kernel object
00062         with set arguments, valid for enqueueing
00063
00064         Retains a reference to the \p cl_kernel object. The Caller
00065         should release the passed cl_kernel object when it is no longer
00066         needed.
00067     */
00068     kernel(cl_kernel k) : kernel { boost::compute::kernel { k } } {}
00069
00070     /** Construct a kernel class instance using a boost::compute::kernel
00071
00072         This is a triSYCL extension for boost::compute interoperation.
00073
00074         Return synchronous errors via the SYCL exception class.
00075     */
00076     kernel(const boost::compute::kernel &k)
00077         : implementation_t { detail::opencv_kernel::instance(k) } {}
00078
00079
00080     /** Return the OpenCL kernel object for this kernel
00081
00082         Retains a reference to the returned cl_kernel object. Caller
00083         should release it when finished.
00084     */
00085     cl_kernel get() const {
00086         return implementation->get();
00087     }
00088 #endif
00089
00090
00091 #if 0
00092     /// Return the context that this kernel is defined for
00093     ///context get_context() const;
00094
00095     /// Return the program that this kernel is part of
00096     ///program get_program() const;
00097
00098     /** Query information from the kernel object using the
00099         info::kernel_info descriptor.
00100     */
00101     template <info::kernel param>
00102     typename info::param_traits<info::kernel, param>::type
00103     get_info() const {
00104         detail::unimplemented();
00105     }
00106 #endif

```



```

00105     }
00106 #endif
00107
00108 };
00109
00110 /// @} End the execution Doxygen group
00111
00112 }
00113 }
00114
00115
00116 /* Inject a custom specialization of std::hash to have the buffer
00117    usable into an unordered associative container
00118
00119    \todo Add this to the spec
00120 */
00121 namespace std {
00122
00123 template <> struct hash<cl::sycl::kernel> {
00124
00125     auto operator()(const cl::sycl::kernel &k) const {
00126         // Forward the hashing to the implementation
00127         return k.hash();
00128     }
00129 };
00130 };
00131
00132 }
00133
00134 /*
00135     # Some Emacs stuff:
00136     ### Local Variables:
00137     ###   ispell-local-dictionary: "american"
00138     ###   eval: (flyspell-prog-mode)
00139     ### End:
00140 */
00141
00142 #endif // TRISYCL_SYCL_KERNEL_HPP

```

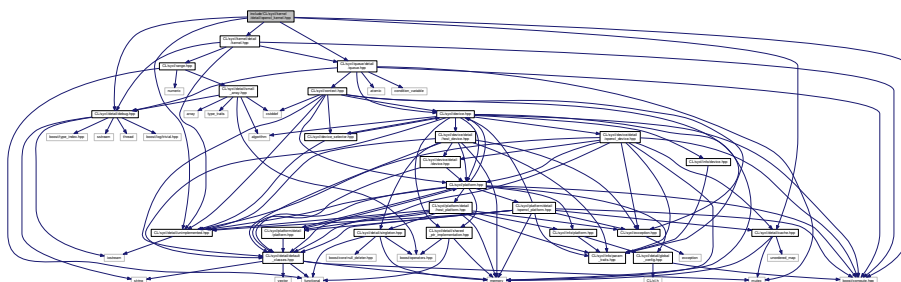
11.93 include/CL/sycl/kernel/detail/opencil_kernel.hpp File Reference

```

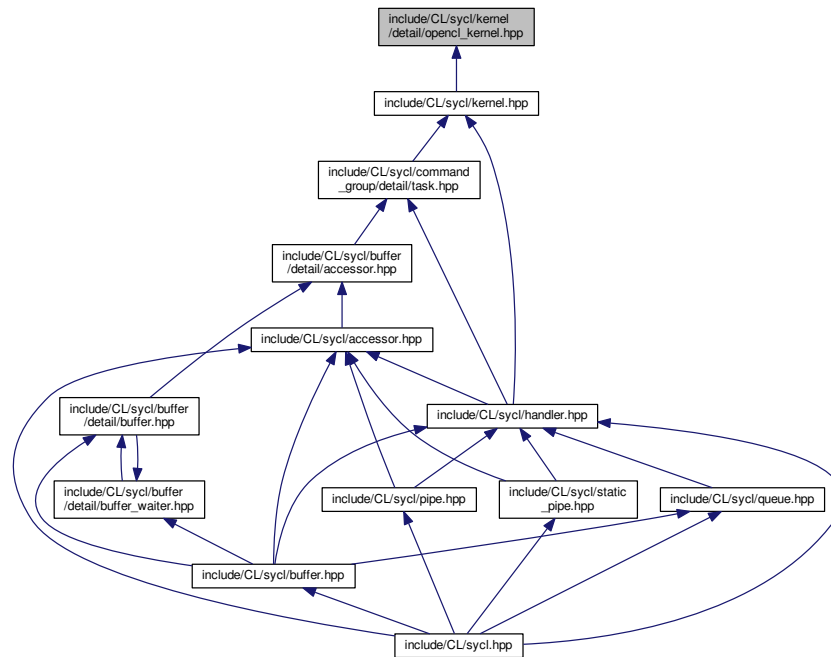
#include <boost/compute.hpp>
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/kernel/detail/kernel.hpp"
#include "CL/sycl/queue/detail/queue.hpp"

```

Include dependency graph for opencil_kernel.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class `cl::sycl::detail::opencv_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::opencv_kernel > opencv_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 [opencil_kernel.hpp](#).

11.94 opencil_kernel.hpp

```
00001 #ifndef TRISYCL_SYCL_KERNEL_DETAIL_OPENCIL_KERNEL_HPP
00002 #define TRISYCL_SYCL_KERNEL_DETAIL_OPENCIL_KERNEL_HPP
00003
00004 /** \file The OpenCL SYCL kernel
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 #ifdef TRISYCL_OPENCIL
00013 #include <boost/compute.hpp>
00014 #endif
00015
00016 #include "CL/sycl/detail/cache.hpp"
00017 #include "CL/sycl/detail/debug.hpp"
00018 #include "CL/sycl/detail/unimplemented.hpp"
00019 // #include "CL/sycl/info/kernel.hpp"
00020 #include "CL/sycl/kernel/detail/kernel.hpp"
00021 #include "CL/sycl/queue/detail/queue.hpp"
00022
00023 namespace cl {
00024 namespace sycl {
00025 namespace detail {
00026
00027     /// An abstraction of the OpenCL kernel
00028     class opencil_kernel : public detail::kernel,
00029         detail::debug<opencil_kernel> {
00030
00031     /// Use the Boost Compute abstraction of the OpenCL kernel
00032     boost::compute::kernel k;
00033
00034     /** A cache to always return the same alive kernel for a given
00035         OpenCL kernel
00036
```

```

00037         C++11 guaranties the static construction is thread-safe
00038     */
00039     static detail::cache<cl_kernel, detail::openccl_kernel>
cache;
00040
00041     openccl_kernel(const boost::compute::kernel &k) : k { k } {}
00042
00043     public:
00044
00045         ///// Get a singleton instance of the openccl_device
00046         static std::shared_ptr<openccl_kernel>
00047         instance(const boost::compute::kernel &k) {
00048             return cache.get_or_register(k.get(),
00049                                         [&] { return new openccl_kernel { k }; });
00050         }
00051
00052         /** Return the underlying OpenCL object
00053
00054             \todo Improve the spec to deprecate C OpenCL host API and move
00055             to C++ instead to avoid this ugly ownership management
00056         */
00057         cl_kernel get() const override {
00058             /// \todo Test error and throw. Externalize this feature in Boost.Compute?
00059             clRetainKernel(k);
00060             return k.get();
00061         }
00062
00063
00064         /** Return the Boost.Compute OpenCL kernel object for this kernel
00065
00066             This is an extension.
00067         */
00068         boost::compute::kernel get_boost_compute() const override {
00069             return k;
00070         }
00071
00072
00073         //context get_context() const override
00074
00075         //program get_program() const override
00076
00077     #if 0
00078         template <info::kernel param>
00079         typename info::param_traits<info::kernel, param>::type
00080         get_info() const {
00081             detail::unimplemented();
00082         }
00083     #endif
00084
00085
00086         /** Launch an OpenCL kernel with a range<>
00087
00088             Do not use a template since it does not work with virtual functions
00089
00090             \todo Think to a cleaner solution
00091         */
00092     #define TRISYCL_ParallelForKernel_RANGE(N)
00093         void parallel_for(std::shared_ptr<detail::task> task,\
00094                         std::shared_ptr<detail::queue> q,\
00095                         const range<N> &num_work_items) override {
00096             static_assert(sizeof(range<N>::value_type) == sizeof(size_t),
00097                           "num_work_items::value_type compatible with "
00098                           "Boost.Compute");
00099             q->get_boost_compute().enqueue_nd_range_kernel
00100             (k,
00101              static_cast<size_t>(N),
00102              NULL,
00103              static_cast<const size_t *>(num_work_items.data()),
00104              NULL);
00105             /* For now use a crude synchronization mechanism to map directly a
00106             host task to an accelerator task */
00107             q->get_boost_compute().finish();
00108         };
00109
00110         TRISYCL_ParallelForKernel_RANGE(1)
00111         TRISYCL_ParallelForKernel_RANGE(2)
00112         TRISYCL_ParallelForKernel_RANGE(3)
00113     #undef TRISYCL_ParallelForKernel_RANGE
00114
00115
00116         /// Unregister from the cache on destruction
00117         ~openccl_kernel() override {
00118             cache.remove(k.get());
00119         }
00120
00121     };
00122

```

```

00123 /* Allocate the cache here but since this is a pure-header library,
00124     use a weak symbol so that only one remains when SYCL headers are
00125     used in different compilation units of a program
00126 */
00127 detail::cache<cl_kernel, detail::opencl_kernel>
00128     opencil_kernel::cache
00129     __attribute__((weak));
00130 }
00131 }
00132 }
00133
00134 /*
00135     # Some Emacs stuff:
00136     ### Local Variables:
00137     ### ispell-local-dictionary: "american"
00138     ### eval: (flyspell-prog-mode)
00139     ### End:
00140 */
00141
00142 #endif // TRISYCL_SYCL_KERNEL_DETAIL_OPENCL_KERNEL_HPP

```

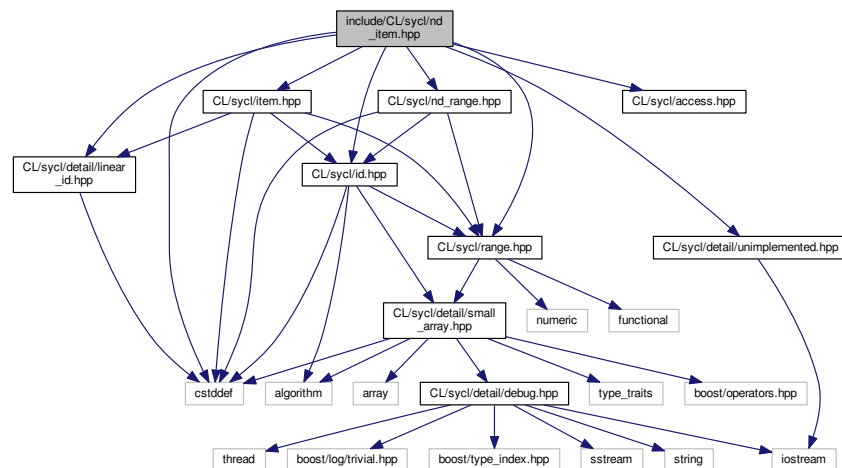
11.95 include/CL/sycl/nd_item.hpp File Reference

```

#include <cstdint>
#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:




```

00028
00029 /** A SYCL nd_item stores information on a work-item within a work-group,
00030     with some more context such as the definition ranges.
00031 */
00032 template <std::size_t dims = 1>
00033 struct nd_item {
00034     /// \todo add this Boost::multi_array or STL concept to the
00035     /// specification?
00036     static constexpr auto dimensionality = dims;
00037 private:
00038
00039     id<dims> global_index;
00040     /* This is a cached value since it can be computed from global_index and
00041        ND_range */
00042     id<dims> local_index;
00043     nd_range<dims> ND_range;
00044 public:
00045
00046     /** Create an empty nd_item<> from an nd_range<>
00047
00048         \todo This is for the triSYCL implementation which is expected to
00049         call set_global() and set_local() later. This should be hidden to
00050         the user.
00051     */
00052     nd_item(nd_range<dims> ndr) : ND_range { ndr } {}
00053
00054     /** Create a full nd_item
00055
00056         \todo This is for validation purpose. Hide this to the programmer
00057         somehow
00058     */
00059     nd_item(id<dims> global_index,
00060             nd_range<dims> ndr) :
00061         global_index { global_index },
00062         // Compute the local index using the offset and the group size
00063         local_index { (global_index - ndr.get_offset())%id<dims> { ndr.get_local() } },
00064         ND_range { ndr }
00065     {}
00066
00067     /** To be able to copy and assign nd_item, use default constructors too
00068
00069         \todo Make most of them protected, reserved to implementation
00070     */
00071     nd_item() = default;
00072
00073     /** Return the constituent global id representing the work-item's
00074         position in the global iteration space
00075     */
00076     id<dims> get_global() const { return global_index; }
00077
00078     /** Return the constituent element of the global id representing the
00079         work-item's position in the global iteration space in the given
00080         dimension
00081     */
00082     size_t get_global(int dimension) const { return get_global()[dimension]; }
00083
00084     /** Return the flattened id of the current work-item after subtracting
00085         the offset
00086     */
00087     size_t get_global_linear_id() const {
00088         return detail::linear_id(get_global_range(),
00089                                 get_global(), get_offset());
00090     }
00091
00092     /** Return the constituent local id representing the work-item's
00093         position within the current work-group
00094     */
00095     id<dims> get_local() const { return local_index; }
00096
00097     /** Return the constituent element of the local id representing the
00098         work-item's position within the current work-group in the given
00099         dimension
00100     */
00101     size_t get_local(int dimension) const { return get_local()[dimension]; }
00102
00103     /** Return the flattened id of the current work-item within the current
00104         work-group
00105     */

```

```

00114     */
00115     size_t get_local_linear_id() const {
00116         return detail::linear_id(get_local_range(),
get_local());
00117     }
00118
00119
00120     /** Return the constituent group group representing the work-group's
00121         position within the overall nd_range
00122     */
00123     id<dims> get_group() const {
00124         /* Convert get_local_range() to an id<> to remove ambiguity into using
00125             implicit conversion either from range<> to id<> or the opposite */
00126         return get_global()/id<dims> { get_local_range() };
00127     }
00128
00129
00130     /** Return the constituent element of the group id representing the
00131         work-group;s position within the overall nd_range in the given
00132         dimension.
00133     */
00134     size_t get_group(int dimension) const {
00135         return get_group()[dimension];
00136     }
00137
00138
00139     /// Return the flattened id of the current work-group
00140     size_t get_group_linear_id() const {
00141         return detail::linear_id(get_num_groups(),
get_group());
00142     }
00143
00144
00145     /// Return the number of groups in the nd_range
00146     id<dims> get_num_groups() const {
00147         return get_nd_range().get_group();
00148     }
00149
00150     /// Return the number of groups for dimension in the nd_range
00151     size_t get_num_groups(int dimension) const {
00152         return get_num_groups()[dimension];
00153     }
00154
00155
00156     /// Return a range<> representing the dimensions of the nd_range<>
00157     range<dims> get_global_range() const {
00158         return get_nd_range().get_global();
00159     }
00160
00161
00162     /// Return a range<> representing the dimensions of the current work-group
00163     range<dims> get_local_range() const {
00164         return get_nd_range().get_local();
00165     }
00166
00167
00168     /** Return an id<> representing the n-dimensional offset provided to the
00169         constructor of the nd_range<> and that is added by the runtime to the
00170         global-ID of each work-item
00171     */
00172     id<dims> get_offset() const { return get_nd_range().get_offset(); }
00173
00174
00175     /// Return the nd_range<> of the current execution
00176     nd_range<dims> get_nd_range() const { return
ND_range; }
00177
00178
00179     /** Allows projection down to an item
00180
00181         \todo Add to the specification
00182     */
00183     item<dims> get_item() const {
00184         return { get_global_range(), get_global(),
get_offset() };
00185     }
00186
00187
00188     /** Execute a barrier with memory ordering on the local address space,
00189         global address space or both based on the value of flag
00190
00191         The current work-item will wait at the barrier until all work-items
00192         in the current work-group have reached the barrier.
00193
00194         In addition, the barrier performs a fence operation ensuring that all
00195         memory accesses in the specified address space issued before the
00196         barrier complete before those issued after the barrier

```



```

00197  */
00198  void barrier(access::fence_space flag =
00199             access::fence_space::global_and_local) const {
00200  #if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
00201      /* Use OpenMP barrier in the implementation with 1 OpenMP thread per
00202         work-item of the work-group */
00203      #pragma omp barrier
00204  #else
00205      // \todo To be implemented efficiently otherwise
00206      detail::unimplemented();
00207  #endif
00208  }
00209
00210
00211  // For the triSYCL implementation, need to set the local index
00212  void set_local(id<dims> Index) { local_index = Index; }
00213
00214
00215  // For the triSYCL implementation, need to set the global index
00216  void set_global(id<dims> Index) { global_index = Index; }
00217
00218  };
00219
00220  /// @} End the parallelism Doxygen group
00221
00222  }
00223
00224
00225  /*
00226      # Some Emacs stuff:
00227      ### Local Variables:
00228      ### ispell-local-dictionary: "american"
00229      ### eval: (flyspell-prog-mode)
00230      ### End:
00231  */
00232
00233  #endif // TRISYCL_SYCL_ND_ITEM_HPP

```

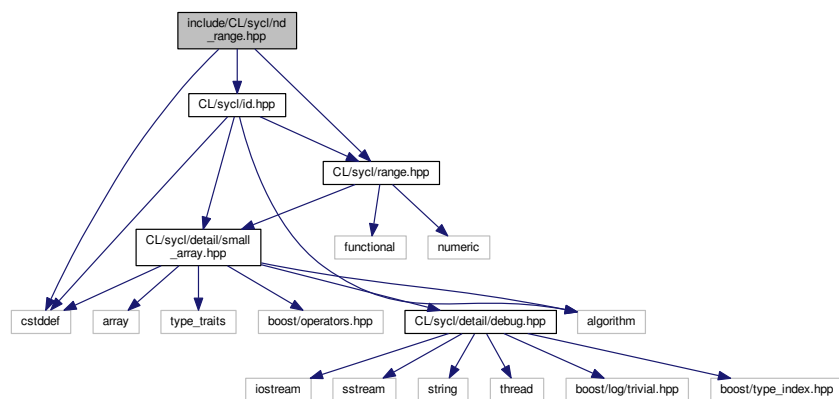
11.97 include/CL/sycl/nd_range.hpp File Reference

```

#include <cstdint>
#include "CL/sycl/id.hpp"
#include "CL/sycl/range.hpp"

```

Include dependency graph for nd_range.hpp:




```

00028     needed.
00029
00030     \todo add copy constructors in the specification
00031 */
00032 template <std::size_t dims = 1>
00033 struct nd_range {
00034     /// \todo add this Boost::multi_array or STL concept to the
00035     /// specification?
00036     static constexpr auto dimensionality = dims;
00037
00038 private:
00039
00040     range<dimensionality> global_range;
00041     range<dimensionality> local_range;
00042     id<dimensionality> offset;
00043
00044 public:
00045
00046     /** Construct a ND-range with all the details available in OpenCL
00047
00048         By default use a zero offset, that is iterations start at 0
00049     */
00050     nd_range(range<dims> global_size,
00051             range<dims> local_size,
00052             id<dims> offset = {}) :
00053         global_range { global_size }, local_range { local_size }, offset { offset }
00054     { }
00055
00056
00057     /// Get the global iteration space range
00058     range<dims> get_global() const { return global_range; }
00059
00060
00061     /// Get the local part of the iteration space range
00062     range<dims> get_local() const { return local_range; }
00063
00064
00065     /// Get the range of work-groups needed to run this ND-range
00066     auto get_group() const {
00067         /* This is basically global_range/local_range, round up to the
00068            next integer, in case the global eange is not a multiple of the
00069            local range. Note this is a motivating example to build a range
00070            from a scalar with a broadcasting constructor. */
00071         return (global_range + local_range - range<dims>{ 1 })/local_range;
00072     }
00073
00074
00075     /// \todo get_offset() is lacking in the specification
00076     id<dims> get_offset() const { return offset; }
00077
00078
00079     /// Display the value for debugging and validation purpose
00080     void display() const {
00081         global_range.display();
00082         local_range.display();
00083         offset.display();
00084     }
00085
00086 };
00087
00088 /// @} End the parallelism Doxygen group
00089
00090 }
00091 }
00092
00093 /*
00094     # Some Emacs stuff:
00095     ### Local Variables:
00096     ### ispell-local-dictionary: "american"
00097     ### eval: (flyspell-prog-mode)
00098     ### End:
00099 */
00100
00101 #endif // TRISYCL_SYCL_ND_RANGE_HPP

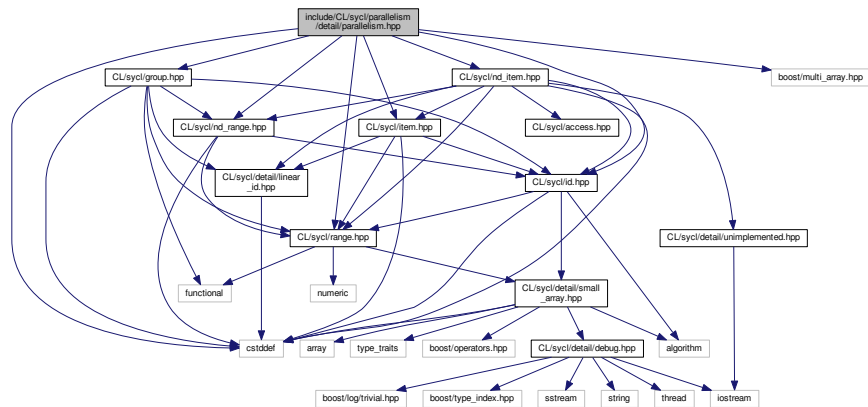
```

11.99 include/CL/sycl/parallelism/detail/parallelism.hpp File Reference

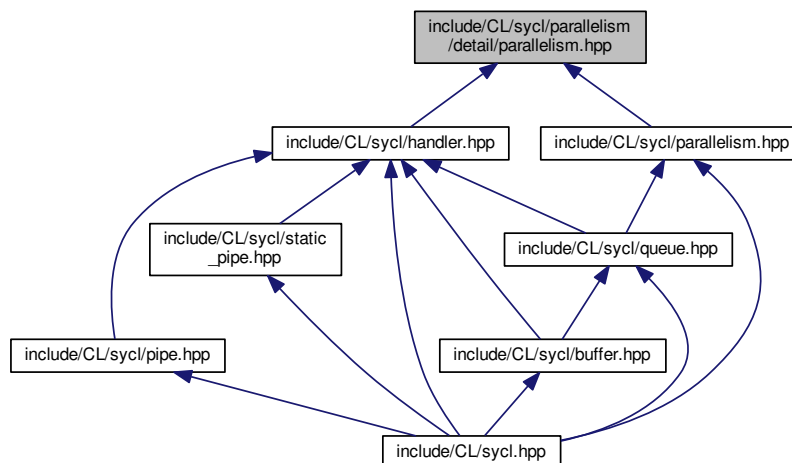
Implement the detail of the parallel constructions to launch kernels.

```
#include <cstdint>
#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:



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

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file [parallelism.hpp](#).

11.100 parallelism.hpp

```

00001 #ifndef TRISYCL_SYCL_PARALLELISM_DETAIL_PARALLELISM_HPP
00002 #define TRISYCL_SYCL_PARALLELISM_DETAIL_PARALLELISM_HPP
00003
00004 /** \file
00005
00006     Implement the detail of the parallel constructions to launch kernels
00007
00008     Ronan at keryell dot FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 #include <cstdlib>
00015 #include <boost/multi_array.hpp>
00016
00017 #include "CL/sycl/group.hpp"
00018 #include "CL/sycl/id.hpp"
00019 #include "CL/sycl/item.hpp"
00020 #include "CL/sycl/nd_item.hpp"
00021 #include "CL/sycl/nd_range.hpp"
00022 #include "CL/sycl/range.hpp"
00023
00024 #ifdef _OPENMP
00025 #include <omp.h>
00026 #endif
00027
00028
00029 /** \addtogroup parallelism
00030     @{
00031 */
00032
00033 namespace cl {
00034 namespace sycl {
00035 namespace detail {
00036
00037
00038 /** A recursive multi-dimensional iterator that ends up calling f
00039
00040     The iteration order may be changed later.
00041
00042     Since partial specialization of function template is not possible in
00043     C++14, use a class template instead with everything in the
00044     constructor.
00045 */
00046 template <std::size_t level, typename Range, typename ParallelForFunctor, typename Id>
00047 struct parallel_for_iterate {
00048     parallel_for_iterate(Range r, ParallelForFunctor &f, Id &index) {
00049         for (boost::multi_array_types::index _sycl_index = 0,
00050              _sycl_end = r[Range::dimensionality - level];
00051              _sycl_index < _sycl_end;
00052              _sycl_index++) {
00053             // Set the current value of the index for this dimension
00054             index[Range::dimensionality - level] = _sycl_index;
00055             // Iterate further on lower dimensions
00056             parallel_for_iterate<level - 1,
00057                                 Range,
00058                                 ParallelForFunctor,
00059                                 Id> { r, f, index };
00060         }
00061     };
00062 };
00063
00064
00065 /** A top-level recursive multi-dimensional iterator variant using OpenMP
00066
00067     Only the top-level loop uses OpenMP and goes on with the normal
00068     recursive multi-dimensional.
00069 */
00070 template <std::size_t level,
00071           typename Range,
00072           typename ParallelForFunctor,
00073           typename Id>
00074 struct parallel_OpenMP_for_iterate {
00075     parallel_OpenMP_for_iterate(Range r, ParallelForFunctor &f) {
00076         // Create the OpenMP threads before the for-loop to avoid creating an
00077         // index in each iteration
00078         #pragma omp parallel
00079         {
00080             // Allocate an OpenMP thread-local index
00081             Id index;
00082             // Make a simple loop end condition for OpenMP
00083             boost::multi_array_types::index _sycl_end =
00084                 r[Range::dimensionality - level];

```

```

00085     /* Distribute the iterations on the OpenMP threads. Some OpenMP
00086     "collapse" could be useful for small iteration space, but it
00087     would need some template specialization to have real contiguous
00088     loop nests */
00089 #pragma omp for
00090     for (boost::multi_array_types::index _sycl_index = 0;
00091          _sycl_index < _sycl_end;
00092          _sycl_index++) {
00093         // Set the current value of the index for this dimension
00094         index[Range::dimensionality - level] = _sycl_index;
00095         // Iterate further on lower dimensions
00096         parallel_for_iterate<level - 1,
00097                               Range,
00098                               ParallelForFunctor,
00099                               Id> { r, f, index };
00100     }
00101 }
00102 }
00103 };
00104
00105
00106 /** Stop the recursion when level reaches 0 by simply calling the
00107     kernel functor with the constructed id */
00108 template <typename Range, typename ParallelForFunctor, typename Id>
00109 struct parallel_for_iterate<0, Range, ParallelForFunctor, Id> {
00110     parallel_for_iterate(Range r, ParallelForFunctor &f, Id &index) {
00111         f(index);
00112     }
00113 };
00114
00115
00116 /** Implementation of a data parallel computation with parallelism
00117     specified at launch time by a range<>. Kernel index is id or int.
00118
00119     This implementation use OpenMP 3 if compiled with the right flag.
00120 */
00121 template <std::size_t Dimensions = 1, typename ParallelForFunctor, typename Id>
00122 void parallel_for(range<Dimensions> r,
00123                  ParallelForFunctor f,
00124                  Id) {
00125 #ifdef _OPENMP
00126     // Use OpenMP for the top loop level
00127     parallel_OpenMP_for_iterate<Dimensions,
00128                                range<Dimensions>,
00129                                ParallelForFunctor,
00130                                id<Dimensions>> { r, f };
00131 #else
00132     // In a sequential execution there is only one index processed at a time
00133     id<Dimensions> index;
00134     parallel_for_iterate<Dimensions,
00135                          range<Dimensions>,
00136                          ParallelForFunctor,
00137                          id<Dimensions>> { r, f, index };
00138 #endif
00139 }
00140
00141
00142 /** Implementation of a data parallel computation with parallelism
00143     specified at launch time by a range<>. Kernel index is item.
00144
00145     This implementation use OpenMP 3 if compiled with the right flag.
00146 */
00147 template <std::size_t Dimensions = 1, typename ParallelForFunctor>
00148 void parallel_for(range<Dimensions> r,
00149                  ParallelForFunctor f,
00150                  item<Dimensions>) {
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 }

```

```

00172
00173
00174 /** Calls the appropriate ternary parallel_for overload based on the
00175     index type of the kernel function object f
00176
00177 */
00178 template <std::size_t Dimensions = 1, typename ParallelForFuncor>
00179 void parallel_for(range<Dimensions> r, ParallelForFuncor f) {
00180     using mf_t = decltype(std::mem_fn(&ParallelForFuncor::operator()));
00181     using arg_t = typename mf_t::second_argument_type;
00182     parallel_for(r, f, arg_t{});
00183 }
00184
00185
00186 /** Implementation of parallel_for with a range<> and an offset */
00187 template <std::size_t Dimensions = 1, typename ParallelForFuncor>
00188 void parallel_for_global_offset(range<Dimensions> global_size,
00189                                id<Dimensions> offset,
00190                                ParallelForFuncor f) {
00191     // Reconstruct the item from its id<> and its offset
00192     auto reconstruct_item = [&] (id<Dimensions> l) {
00193         // Reconstruct the global item
00194         item<Dimensions> index { global_size, l + offset, offset };
00195         // Call the user kernel with the item<> instead of the id<>
00196         f(index);
00197     };
00198
00199     // First iterate on all the work-groups
00200     parallel_for(global_size, reconstruct_item);
00201 }
00202
00203
00204 /** Implement a variation of parallel_for to take into account a
00205     nd_range<>
00206
00207     \todo Add an OpenMP implementation
00208
00209     \todo Deal with incomplete work-groups
00210
00211     \todo Implement with parallel_for_workgroup()/parallel_for_workitem()
00212 */
00213 template <std::size_t Dimensions = 1, typename ParallelForFuncor>
00214 void parallel_for(nd_range<Dimensions> r,
00215                  ParallelForFuncor f) {
00216     // In a sequential execution there is only one index processed at a time
00217     nd_item<Dimensions> index { r };
00218     // To iterate on the work-group
00219     id<Dimensions> group;
00220     range<Dimensions> group_range = r.get_group();
00221     // To iterate on the local work-item
00222     id<Dimensions> local;
00223
00224     range<Dimensions> local_range = r.get_local();
00225
00226     // Reconstruct the nd_item from its group and local id
00227     auto reconstruct_item = [&] (id<Dimensions> l) {
00228         //local.display();
00229         // Reconstruct the global nd_item
00230         index.set_local(local);
00231         // Upgrade local_range to an id<> so that we can * with the group (an id<>)
00232         index.set_global(local + id<Dimensions>(local_range)*group);
00233         // Call the user kernel at last
00234         f(index);
00235     };
00236
00237     // To recycle the parallel_for on range<>, wrap the ParallelForFuncor f
00238     // into another functor that iterates inside the work-group and then
00239     // calls f */
00240     auto iterate_in_work_group = [&] (id<Dimensions> g) {
00241         //group.display();
00242         // Then iterate on the local work-groups
00243         parallel_for_iterate<Dimensions,
00244                                range<Dimensions>,
00245                                decltype(reconstruct_item),
00246                                id<Dimensions>> { local_range,
00247                                                reconstruct_item,
00248                                                local };
00249     };
00250
00251     // First iterate on all the work-groups
00252     parallel_for_iterate<Dimensions,
00253                          range<Dimensions>,
00254                          decltype(iterate_in_work_group),
00255                          id<Dimensions>> { group_range,
00256                                          iterate_in_work_group,
00257                                          group };
00258 }

```

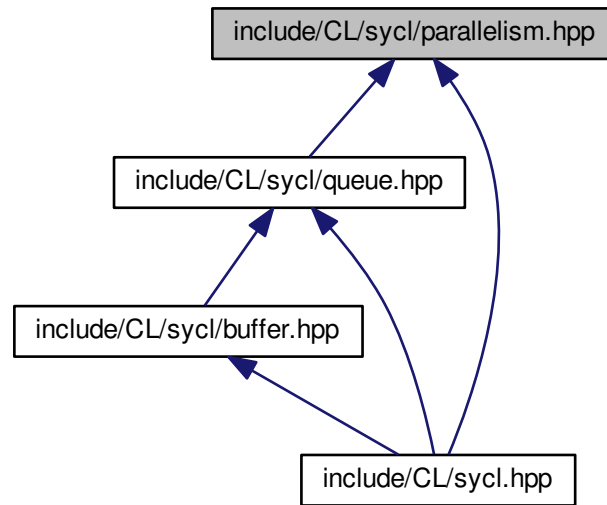


```

00259
00260
00261 /// Implement the loop on the work-groups
00262 template <std::size_t Dimensions = 1, typename ParallelForFuncor>
00263 void parallel_for_workgroup(nd_range<Dimensions> r,
00264                             ParallelForFuncor f) {
00265     // In a sequential execution there is only one index processed at a time
00266     group<Dimensions> g { r };
00267
00268     // First iterate on all the work-groups
00269     parallel_for_iterate<Dimensions,
00270                         range<Dimensions>,
00271                         ParallelForFuncor,
00272                         group<Dimensions>> {
00273         r.get_group(),
00274         f,
00275         g };
00276 }
00277
00278
00279 /** Implement the loop on the work-items inside a work-group
00280
00281     \todo Better type the functor
00282 */
00283 template <std::size_t Dimensions, typename ParallelForFuncor>
00284 void parallel_for_workitem(const group<Dimensions> &g,
00285                             ParallelForFuncor f) {
00286     #if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
00287     /* To implement barriers With OpenMP, one thread is created for each
00288     work-item in the group and thus an OpenMP barrier has the same effect
00289     of an OpenCL barrier executed by the work-items in a workgroup
00290
00291     The issue is that the parallel_for_workitem() execution is slow even
00292     when nd_item::barrier() is not used
00293     */
00294
00295     // Is the above comment true anymore ?
00296     // Maybe the following will be enough
00297     // #ifdef _OPENMP
00298
00299     // With OMP, one task is created for each work-item in the group
00300
00301     range<Dimensions> l_r = g.get_nd_range().get_local();
00302     int tot = l_r.get(0);
00303     for (int i = 1; i < (int) Dimensions; ++i){
00304         tot += l_r.get(i);
00305     }
00306     #pragma omp parallel
00307     {
00308         #pragma omp single nowait
00309         {
00310             for (int th_id = 0; th_id < tot; ++th_id) {
00311                 #pragma omp task firstprivate(th_id)
00312                 {
00313                     nd_item<Dimensions> index { g.get_nd_range() };
00314                     id<Dimensions> local; // to initialize correctly
00315
00316                     if (Dimensions == 1) {
00317                         local[0] = th_id;
00318                     } else if (Dimensions == 2) {
00319                         local[0] = th_id / l_r.get(1);
00320                         local[1] = th_id - local[0]*l_r.get(1);
00321                     } else if (Dimensions == 3) {
00322                         int tmp = l_r.get(1)*l_r.get(2);
00323                         local[0] = th_id / tmp;
00324                         local[1] = (th_id - local[0]*tmp) / l_r.get(1);
00325                         local[2] = th_id - local[0]*tmp - local[1]*l_r.get(1);
00326                     }
00327                     index.set_local(local);
00328                     index.set_global(local + id<Dimensions>(l_r)*g.get());
00329                     f(index);
00330                 }
00331             }
00332         }
00333     }
00334 }
00335 #else
00336 // In a sequential execution there is only one index processed at a time
00337 nd_item<Dimensions> index { g.get_nd_range() };
00338 // To iterate on the local work-item
00339 id<Dimensions> local;
00340
00341 // Reconstruct the nd_item from its group and local id
00342 auto reconstruct_item = [&] (id<Dimensions> l) {
00343     //local.display();
00344     //l.display();
00345     // Reconstruct the global nd_item

```


This graph shows which files directly or indirectly include this file:



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

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file [parallelism.hpp](#).

11.102 parallelism.hpp

```

00001 #ifndef TRISYCL_SYCL_PARALLELISM_HPP
00002 #define TRISYCL_SYCL_PARALLELISM_HPP
00003
00004 /** \file
00005
00006     Implement parallel constructions to launch kernels
00007
00008     Ronan at keryell dot FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 #include "CL/sycl/parallelism/detail/parallelism.hpp"
00015
00016 namespace cl {
00017 namespace sycl {
00018
00019 /** \addtogroup parallelism
00020     @{
00021 */
00022
00023 /// SYCL parallel_for version that allows a Program object to be specified
00024 /// \todo To be implemented
00025 /* template <typename Range, typename Program, typename ParallelForFunctor>
00026 void parallel_for(Range r, Program p, ParallelForFunctor f) {
00027     /// \todo deal with Program
00028     parallel_for(r, f);
00029 }
00030 */
00031
00032 /** Loop on the work-items inside a work-group
00033
00034     \todo Deprecate this function in the specification to use
00035     instead the group method
00036 */
00037 template <std::size_t Dimensions = 1, typename ParallelForFunctor>
00038 void parallel_for_work_item(const group<Dimensions> &g,
00039                             ParallelForFunctor f) {
00040     detail::parallel_for_workitem(g, f);
00041 }
00042
00043
00044 }
00045 }
00046 }
00047
00048 /// @} End the parallelism Doxygen group
00049
00050 /*
00051     # Some Emacs stuff:
00052     ### Local Variables:
00053     ### ispell-local-dictionary: "american"
00054     ### eval: (flyspell-prog-mode)
00055     ### End:
00056 */
00057
00058 #endif // TRISYCL_SYCL_PARALLELISM_HPP

```

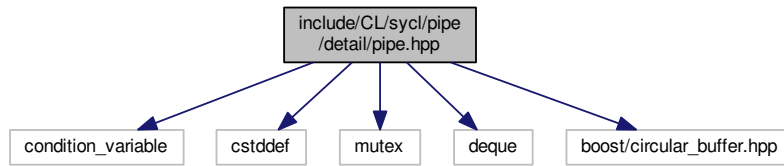
11.103 include/CL/sycl/pipe/detail/pipe.hpp File Reference

```

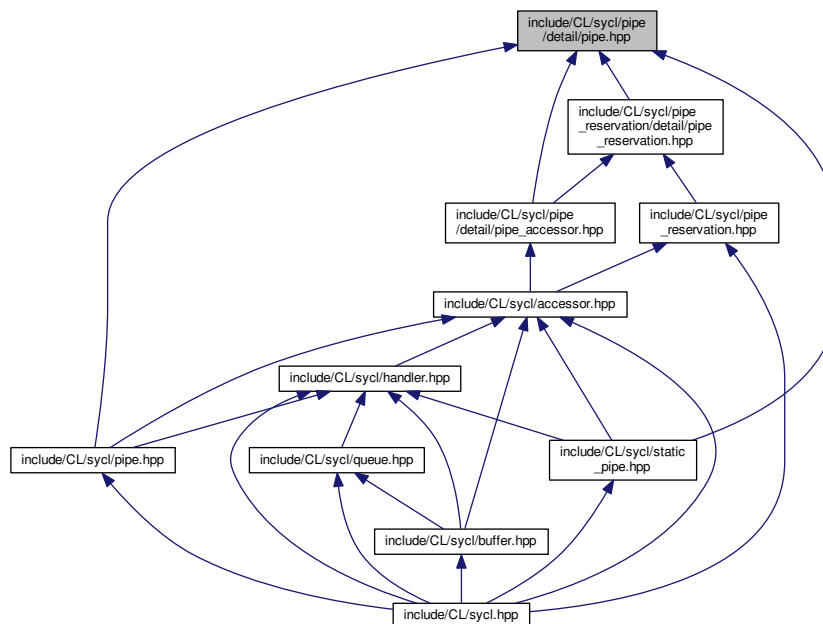
#include <condition_variable>
#include <cstdint>
#include <mutex>
#include <deque>
#include <boost/circular_buffer.hpp>

```

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

```

00001 #ifndef TRISYCL_SYCL_PIPE_DETAIL_PIPE_HPP
00002 #define TRISYCL_SYCL_PIPE_DETAIL_PIPE_HPP
00003
00004 /** \file The OpenCL SYCL pipe<> details
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 #include <condition_variable>
00013 #include <cstdint>
00014 #include <mutex>
00015 #include <deque>
00016
00017 #ifdef MAKE_BOOST_CIRCULARBUFFER_THREAD_SAFE
00018 /* The debug mode of boost/circular_buffer.hpp has a nasty side effect
00019    in multithread applications using several iterators at the same
00020    time even in read-only mode because the library tracks them for
00021    debugging purpose in a... non-thread safe way
00022
00023    This is described in https://svn.boost.org/trac/boost/ticket/6277
00024    and fixed with https://github.com/boostorg/circular_buffer/pull/9
00025 */
00026 #define BOOST_CB_DISABLE_DEBUG
00027 #endif
00028 #include <boost/circular_buffer.hpp>
00029
00030 namespace cl {
00031 namespace sycl {
00032 namespace detail {
00033
00034 /** \addtogroup data Data access and storage in SYCL
00035     @{
00036 */
00037
00038 /// A private description of a reservation station
00039 template <typename T>
00040 struct reserve_id {
00041     /// Start of the reservation in the pipe storage
00042     typename boost::circular_buffer<T>::iterator start;
00043
00044     /// Number of elements in the reservation
00045     std::size_t size;
00046
00047     /* True when the reservation has been committed and is ready to be
00048        released */
00049     bool ready = false;
00050
00051     /** Track a reservation not committed yet
00052
00053         \param[in] start point to the start of the reservation in the
00054         pipe storage
00055
00056         \param[in] size is the number of elements in the reservation
00057     */
00058     reserve_id(typename boost::circular_buffer<T>::iterator start,
00059               std::size_t size) : start { start }, size { size } {}
00060 };
00061
00062 /** Implement a pipe object
00063
00064     Use some mutable members so that the pipe object can be changed even
00065     when the accessors are captured in a lambda.
00066 */
00067 template <typename T>
00070 class pipe : public detail::debug<pipe<T>> {
00071 public:
00072     using value_type = T;
00073
00074     /// Implement the pipe with a circular buffer
00075     using implementation_t = boost::circular_buffer<value_type>;
00076
00077 private:
00078     /// The circular buffer to store the elements
00079     boost::circular_buffer<value_type> cb;
00080
00081     /** To protect the access to the circular buffer.

```

```

00085
00086     In case the object is capture in a lambda per copy, make it
00087     mutable. */
00088     mutable std::mutex cb_mutex;
00089
00090     /// The queue of pending write reservations
00091     std::deque<reserve_id<value_type>> w_rid_q;
00092
00093 public:
00094
00095     using rid_iterator = typename decltype(w_rid_q)::iterator;
00096
00097 private:
00098
00099     /// The queue of pending read reservations
00100     std::deque<reserve_id<value_type>> r_rid_q;
00101
00102     /// Track the number of frozen elements related to read reservations
00103     std::size_t read_reserved_frozen;
00104
00105     /// To signal that a read has been successful
00106     std::condition_variable read_done;
00107
00108     /// To signal that a write has been successful
00109     std::condition_variable write_done;
00110
00111     /// To control the debug mode, disabled by default
00112     bool debug_mode = false;
00113
00114 public:
00115
00116     /// True when the pipe is currently used for reading
00117     bool used_for_reading = false;
00118
00119     /// True when the pipe is currently used for writing
00120     bool used_for_writing = false;
00121
00122     /// Create a pipe as a circular buffer of the required capacity
00123     pipe(std::size_t capacity) : cb { capacity }, read_reserved_frozen { 0 } { }
00124
00125
00126     /** Return the maximum number of elements that can fit in the pipe
00127     */
00128     std::size_t capacity() const {
00129         // No lock required since it is fixed and set at construction time
00130         return cb.capacity();
00131     }
00132
00133 private:
00134
00135     /** Get the current number of elements in the pipe that can be read
00136
00137     This is obviously a volatile value which is constrained by the
00138     theory of restricted relativity.
00139
00140     Note that on some devices it may be costly to implement (for
00141     example on FPGA).
00142     */
00143     std::size_t size() const {
00144         TRISYCL_DUMP_T("size() cb.size() = " << cb.size()
00145             << " cb.end() = " << (void *)&cb.end()
00146             << " reserved_for_reading() = " << reserved_for_reading()
00147             << " reserved_for_writing() = " << reserved_for_writing());
00148         /* The actual number of available elements depends from the
00149         elements blocked by some reservations.
00150         This prevents a consumer to read into reserved area. */
00151         return cb.size() - reserved_for_reading() - reserved_for_writing();
00152     }
00153
00154
00155     /** Test if the pipe is empty
00156
00157     This is obviously a volatile value which is constrained by
00158     restricted relativity.
00159
00160     Note that on some devices it may be costly to implement on the
00161     write side (for example on FPGA).
00162     */
00163     bool empty() const {
00164         TRISYCL_DUMP_T("empty() cb.size() = " << cb.size()
00165             << " size() = " << size());
00166         // It is empty when the size is zero, taking into account reservations
00167         return size() == 0;
00168     }
00169
00170
00171     /** Test if the pipe is full

```

```

00172
00173     This is obviously a volatile value which is constrained by
00174     restricted relativity.
00175
00176     Note that on some devices it may be costly to implement on the
00177     read side (for example on FPGA).
00178 */
00179 bool full() const {
00180     return cb.full();
00181 }
00182
00183
00184 public:
00185
00186     /// The size() method used outside needs to lock the datastructure
00187     std::size_t size_with_lock() const {
00188         std::lock_guard<std::mutex> lg { cb_mutex };
00189         return size();
00190     }
00191
00192
00193     /// The empty() method used outside needs to lock the datastructure
00194     bool empty_with_lock() const {
00195         std::lock_guard<std::mutex> lg { cb_mutex };
00196         return empty();
00197     }
00198
00199
00200     // The full() method used outside needs to lock the datastructure
00201     bool full_with_lock() const {
00202         std::lock_guard<std::mutex> lg { cb_mutex };
00203         return full();
00204     }
00205
00206
00207     /** Try to write a value to the pipe
00208
00209         \param[in] value is what we want to write
00210
00211         \param[in] blocking specify if the call wait for the operation
00212         to succeed
00213
00214         \return true on success
00215
00216         \todo provide a && version
00217     */
00218     bool write(const T &value, bool blocking = false) {
00219         // Lock the pipe to avoid being disturbed
00220         std::unique_lock<std::mutex> ul { cb_mutex };
00221         TRISYCL_DUMP_T("Write pipe full = " << full()
00222             << " value = " << value);
00223
00224         if (blocking)
00225             /* If in blocking mode, wait for the not full condition, that
00226                may be changed when a read is done */
00227             read_done.wait(ul, [&] { return !full(); });
00228         else if (full())
00229             return false;
00230
00231         cb.push_back(value);
00232         TRISYCL_DUMP_T("Write pipe front = " << cb.front()
00233             << " back = " << cb.back()
00234             << " cb.begin() = " << (void *)&cb.begin()
00235             << " cb.size() = " << cb.size()
00236             << " cb.end() = " << (void *)&cb.end()
00237             << " reserved_for_reading() = " << reserved_for_reading()
00238             << " reserved_for_writing() = " << reserved_for_writing());
00239         // Notify the clients waiting to read something from the pipe
00240         write_done.notify_all();
00241         return true;
00242     }
00243
00244
00245     /** Try to read a value from the pipe
00246
00247         \param[out] value is the reference to where to store what is
00248         read
00249
00250         \param[in] blocking specify if the call wait for the operation
00251         to succeed
00252
00253         \return true on success
00254     */
00255     bool read(T &value, bool blocking = false) {
00256         // Lock the pipe to avoid being disturbed
00257         std::unique_lock<std::mutex> ul { cb_mutex };
00258         TRISYCL_DUMP_T("Read pipe empty = " << empty());

```



```

00259
00260     if (blocking)
00261         /* If in blocking mode, wait for the not empty condition, that
00262            may be changed when a write is done */
00263         write_done.wait(ul, [&] { return !empty(); });
00264     else if (empty())
00265         return false;
00266
00267     TRISYCL_DUMP_T("Read pipe front = " << cb.front()
00268                   << " back = " << cb.back()
00269                   << " reserved_for_reading() = " << reserved_for_reading());
00270     if (read_reserved_frozen)
00271         /** If there is a pending reservation, read the next element to
00272            be read and update the number of reserved elements */
00273         value = cb.begin()[read_reserved_frozen++];
00274     else {
00275         /* There is no pending read reservation, so pop the read value
00276            from the pipe */
00277         value = cb.front();
00278         cb.pop_front();
00279     }
00280
00281     TRISYCL_DUMP_T("Read pipe value = " << value);
00282     // Notify the clients waiting for some room to write in the pipe
00283     read_done.notify_all();
00284     return true;
00285 }
00286
00287 /** Compute the amount of elements blocked by read reservations, not yet
00288     committed
00289
00290     This includes some normal reads to pipes between/after
00291     un-committed reservations
00292
00293     This function assumes that the data structure is locked
00294 */
00295 std::size_t reserved_for_reading() const {
00296     return read_reserved_frozen;
00297 }
00298
00299 /** Compute the amount of elements blocked by write reservations, not yet
00300     committed
00301
00302     This includes some normal writes to pipes between/after
00303     un-committed reservations
00304
00305     This function assumes that the data structure is locked
00306 */
00307 std::size_t reserved_for_writing() const {
00308     if (w_rid_q.empty())
00309         // No on-going reservation
00310         return 0;
00311     else
00312         /* The reserved size is from the first element of the first
00313            on-going reservation up to the end of the pipe content */
00314         return cb.end() - w_rid_q.front().start;
00315 }
00316
00317 /** Reserve some part of the pipe for reading
00318
00319     \param[in] s is the number of element to reserve
00320
00321     \param[out] rid is an iterator to a description of the
00322     reservation that has been done if successful
00323
00324     \param[in] blocking specify if the call wait for the operation
00325     to succeed
00326
00327     \return true if the reservation was successful
00328 */
00329 bool reserve_read(std::size_t s,
00330                  rid_iterator &rid,
00331                  bool blocking = false) {
00332     // Lock the pipe to avoid being disturbed
00333     std::unique_lock<std::mutex> ul { cb_mutex };
00334
00335     TRISYCL_DUMP_T("Before read reservation cb.size() = " << cb.size()
00336                   << " size() = " << size());
00337     if (s == 0)
00338         // Empty reservation requested, so nothing to do
00339         return false;
00340
00341     if (blocking)
00342         /* If in blocking mode, wait for enough elements to read in the

```

```

00346         pipe for the reservation. This condition can change when a
00347         write is done */
00348         write_done.wait(ul, [&] { return s <= size(); });
00349     else if (s > size())
00350         // Not enough elements to read in the pipe for the reservation
00351         return false;
00352
00353     // Compute the location of the first element of the reservation
00354     auto first = cb.begin() + read_reserved_frozen;
00355     // Increment the number of frozen elements
00356     read_reserved_frozen += s;
00357     /* Add a description of the reservation at the end of the
00358     reservation queue */
00359     r_rid_q.emplace_back(first, s);
00360     // Return the iterator to the last reservation descriptor
00361     rid = r_rid_q.end() - 1;
00362     TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
00363                   << " size() = " << size());
00364     return true;
00365 }
00366
00367 /** Reserve some part of the pipe for writing
00368
00369     \param[in] s is the number of element to reserve
00370
00371     \param[out] rid is an iterator to a description of the
00372     reservation that has been done if successful
00373
00374     \param[in] blocking specify if the call wait for the operation
00375     to succeed
00376
00377     \return true if the reservation was successful
00378 */
00379 bool reserve_write(std::size_t s,
00380                  rid_iterator &rid,
00381                  bool blocking = false) {
00382     // Lock the pipe to avoid being disturbed
00383     std::unique_lock<std::mutex> ul { cb_mutex };
00384
00385     TRISYCL_DUMP_T("Before write reservation cb.size() = " << cb.size()
00386                   << " size() = " << size());
00387     if (s == 0)
00388         // Empty reservation requested, so nothing to do
00389         return false;
00390
00391     if (blocking)
00392         /* If in blocking mode, wait for enough room in the pipe, that
00393         may be changed when a read is done. Do not use a difference
00394         here because it is only about unsigned values */
00395         read_done.wait(ul, [&] { return cb.size() + s <= capacity(); });
00396     else if (cb.size() + s > capacity())
00397         // Not enough room in the pipe for the reservation
00398         return false;
00399
00400     /* If there is enough room in the pipe, just create default values
00401     in it to do the reservation */
00402     for (std::size_t i = 0; i != s; ++i)
00403         cb.push_back();
00404     /* Compute the location of the first element a posteriori since it
00405     may not exist a priori if cb was empty before */
00406     auto first = cb.end() - s;
00407     /* Add a description of the reservation at the end of the
00408     reservation queue */
00409     w_rid_q.emplace_back(first, s);
00410     // Return the iterator to the last reservation descriptor
00411     rid = w_rid_q.end() - 1;
00412     TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
00413                   << " size() = " << size());
00414     return true;
00415 }
00416
00417 /** Process the read reservations that are ready to be released in the
00418     reservation queue
00419 */
00420 void move_read_reservation_forward() {
00421     // Lock the pipe to avoid nuisance
00422     std::lock_guard<std::mutex> lg { cb_mutex };
00423
00424     for (;;) {
00425         if (r_rid_q.empty())
00426             // No pending reservation, so nothing to do
00427             break;
00428         if (!r_rid_q.front().ready)
00429             /* If the first reservation is not ready to be released, stop
00430             because it is blocking all the following in the queue

```

```

00433         anyway */
00434         break;
00435         // Remove the reservation to be released from the queue
00436         r_rid_q.pop_front();
00437         std::size_t n_to_pop;
00438         if (r_rid_q.empty())
00439             // If it was the last one, remove all the reservation
00440             n_to_pop = read_reserved_frozen;
00441         else
00442             // Else remove everything up to the next reservation
00443             n_to_pop = r_rid_q.front().start - cb.begin();
00444         // No longer take into account these reserved slots
00445         read_reserved_frozen -= n_to_pop;
00446         // Release the elements from the FIFO
00447         while (n_to_pop--)
00448             cb.pop_front();
00449         // Notify the clients waiting for some room to write in the pipe
00450         read_done.notify_all();
00451         /* ...and process the next reservation to see if it is ready to
00452            be released too */
00453     }
00454 }
00455
00456
00457 /** Process the write reservations that are ready to be released in the
00458     reservation queue
00459 */
00460 void move_write_reservation_forward() {
00461     // Lock the pipe to avoid nuisance
00462     std::lock_guard<std::mutex> lg { cb_mutex };
00463
00464     for (;;) {
00465         if (w_rid_q.empty())
00466             // No pending reservation, so nothing to do
00467             break;
00468         // Get the first reservation
00469         const auto &rid = w_rid_q.front();
00470         if (!rid.ready)
00471             /* If the reservation is not ready to be released, stop
00472                because it is blocking all the following in the queue
00473                anyway */
00474             break;
00475         // Remove the reservation to be released from the queue
00476         w_rid_q.pop_front();
00477         // Notify the clients waiting to read something from the pipe
00478         write_done.notify_all();
00479         /* ...and process the next reservation to see if it is ready to
00480            be released too */
00481     }
00482 }
00483
00484 };
00485
00486 /// @} End the execution Doxygen group
00487
00488 }
00489 }
00490 }
00491
00492 /*
00493  # Some Emacs stuff:
00494  ### Local Variables:
00495  ### ispell-local-dictionary: "american"
00496  ### eval: (flyspell-prog-mode)
00497  ### End:
00498 */
00499
00500 #endif // TRISYCL_SYCL_PIPE_DETAIL_PIPE_HPP

```

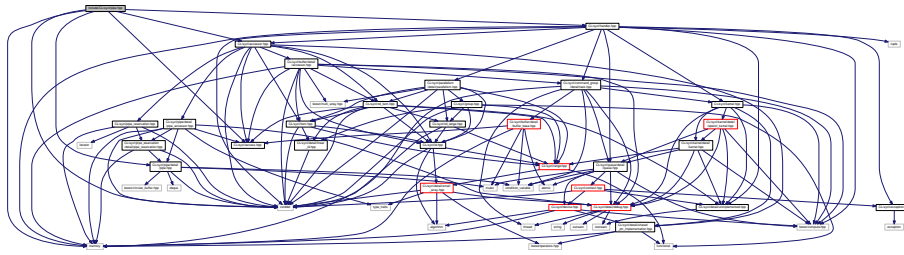
11.105 include/CL/sycl/pipe.hpp File Reference

```

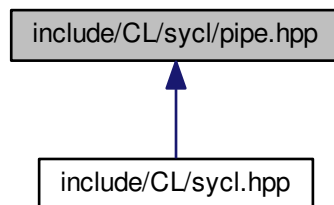
#include <cstdint>
#include <memory>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/pipe/detail/pipe.hpp"

```

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

```

00001 #ifndef TRISYCL_SYCL_PIPE_HPP
00002 #define TRISYCL_SYCL_PIPE_HPP
00003
00004 /** \file The OpenCL SYCL pipe<>
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 #include <cstddef>
00013 #include <memory>
00014
```

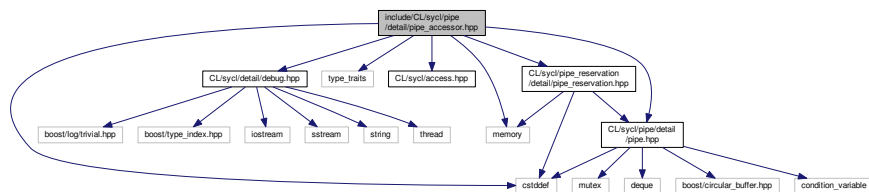
```

00015 #include "CL/sycl/access.hpp"
00016 #include "CL/sycl/accessor.hpp"
00017 #include "CL/sycl/handler.hpp"
00018 #include "CL/sycl/pipe/detail/pipe.hpp"
00019
00020 namespace cl {
00021 namespace sycl {
00022
00023 /** \addtogroup data Data access and storage in SYCL
00024     @{
00025 */
00026
00027 /** A SYCL pipe
00028
00029     Implement a FIFO-style object that can be used through accessors
00030     to send some objects T from the input to the output
00031 */
00032 template <typename T>
00033 class pipe
00034     /* Use the underlying pipe implementation that can be shared in
00035     the SYCL model */
00036 : public detail::shared_ptr_implementation<pipe<T>, detail::pipe<T>>,
00037     detail::debug<pipe<T>> {
00038
00039     // The type encapsulating the implementation
00040     using implementation_t =
00041     detail::shared_ptr_implementation<pipe<T>,
00042     detail::pipe<T>>;
00043
00044     // Make the implementation member directly accessible in this class
00045     using implementation_t::implementation;
00046 public:
00047
00048     /// The STL-like types
00049     /* Since a pipe element cannot be directly addressed without
00050     accessor, only define value_type here */
00051     using value_type = T;
00052
00053
00054     /// Construct a pipe able to store up to capacity T objects
00055     pipe(std::size_t capacity)
00056         : implementation_t { new detail::pipe<T> { capacity } } {}
00057
00058
00059     /** Get an accessor to the pipe with the required mode
00060
00061         \param Mode is the requested access mode
00062
00063         \param Target is the type of pipe access required
00064
00065         \param[in] command_group_handler is the command group handler in
00066         which the kernel is to be executed
00067     */
00068     template <access::mode Mode,
00069             access::target Target = access::target::pipe>
00070     accessor<value_type, 1, Mode, Target>
00071     get_access(handler &command_group_handler) {
00072         static_assert(Target == access::target::pipe
00073             || Target == access::target::blocking_pipe,
00074             "get_access(handler) with pipes can only deal with "
00075             "access::pipe or access::blocking_pipe");
00076         return { implementation, command_group_handler };
00077     }
00078
00079
00080     /// Return the maximum number of elements that can fit in the pipe
00081     std::size_t capacity() const {
00082         return implementation->capacity();
00083     }
00084
00085 };
00086
00087 /// @} End the execution Doxygen group
00088 }
00089 }
00090
00091 /**
00092     # Some Emacs stuff:
00093     ### Local Variables:
00094     ### ispell-local-dictionary: "american"
00095     ### eval: (flyspell-prog-mode)
00096     ### End:
00097 */
00098
00099 #endif // TRISYCL_SYCL_PIPE_HPP

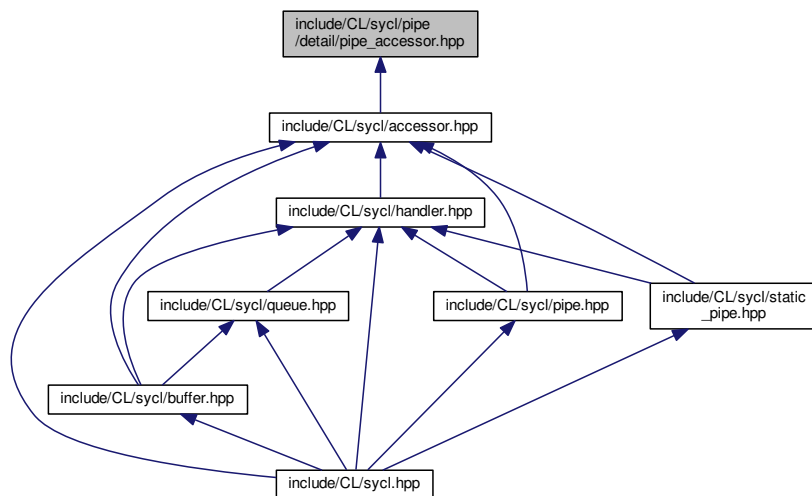
```

11.107 include/CL/sycl/pipe/detail/pipe_accessor.hpp File Reference

```
#include <cstdint>
#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:



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

```

00001 #ifndef TRISYCL_SYCL_PIPE_DETAIL_PIPE_ACCESSOR_HPP
00002 #define TRISYCL_SYCL_PIPE_DETAIL_PIPE_ACCESSOR_HPP
00003
00004 /** \file The OpenCL SYCL pipe accessor<> detail behind the scene
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 #include <stddef>
00013 #include <memory>
00014 #include <type_traits>
00015
00016 #include "CL/sycl/access.hpp"
00017 #include "CL/sycl/detail/debug.hpp"
00018 #include "CL/sycl/pipe/detail/pipe.hpp"
00019 #include "CL/sycl/pipe_reservation/detail/pipe_reservation.hpp"
00020
00021 namespace cl {
00022 namespace sycl {
00023
00024     class handler;
00025
00026     namespace detail {
00027
00028         // Forward declaration of detail::accessor to declare the specialization
00029         template <typename T,
00030                 std::size_t Dimensions,
00031                 access::mode Mode,
00032                 access::target Target>
00033         class accessor;
00034         /** \addtogroup data Data access and storage in SYCL
00035             @{
00036         */
00037
00038         /** The accessor abstracts the way pipe data are accessed inside a
00039             kernel
00040         */
00041         template <typename T,
00042                 access::mode AccessMode,
00043                 access::target Target>
00044         class pipe_accessor :
00045             public detail::debug<detail::pipe_accessor<T, AccessMode, Target>> {
00046
00047         public:
00048
00049             static constexpr auto rank = 1;
00050             static constexpr auto mode = AccessMode;
00051             static constexpr auto target = Target;
00052
00053             static constexpr bool blocking =
00054                 (target == cl::sycl::access::target::blocking_pipe);
00055
00056             /// The STL-like types
00057             using value_type = T;
00058             using reference = value_type&;
00059             using const_reference = const value_type&;
00060
00061         private:
00062
00063             /// The real pipe implementation behind the hood
00064             std::shared_ptr<detail::pipe<T>> implementation;
00065
00066             /** Store the success status of last pipe operation
00067
00068                 It is not impacted by reservation success.
00069             */

```

```

00070         It does exist even if the pipe accessor is not evaluated in a
00071         boolean context for, but a use-def analysis can optimise it out
00072         in that case and not use some storage
00073
00074         Use a mutable state here so that it can work with a [=] lambda
00075         capture without having to declare the whole lambda as mutable
00076     */
00077     bool mutable ok = false;
00078
00079 public:
00080
00081     /** Construct a pipe accessor from an existing pipe
00082     */
00083     pipe_accessor(const std::shared_ptr<detail::pipe<T>> &p,
00084                  handler &command_group_handler) :
00085         implementation { p } {
00086         // TRISYCL_DUMP_T("Create a kernel pipe accessor write = "
00087         //               << is_write_access());
00088         // Verify that the pipe is not already used in the requested mode
00089         if (mode == access::mode::write)
00090             if (implementation->used_for_writing)
00091                 /// \todo Use pipe_exception instead
00092                 throw std::logic_error { "The pipe is already used for writing." };
00093             else
00094                 implementation->used_for_writing = true;
00095         else
00096             if (implementation->used_for_reading)
00097                 throw std::logic_error { "The pipe is already used for reading." };
00098             else
00099                 implementation->used_for_reading = true;
00100     }
00101
00102
00103     pipe_accessor() = default;
00104
00105
00106     /// Return the maximum number of elements that can fit in the pipe
00107     std::size_t capacity() const {
00108         return implementation->capacity();
00109     }
00110
00111     /** Get the current number of elements in the pipe
00112
00113         This is obviously a volatile value which is constrained by
00114         restricted relativity.
00115
00116         Note that on some devices it may be costly to implement (for
00117         example on FPGA).
00118     */
00119     std::size_t size() const {
00120         return implementation->size_with_lock();
00121     }
00122
00123
00124     /** Test if the pipe is empty
00125
00126         This is obviously a volatile value which is constrained by
00127         restricted relativity.
00128
00129         Note that on some devices it may be costly to implement on the
00130         write side (for example on FPGA).
00131     */
00132     bool empty() const {
00133         return implementation->empty_with_lock();
00134     }
00135
00136
00137     /** Test if the pipe is full
00138
00139         This is obviously a volatile value which is constrained by
00140         restricted relativity.
00141
00142         Note that on some devices it may be costly to implement on the
00143         read side (for example on FPGA).
00144     */
00145     bool full() const {
00146         return implementation->full_with_lock();
00147     }
00148
00149
00150     /** In an explicit bool context, the accessor gives the success
00151         status of the last access
00152
00153         It is not impacted by reservation success.
00154
00155         The explicitness is related to avoid \code some_pipe <<
00156         some_value \endcode to be interpreted as \code some_bool <<

```



```

00157         some_value \endcode when the type of \code some_value \endcode
00158         is not the same type as the pipe type.
00159
00160         \return true on success of the previous read or write operation
00161     */
00162     explicit operator bool() const {
00163         return ok;
00164     }
00165
00166
00167     /** Try to write a value to the pipe
00168
00169         \param[in] value is what we want to write
00170
00171         \return this so we can apply a sequence of write for example
00172         (but do not do this on a non blocking pipe...)
00173
00174         \todo provide a && version
00175
00176         This function is const so it can work when the accessor is
00177         passed by copy in the [=] kernel lambda, which is not mutable by
00178         default
00179     */
00180     const pipe_accessor &write(const value_type &value) const {
00181         static_assert(mode == access::mode::write,
00182             "''.write(const value_type &value)\' method on a pipe accessor"
00183             " is only possible with write access mode");
00184         ok = implementation->write(value, blocking);
00185         // Return a reference to *this so we can apply a sequence of write
00186         return *this;
00187     }
00188
00189
00190     /** Some syntactic sugar to use \code a << v \endcode instead of
00191         \code a.write(v) \endcode */
00192     const pipe_accessor &operator<<(const value_type &value) const {
00193         static_assert(mode == access::mode::write,
00194             "'<<' operator on a pipe accessor is only possible"
00195             " with write access mode");
00196         // Return a reference to *this so we can apply a sequence of >>
00197         return write(value);
00198     }
00199
00200
00201     /** Try to read a value from the pipe
00202
00203         \param[out] value is the reference to where to store what is
00204         read
00205
00206         \return \code this \endcode so we can apply a sequence of read
00207         for example (but do not do this on a non blocking pipe...)
00208
00209         This function is const so it can work when the accessor is
00210         passed by copy in the [=] kernel lambda, which is not mutable by
00211         default
00212     */
00213     const pipe_accessor &read(value_type &value) const {
00214         static_assert(mode == access::mode::read,
00215             "''.read(value_type &value)\' method on a pipe accessor"
00216             " is only possible with read access mode");
00217         ok = implementation->read(value, blocking);
00218         // Return a reference to *this so we can apply a sequence of read
00219         return *this;
00220     }
00221
00222
00223     /** Read a value from a blocking pipe
00224
00225         \return the read value directly, since it cannot fail on
00226         blocking pipe
00227
00228         This function is const so it can work when the accessor is
00229         passed by copy in the [=] kernel lambda, which is not mutable by
00230         default
00231     */
00232     value_type read() const {
00233         static_assert(mode == access::mode::read,
00234             "''.read()\' method on a pipe accessor is only possible"
00235             " with read access mode");
00236         static_assert(blocking,
00237             "''.read()\' method on a pipe accessor is only possible"
00238             " with a blocking pipe");
00239         value_type value;
00240         implementation->read(value, blocking);
00241         return value;
00242     }
00243

```

```

00244
00245  /** Some syntactic sugar to use \code a >> v \endcode instead of
00246      \code a.read(v) \endcode */
00247  const pipe_accessor &operator>>(value_type &value) const {
00248      static_assert(mode == access::mode::read,
00249          "'>>' operator on a pipe accessor is only possible"
00250          " with read access mode");
00251      // Return a reference to *this so we can apply a sequence of >>
00252      return read(value);
00253  }
00254
00255
00256  detail::pipe_reservation<pipe_accessor>
reserve(std::size_t size) const {
00257      return { *implementation, size };
00258  }
00259
00260
00261  /// Set debug mode
00262  void set_debug(bool enable) const {
00263      implementation->debug_mode = enable;
00264  }
00265
00266
00267  auto &get_pipe_detail() {
00268      return implementation;
00269  }
00270
00271
00272  ~pipe_accessor() {
00273      /// Free the pipe for a future usage for the current mode
00274      if (mode == access::mode::write)
00275          implementation->used_for_writing = false;
00276      else
00277          implementation->used_for_reading = false;
00278  }
00279
00280 };
00281
00282 /// @} End the data Doxygen group
00283
00284 }
00285 }
00286 }
00287
00288 /*
00289     # Some Emacs stuff:
00290     ### Local Variables:
00291     ### ispell-local-dictionary: "american"
00292     ### eval: (flyspell-prog-mode)
00293     ### End:
00294 */
00295
00296 #endif // TRISYCL_SYCL_PIPE_DETAIL_PIPE_ACCESSOR_HPP

```

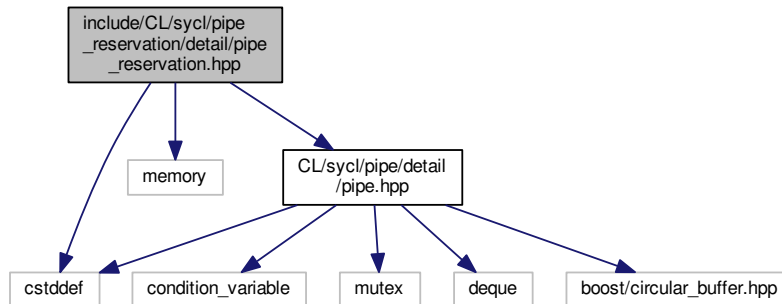
11.109 include/CL/sycl/pipe_reservation/detail/pipe_reservation.hpp File Reference

```

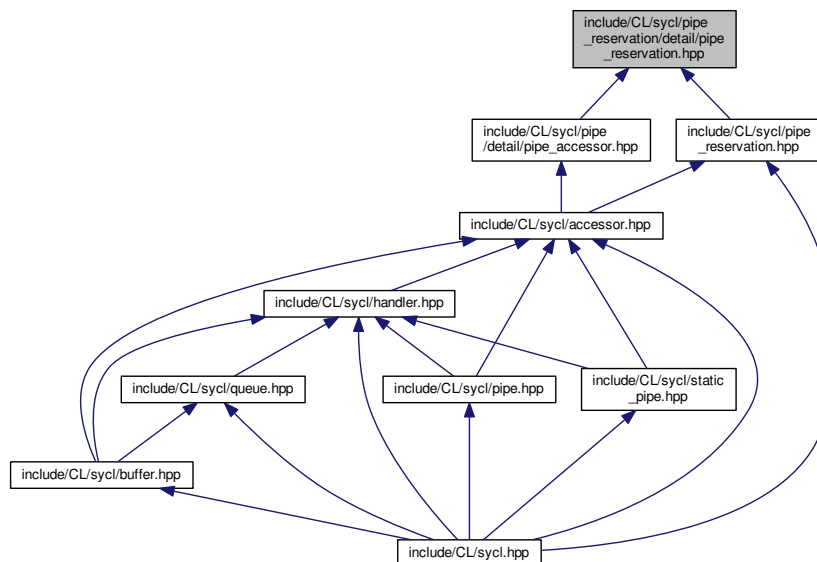
#include <cstdint>
#include <memory>
#include "CL/sycl/pipe/detail/pipe.hpp"

```

Include dependency graph for pipe_reservation.hpp:



This graph shows which files directly or indirectly include this file:



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

```

00001 #ifndef TRISYCL_SYCL_PIPE_RESERVATION_DETAIL_PIPE_RESERVATION_HPP
00002 #define TRISYCL_SYCL_PIPE_RESERVATION_DETAIL_PIPE_RESERVATION_HPP
00003
00004 /** \file The OpenCL SYCL pipe reservation detail behind the scene
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 #include <cstdint>
00013 #include <memory>
00014
00015 #include "CL/sycl/pipe/detail/pipe.hpp"
00016
00017 namespace cl {
00018 namespace sycl {
00019 namespace detail {
00020
00021     template <typename T,
00022               std::size_t Dimensions,
00023               access::mode Mode,
00024               access::target Target>
00025     class accessor;
00026
00027     /** \addtogroup data Data access and storage in SYCL
00028         @{
00029     */
00030
00031     /// The implementation of the pipe reservation station
00032     template <typename PipeAccessor>
00033     class pipe_reservation :
00034     public detail::debug<detail::pipe_reservation<PipeAccessor>> {
00035     using accessor_type = PipeAccessor;
00036     static constexpr bool blocking =
00037     (accessor_type::target ==
00038     cl::sycl::access::target::blocking_pipe);
00038     using value_type = typename accessor_type::value_type;
00039     using reference = typename accessor_type::reference;
00040
00041     public:
00042
00043     using iterator =
00044     typename detail::pipe<value_type>::implementation_t::iterator
00045     ;
00046     using const_iterator =
00047     typename detail::pipe<value_type>::implementation_t::const_iterator
00048     ;
00049
00050     // \todo Add to the specification
00051     static constexpr access::mode mode = accessor_type::mode;
00052     static constexpr access::target target =
00053     accessor_type::target;
00054
00055     /** True if the reservation was successful and still uncommitted. B
00056         default a pipe_reservation is not reserved and cannot be
00057         committed */
00058     bool ok = false;
00059
00060     /// Point into the reservation buffer. Only valid if ok is true
00061     typename detail::pipe<value_type>::rid_iterator
00062     rid;
00063
00064     /** Keep a reference on the pipe to access to the data and methods
00065
00066         Note that with inlining and CSE it should not use more register
00067         when compiler optimization is in use. */
00068     detail::pipe<value_type> &p;
00069
00070     /** Test that the reservation is in a usable state
00071
00072         \todo Throw exception instead
00073     */
00074     void assume_validity() {
00075     assert(ok);
00076     }
00077
00078     public:
00079
00080     /// Create a pipe reservation station that reserves the pipe itself
00081     pipe_reservation(detail::pipe<value_type> &p, std::size_t s) : p
00082     { p } {

```

```

00079     static_assert(mode == access::mode::write
00080                   || mode == access::mode::read,
00081                   "A pipe can only be accessed in read or write mode,"
00082                   " exclusively");
00083
00084     /* Since this test is constexpr and dependent of a template
00085        parameter, it should be equivalent to a specialization of the
00086        method but in a clearer way */
00087     if (mode == access::mode::write)
00088         ok = p.reserve_write(s, rid, blocking);
00089     else
00090         ok = p.reserve_read(s, rid, blocking);
00091 }
00092
00093
00094 /** No copy constructor with some spurious commit in the destructor
00095     of the original object
00096 */
00097 pipe_reservation(const pipe_reservation &) = delete;
00098
00099
00100 /// Only a move constructor is required to move it into the shared_ptr
00101 pipe_reservation(pipe_reservation &&orig) :
00102     ok {orig.ok },
00103     rid {orig.rid },
00104     p { orig.p } {
00105     /* Even when an object is moved, the destructor of the old
00106        object is eventually called, so leave the old object in a
00107        destructable state but without any commit capability */
00108     orig.ok = false;
00109 }
00110
00111
00112 /** Keep the default constructors too
00113
00114     Otherwise there is no move semantics and the copy is made by
00115     creating a new reservation and destructing the old one with a
00116     spurious commit in the meantime...
00117 */
00118 pipe_reservation() = default;
00119
00120
00121 /** Test if the reservation succeeded and thus if the reservation
00122     can be committed
00123
00124     Note that it is up to the user to ensure that all the
00125     reservation elements have been initialized correctly in the case
00126     of a write for example
00127 */
00128 operator bool() {
00129     return ok;
00130 }
00131
00132
00133 /// Start of the reservation area
00134 iterator begin() {
00135     assume_validity();
00136     return rid->start;
00137 }
00138
00139
00140 /// Past the end of the reservation area
00141 iterator end() {
00142     assume_validity();
00143     return rid->start + rid->size;
00144 }
00145
00146
00147 /// Get the number of elements in the reservation station
00148 std::size_t size() {
00149     assume_validity();
00150     return rid->size;
00151 }
00152
00153
00154 /// Access to an element of the reservation
00155 reference operator[](std::size_t index) {
00156     assume_validity();
00157     TRISYCL_DUMP_T("[ index = " << index
00158                    << " Reservation write address = " << &(rid->start[index]));
00159     return rid->start[index];
00160 }
00161
00162
00163
00164 /** Commit the reservation station
00165

```

```

00166     \todo Add to the specification that for simplicity a reservation
00167     can be committed several times but only the first one is taken
00168     into account
00169     */
00170     void commit() {
00171         if (ok) {
00172             // If the reservation is in a committable state, commit
00173             TRISYCL_DUMP_T("Commit");
00174             rid->ready = true;
00175             if (mode == access::mode::write)
00176                 p.move_write_reservation_forward();
00177             else
00178                 p.move_read_reservation_forward();
00179             ok = false;
00180         }
00181     }
00182 }
00183
00184 /// An implicit commit is made in the destructor
00185 ~pipe_reservation() {
00186     commit();
00187 }
00188
00189 };
00190
00191 /// @} End the data Doxygen group
00192
00193 }
00194 }
00195 }
00196
00197 /*
00198     # Some Emacs stuff:
00199     ### Local Variables:
00200     ### ispell-local-dictionary: "american"
00201     ### eval: (flyspell-prog-mode)
00202     ### End:
00203 */
00204
00205 #endif // TRISYCL_SYCL_PIPE_RESERVATION_DETAIL_PIPE_RESERVATION_HPP

```

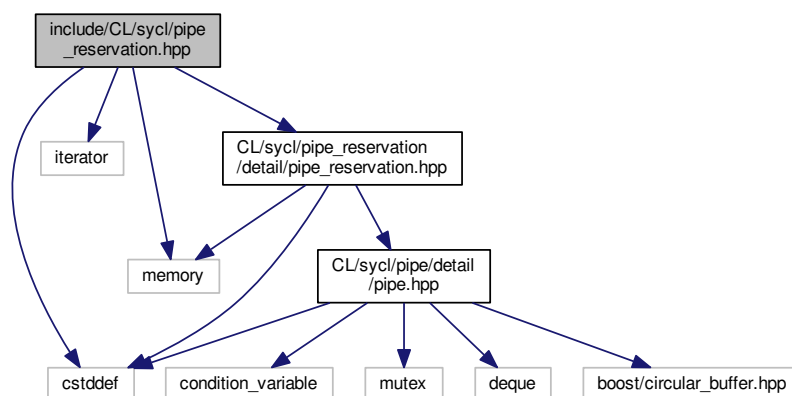
11.111 include/CL/sycl/pipe_reservation.hpp File Reference

```

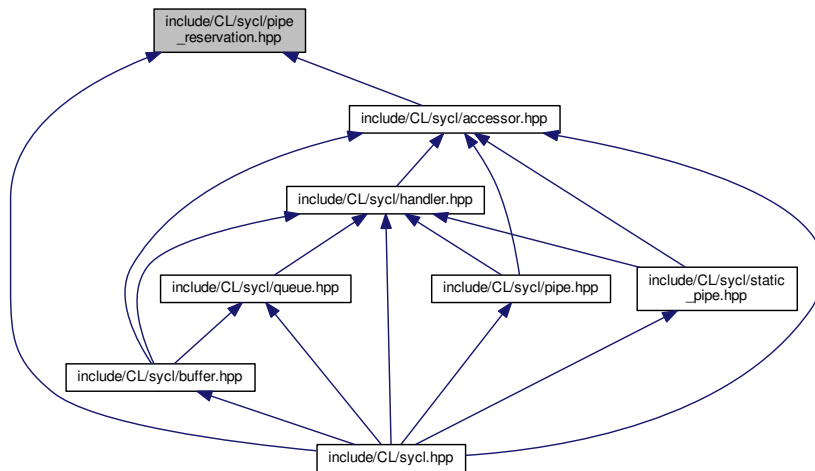
#include <cstddef>
#include <iterator>
#include <memory>
#include "CL/sycl/pipe_reservation/detail/pipe_reservation.hpp"

```

Include dependency graph for pipe_reservation.hpp:



This graph shows which files directly or indirectly include this file:



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

```

00001 #ifndef TRISYCL_SYCL_PIPE_RESERVATION_HPP
00002 #define TRISYCL_SYCL_PIPE_RESERVATION_HPP
00003
00004 /** \file The reservation station for OpenCL SYCL pipe accessor<>
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 #include <cstdint>
00013 #include <iterator>
00014 #include <memory>
00015
00016 #include "CL/sycl/pipe_reservation/detail/pipe_reservation.hpp"
00017
00018 namespace cl {
00019 namespace sycl {
00020
00021 /** \addtogroup data Data access and storage in SYCL
00022     @{
00023 */
00024

```

```

00025 /** The pipe reservation station allows to reserve an array-like view
00026     inside the pipe for ordered race-free access from various
00027     work-items for example
00028 */
00029 template <typename PipeAccessor>
00030 struct pipe_reservation {
00031     using accessor_type = PipeAccessor;
00032     static constexpr bool blocking =
00033         (accessor_type::target ==
00034          cl::sycl::access::target::blocking_pipe);
00034     using accessor_detail = typename accessor_type::accessor_detail;
00035     /// The STL-like types
00036     using value_type = typename accessor_type::value_type;
00037     using reference = value_type&;
00038     using const_reference = const value_type&;
00039     using pointer = value_type*;
00040     using const_pointer = const value_type*;
00041     using size_type = std::size_t;
00042     using difference_type = ptrdiff_t;
00043     using iterator =
00044         typename detail::pipe_reservation<accessor_detail>::iterator
00045 ;
00046     using const_iterator =
00047         typename detail::pipe_reservation<accessor_detail>::const_iterator
00048 ;
00049     using reverse_iterator = std::reverse_iterator<iterator>;
00050     using const_reverse_iterator = std::reverse_iterator<const_iterator>;
00051     /** Point to the underlying implementation that can be shared in the
00052         SYCL model with a handler semantics */
00053     typename std::shared_ptr<detail::pipe_reservation<accessor_detail>>
00054     implementation;
00055     /** Use default constructors so that we can create a new buffer copy
00056         from another one, with either a l-value or a r-value (for
00057         std::move() for example).
00058         Since we just copy the shared_ptr<> above, this is where/how the
00059         sharing magic is happening with reference counting in this case.
00060     */
00061     pipe_reservation() = default;
00062     /// Create a pipe_reservation for an accessor and a number of elements
00063     pipe_reservation(accessor_type &accessor, std::size_t s)
00064         : implementation {
00065             new detail::pipe_reservation<accessor_detail> {
00066                 get_pipe_detail(accessor), s }
00067         } {}
00068     /** Create a pipe_reservation from the implementation detail
00069         This is an internal constructor to allow reserve() on the
00070         implementation to lift a full-fledged object through
00071         accessor::reserve().
00072         \todo Make it private and add required friends
00073     */
00074     pipe_reservation(detail::pipe_reservation<accessor_detail>
00075         &&pr)
00076         : implementation {
00077             new detail::pipe_reservation<accessor_detail> { std::move(pr)
00078         } }
00079     {}
00080     /** Test if the pipe_reservation has been correctly allocated
00081         \return true if the pipe_reservation can be used and committed
00082     */
00083     operator bool() const {
00084         return *implementation;
00085     }
00086     /// Get the number of reserved element(s)
00087     std::size_t size() const {
00088         return implementation->size();
00089     }
00090     /// Access to a given element of the reservation
00091     reference operator[](std::size_t index) const {
00092         return (*implementation)[index];
00093     }
00094 }

```



```

00107
00108  /** Force a commit operation
00109
00110      Normally the commit is implicitly done in the destructor, but
00111      sometime it is useful to do it earlier.
00112  */
00113  void commit() const {
00114      return implementation->commit();
00115  }
00116
00117
00118  /// Get an iterator on the first element of the reservation station
00119  iterator begin() const {
00120      return implementation->begin();
00121  }
00122
00123
00124  /// Get an iterator past the end of the reservation station
00125  iterator end() const {
00126      return implementation->end();
00127  }
00128
00129
00130  /// Build a constant iterator on the first element of the reservation station
00131  const_iterator cbegin() const {
00132      return implementation->begin();
00133  }
00134
00135
00136  /// Build a constant iterator past the end of the reservation station
00137  const_iterator cend() const {
00138      return implementation->end();
00139  }
00140
00141
00142  /// Get a reverse iterator on the last element of the reservation station
00143  reverse_iterator rbegin() const {
00144      return std::make_reverse_iterator(end());
00145  }
00146
00147
00148  /** Get a reverse iterator on the first element past the end of the
00149      reservation station */
00150  reverse_iterator rend() const {
00151      return std::make_reverse_iterator(begin());
00152  }
00153
00154
00155  /** Get a constant reverse iterator on the last element of the
00156      reservation station */
00157  const_reverse_iterator crbegin() const {
00158      return std::make_reverse_iterator(cend());
00159  }
00160
00161
00162  /** Get a constant reverse iterator on the first element past the
00163      end of the reservation station */
00164  const_reverse_iterator crend() const {
00165      return std::make_reverse_iterator(cbegin());
00166  }
00167
00168 };
00169
00170 /// @} End the data Doxygen group
00171
00172 }
00173 }
00174
00175 /*
00176  # Some Emacs stuff:
00177  ### Local Variables:
00178  ###  ispell-local-dictionary: "american"
00179  ###  eval: (flyspell-prog-mode)
00180  ###  End:
00181  */
00182
00183 #endif // TRISYCL_SYCL_PIPE_RESERVATION_HPP

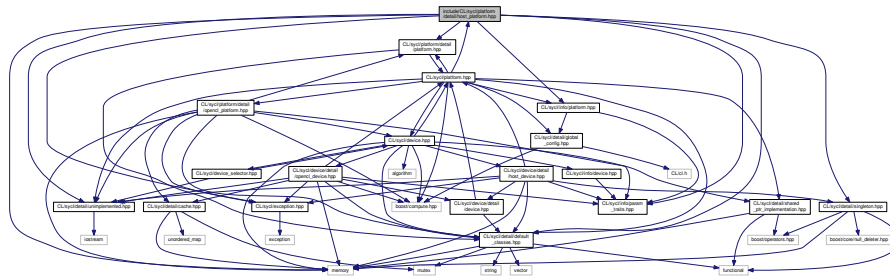
```

11.113 include/CL/sycl/platform/detail/host_platform.hpp File Reference

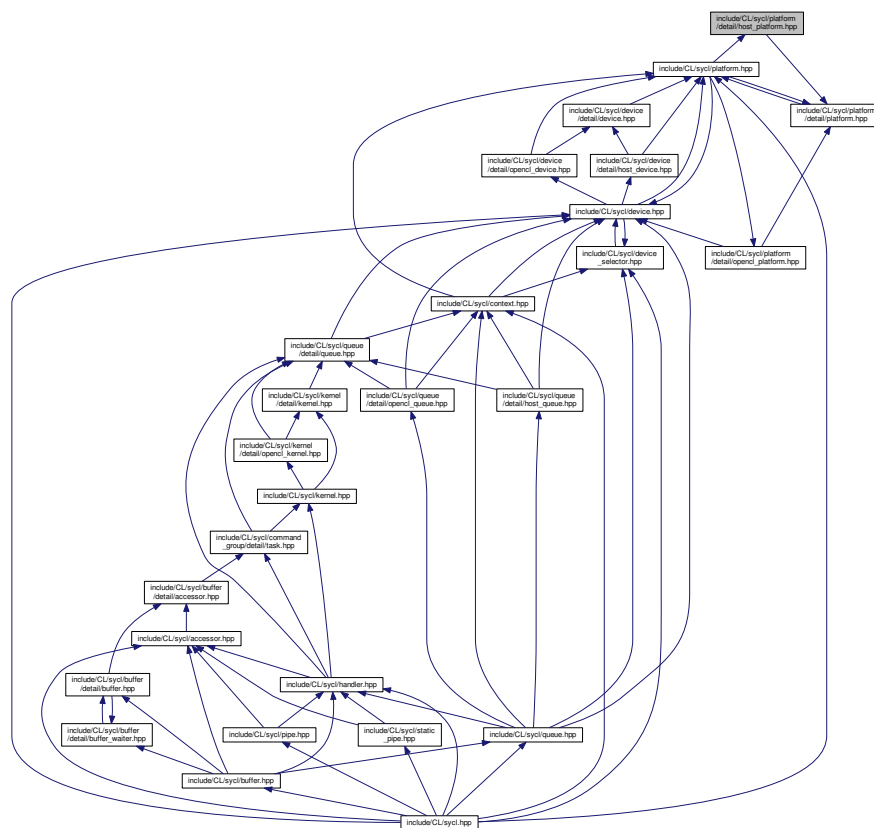
```
#include <memory>
```

```
#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:



This graph shows which files directly or indirectly include this file:



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

```

00001 #ifndef TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_HPP
00002 #define TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_HPP
00003
00004 /** \file The OpenCL triSYCL host platform implementation
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 #include <memory>
00012
00013 #include "CL/sycl/detail/default_classes.hpp"
00014
00015 #include "CL/sycl/detail/singleton.hpp"
00016 #include "CL/sycl/detail/unimplemented.hpp"
00017 #include "CL/sycl/exception.hpp"
00018 #include "CL/sycl/info/param_traits.hpp"
00019 #include "CL/sycl/info/platform.hpp"
00020 #include "CL/sycl/platform/detail/platform.hpp"
00021
00022 namespace cl {
00023 namespace sycl {
00024 namespace detail {
00025
00026 /** \addtogroup execution Platforms, contexts, devices and queues
00027     @{
00028 */
00029
00030 /// SYCL host platform
00031 class host_platform : public detail::platform,
00032                     public detail::singleton<host_platform> {
00033
00034     // \todo Have this compatible with has_extension
00035     auto static constexpr platform_extensions = "Xilinx_blocking_pipes";
00036
00037 public:
00038
00039 #ifndef TRISYCL_OPENCL
00040     /** Return the cl_platform_id of the underlying OpenCL platform
00041
00042         This throws an error since there is no OpenCL platform associated
00043         to the host platform.
00044     */
00045     cl_platform_id get() const override {
00046         throw non_cl_error("The host platform has no OpenCL platform");
00047     }
00048 #endif
00049
00050
00051     /// Return true since this platform is the SYCL host platform
00052     bool is_host() const override {
00053         return true;
00054     }
00055
00056
00057 #if 0
00058     /** Returns at most the host device for this platform, according to
00059         the requested kind
00060
00061         By default returns all the devices, which is obviously the host
00062         one here
00063
00064         \todo To be implemented
00065     */
00066     vector_class<device>
00067     get_devices(info::device_type device_type =
00068         info::device_type::all)
00069         const override
00070     {

```

```

00070     detail::unimplemented();
00071     return {};
00072 }
00073 #endif
00074
00075
00076 /** Returning the information parameters for the host platform
00077     implementation
00078 */
00079 string_class get_info_string(info::platform param) const
00080 override {
00081     switch (param) {
00082     case info::platform::profile:
00083         /* Well... Is the host platform really a full profile whereas it
00084            is not really OpenCL? */
00085         return "FULL_PROFILE";
00086     case info::platform::version:
00087         // \todo I guess it should include the software version too...
00088         return "2.2";
00089     case info::platform::name:
00090         return "triSYCL host platform";
00091     case info::platform::vendor:
00092         return "triSYCL Open Source project";
00093     case info::platform::extensions:
00094         return platform_extensions;
00095     default:
00096         // \todo Define some SYCL exception type for this type of errors
00097         throw std::invalid_argument {
00098             "Unknown parameter value for SYCL platform information" };
00099     }
00100 }
00101
00102 /** Specify whether a specific extension is supported on the platform
00103     \todo To be implemented
00104 */
00105 bool has_extension(const string_class &extension) const override {
00106     detail::unimplemented();
00107     return {};
00108 }
00109 };
00110
00111 /// @} to end the execution Doxygen group
00112
00113 }
00114 }
00115
00116 /**
00117     # Some Emacs stuff:
00118     ### Local Variables:
00119     ### ispell-local-dictionary: "american"
00120     ### eval: (flyspell-prog-mode)
00121     ### End:
00122 */
00123
00124 #endif // TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_HPP

```

11.115 include/CL/sycl/platform/detail/ocl_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"

```

[illegible]

- 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::opencil_kernel > opencil_kernel::cache` `cl::sycl::detail::__attribute__` ((weak))

11.116 opencil_platform.hpp

```

00001 #ifndef TRISYCL_SYCL_PLATFORM_DETAIL_OPENCIL_PLATFORM_HPP
00002 #define TRISYCL_SYCL_PLATFORM_DETAIL_OPENCIL_PLATFORM_HPP
00003
00004 /** \file The OpenCL trisYCL OpenCL platform implementation
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 #include <memory>
00012
00013 #include <boost/compute.hpp>
00014
00015 #include "CL/sycl/detail/default_classes.hpp"
00016
00017 #include "CL/sycl/detail/cache.hpp"
00018 #include "CL/sycl/detail/unimplemented.hpp"
00019 #include "CL/sycl/device.hpp"
00020 #include "CL/sycl/exception.hpp"
00021 #include "CL/sycl/info/param_traits.hpp"
00022 #include "CL/sycl/platform/detail/platform.hpp"
00023
00024 namespace cl {
00025 namespace sycl {
00026
00027     class device;
00028
00029     namespace detail {
00030
00031         /** \addtogroup execution Platforms, contexts, devices and queues
00032             @{
00033         */
00034
00035         /// SYCL OpenCL platform
00036         class opencil_platform : public detail::platform {
00037
00038             /// Use the Boost Compute abstraction of the OpenCL platform
00039             boost::compute::platform p;
00040
00041             /** A cache to always return the same live platform for a given OpenCL
00042                 platform
00043
00044                 C++11 guaranties the static construction is thread-safe
00045             */
00046             static detail::cache<cl_platform_id, detail::opencil_platform>
00047                 cache;
00048
00049         public:
00050
00051             /// Return the cl_platform_id of the underlying OpenCL platform
00052             cl_platform_id get() const override {
00053                 return p.id();
00054             }
00055
00056             /// Return false since an OpenCL platform is not the SYCL host platform
00057             bool is_host() const override {
00058                 return false;
00059             }
00060
00061         }
00062     }
00063 }
00064 }

```

```

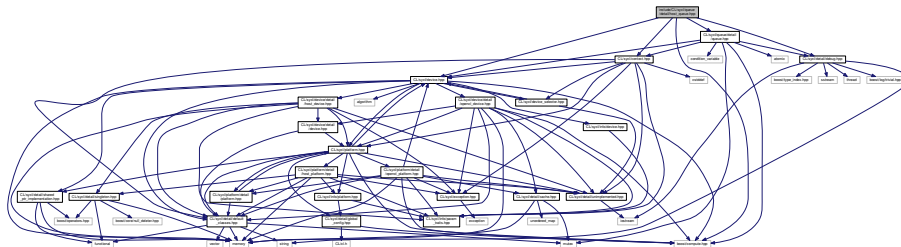
00062 #if 0
00063 /** Returns at most the host device for this platform, according to
00064     the requested kind
00065
00066     By default returns all the devices, which is obviously the host
00067     one here
00068
00069     \todo To be implemented
00070 */
00071 vector_class<cl::sycl::device>
00072 get_devices(info::device_type device_type =
00073     info::device_type::all)
00074     const override
00075 {
00076     detail::unimplemented();
00077     return {};
00078 }
00079 #endif
00080
00081 /// Returning the information string parameters for the OpenCL platform
00082 string_class get_info_string(info::platform param) const
00083     override {
00084     /* Use the fact that the triSYCL info values are the same as the
00085     OpenCL ones used in Boost.Compute to just cast the enum class
00086     to the int value */
00087     return p.get_info<std::string>(static_cast<cl_platform_info>(param));
00088 }
00089
00090 /// Specify whether a specific extension is supported on the platform
00091 bool has_extension(const string_class &extension) const override {
00092     return p.supports_extension(extension);
00093 }
00094
00095
00096 ///// Get a singleton instance of the opencil_platform
00097 static std::shared_ptr<opencil_platform>
00098 instance(const boost::compute::platform &p) {
00099     return cache.get_or_register(p.id(),
00100         [&] { return new opencil_platform { p }; });
00101 }
00102
00103 private:
00104
00105     /// Only the instance factory can built it
00106     opencil_platform(const boost::compute::platform &p) : p { p } {}
00107
00108 public:
00109
00110     /// Unregister from the cache on destruction
00111     ~opencil_platform() override {
00112         cache.remove(p.id());
00113     }
00114
00115 };
00116
00117 /* Allocate the cache here but since this is a pure-header library,
00118     use a weak symbol so that only one remains when SYCL headers are
00119     used in different compilation units of a program
00120 */
00121 detail::cache<cl_platform_id, detail::opencil_platform>
00122 opencil_platform::cache
00123     __attribute__((weak));
00124
00125 /// @} to end the execution Doxygen group
00126 }
00127 }
00128 }
00129
00130 /*
00131     # Some Emacs stuff:
00132     ### Local Variables:
00133     ### ispell-local-dictionary: "american"
00134     ### eval: (flyspell-prog-mode)
00135     ### End:
00136 */
00137
00138 #endif // TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_HPP

```

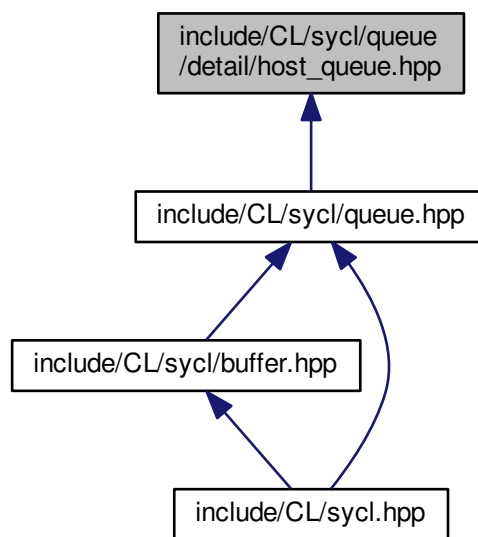
11.117 include/CL/sycl/queue/detail/host_queue.hpp File Reference

```
#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

```

00001 #ifndef TRISYCL_SYCL_QUEUE_DETAIL_HOST_QUEUE_HPP
00002 #define TRISYCL_SYCL_QUEUE_DETAIL_HOST_QUEUE_HPP
00003
00004 /** \file Some implementation details of the host queue
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 #ifndef TRISYCL_OPENCL
00013 #include <boost/compute.hpp>
00014 #endif
00015
00016 #include "CL/sycl/context.hpp"
00017 #include "CL/sycl/detail/debug.hpp"
00018 #include "CL/sycl/device.hpp"
00019 #include "CL/sycl/queue/detail/queue.hpp"
00020
00021 namespace cl {
00022 namespace sycl {
00023 namespace detail {
00024
00025 /** Some implementation details about the SYCL queue
00026
00027     \todo Once a trisYCL queue is no longer blocking, make this a singleton
00028 */
00029 class host_queue : public detail::queue,
00030                   detail::debug<host_queue> {
00031
00032 #ifndef TRISYCL_OPENCL
00033     /** Return the cl_command_queue of the underlying OpenCL queue
00034
00035         This throws an error since there is no OpenCL queue associated
00036         to the host queue.
00037     */
00038     cl_command_queue get() const override {
00039         throw non_cl_error("The host queue has no OpenCL command queue");
00040     }
00041
00042
00043     /** Return the underlying Boost.Compute command queue
00044
00045         This throws an error since there is no OpenCL queue associated
00046         to the host queue.
00047     */
00048     boost::compute::command_queue &get_boost_compute() override {
00049         throw non_cl_error("The host queue has no OpenCL command queue");
00050     }
00051 #endif
00052
00053
00054     /// Return the SYCL host queue's host context
00055     cl::sycl::context get_context() const override {
00056         // Return the default context which is the host context
00057         return {};
00058     }
00059
00060
00061     /// Return the SYCL host device the host queue is associated with
00062     cl::sycl::device get_device() const override {
00063         // Return the default device which is the host device
00064         return {};
00065     }
00066
00067
00068     /// Claim proudly that the queue is executing on the SYCL host device
00069     bool is_host() const override {
00070         return true;
00071     }
00072
00073 };
00074
00075 }
00076 }
00077 }
00078 }
00079
00080 /**
00081     # Some Emacs stuff:
00082     ### Local Variables:
00083     ### ispell-local-dictionary: "american"
00084     ### eval: (flyspell-prog-mode)

```

```

00085     ### End:
00086 */
00087
00088 #endif // TRISYCL_SYCL_QUEUE_DETAIL_HOST_QUEUE_HPP

```

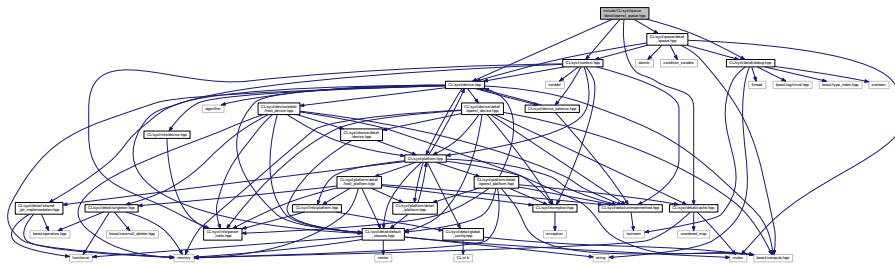
11.119 include/CL/sycl/queue/detail/opengl_queue.hpp File Reference

```

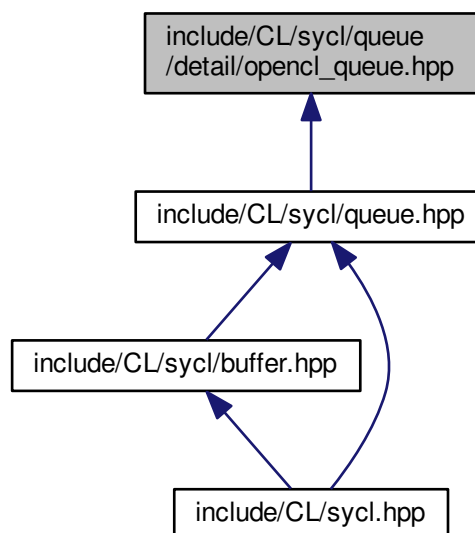
#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/queue/detail/queue.hpp"

```

Include dependency graph for opengl_queue.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [cl::sycl::detail::opengl_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::opencil_kernel > opencil_kernel::cache` `cl::sycl::detail::__attribute__` ((weak))

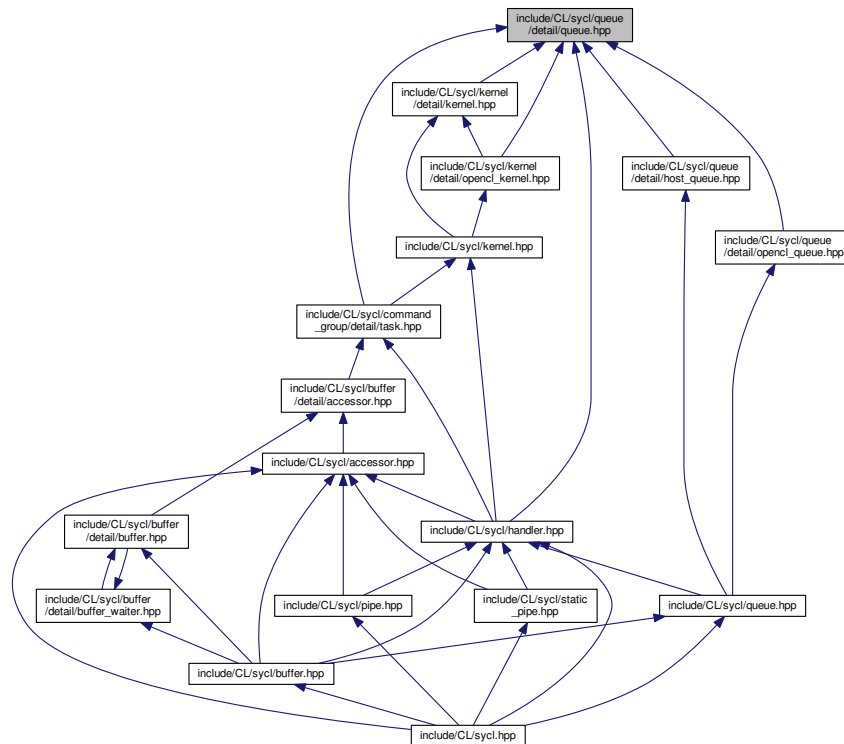
11.120 opencil_queue.hpp

```

00001 #ifndef TRISYCL_SYCL_QUEUE_DETAIL_OPENCIL_QUEUE_HPP
00002 #define TRISYCL_SYCL_QUEUE_DETAIL_OPENCIL_QUEUE_HPP
00003
00004 /** \file Some implementation details of the OpenCL queue
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 #include "CL/sycl/context.hpp"
00013 #include "CL/sycl/detail/cache.hpp"
00014 #include "CL/sycl/detail/debug.hpp"
00015 #include "CL/sycl/device.hpp"
00016 #include "CL/sycl/queue/detail/queue.hpp"
00017
00018 namespace cl {
00019 namespace sycl {
00020 namespace detail {
00021
00022     /// Some implementation details about the SYCL queue
00023     class opencil_queue : public detail::queue,
00024                          detail::debug<opencil_queue> {
00025     /// Use the Boost Compute abstraction of the OpenCL command queue
00026     boost::compute::command_queue q;
00027
00028     /** A cache to always return the same alive queue for a given OpenCL
00029         command queue
00030
00031         C++11 guaranties the static construction is thread-safe
00032         */
00033     static detail::cache<cl_command_queue, detail::opencil_queue>
00034     cache;
00035
00036     /// Return the cl_command_queue of the underlying OpenCL queue
00037     cl_command_queue get() const override {
00038         return q.get();
00039     }
00040
00041     /// Return the underlying Boost.Compute command queue
00042     boost::compute::command_queue &get_boost_compute() override {
00043         return q;
00044     }
00045
00046     /// Return the SYCL context associated to the queue
00047     /// \todo Finish context
00048     cl::sycl::context get_context() const override {
00049         return q.get_context();
00050     }
00051
00052     /// Return the SYCL device associated to the queue
00053     cl::sycl::device get_device() const override {
00054         return q.get_device();
00055     }
00056
00057     /// Claim proudly that an OpenCL queue cannot be the SYCL host queue

```


This graph shows which files directly or indirectly include this file:



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

```

00001 #ifndef TRISYCL_SYCL_QUEUE_DETAIL_QUEUE_HPP
00002 #define TRISYCL_SYCL_QUEUE_DETAIL_QUEUE_HPP
00003
00004 /** \file Some implementation details of queue.
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 #include <atomic>
00013 #include <condition_variable>

```

```

00014 #include <mutex>
00015
00016 #ifdef TRISYCL_OPENCL
00017 #include <boost/compute.hpp>
00018 #endif
00019
00020 #include "CL/sycl/context.hpp"
00021 #include "CL/sycl/device.hpp"
00022 #include "CL/sycl/detail/debug.hpp"
00023
00024 namespace cl {
00025     namespace sycl {
00026         namespace detail {
00027
00028             /** Some implementation details about the SYCL queue
00029             */
00030             struct queue : detail::debug<detail::queue> {
00031                 /** Track the number of kernels still running to wait for their completion
00032                 std::atomic<size_t> running_kernels;
00033
00034                 /** To signal when all the kernels have completed
00035                 std::condition_variable finished;
00036                 /** To protect the access to the condition variable
00037                 std::mutex finished_mutex;
00038
00039
00040                 /** Initialize the queue with 0 running kernel
00041                 queue() {
00042                     running_kernels = 0;
00043                 }
00044
00045
00046                 /** Wait for all kernel completion
00047                 void wait_for_kernel_execution() {
00048                     TRISYCL_DUMP_T("Queue waiting for kernel completion");
00049                     std::unique_lock<std::mutex> ul { finished_mutex };
00050                     finished.wait(ul, [&] {
00051                         // When there is no kernel running in this queue, we are ready to go
00052                         return running_kernels == 0;
00053                     });
00054                 }
00055
00056
00057                 /** Signal that a new kernel started on this queue
00058                 void kernel_start() {
00059                     TRISYCL_DUMP_T("A kernel has been added to the queue");
00060                     // One more kernel
00061                     ++running_kernels;
00062                 }
00063
00064
00065                 /** Signal that a new kernel finished on this queue
00066                 void kernel_end() {
00067                     TRISYCL_DUMP_T("A kernel of the queue ended");
00068                     if (--running_kernels == 0) {
00069                         /* It was the last kernel running, so signal the queue just in
00070                         case it was working for it for completion */
00071                         finished.notify_one();
00072                     }
00073                 }
00074
00075
00076 #ifdef TRISYCL_OPENCL
00077             /** Return the underlying OpenCL command queue after doing a retain
00078
00079                 This memory object is expected to be released by the developer.
00080
00081                 Retain a reference to the returned cl_command_queue object.
00082
00083                 Caller should release it when finished.
00084
00085                 If the queue is a SYCL host queue then an exception is thrown.
00086             */
00087             virtual cl_command_queue get() const = 0;
00088
00089             /** Return the underlying Boost.Compute command queue
00090             virtual boost::compute::command_queue &get_boost_compute() = 0;
00091 #endif
00092
00093
00094             /** Return the SYCL queue's context
00095
00096                 Report errors using SYCL exception classes.
00097             */
00098             virtual cl::sycl::context get_context() const = 0;
00099
00100

```

```

00101  /** Return the SYCL device the queue is associated with
00102
00103      Report errors using SYCL exception classes.
00104  */
00105  virtual cl::sycl::device get_device() const = 0;
00106
00107
00108  /// Return whether the queue is executing on a SYCL host device
00109  virtual bool is_host() const = 0;
00110
00111
00112  /// Wait for all kernel completion before the queue destruction
00113  /// \todo Update according spec since queue destruction is non blocking
00114  virtual ~queue() {
00115      wait_for_kernel_execution();
00116  }
00117
00118 };
00119
00120 }
00121 }
00122 }
00123
00124 /*
00125  # Some Emacs stuff:
00126  ### Local Variables:
00127  ### ispell-local-dictionary: "american"
00128  ### eval: (flyspell-prog-mode)
00129  ### End:
00130 */
00131
00132 #endif // TRISYCL_SYCL_QUEUE_DETAIL_QUEUE_HPP

```

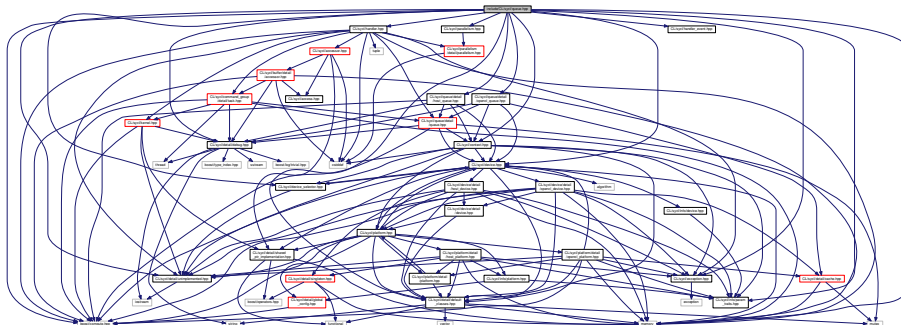
11.123 include/CL/sycl/queue.hpp File Reference

```

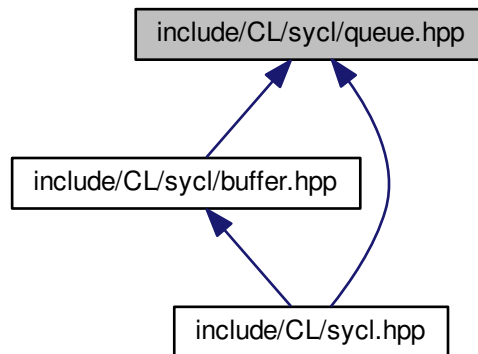
#include <memory>
#include <boost/compute.hpp>
#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/exception.hpp"
#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

```

00001 #ifndef TRISYCL_SYCL_QUEUE_HPP
00002 #define TRISYCL_SYCL_QUEUE_HPP
00003
00004 /** \file The OpenCL SYCL queue
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 #include <memory>
00013
00014 #ifdef TRISYCL_OPENCL
00015 #include <boost/compute.hpp>
00016 #endif
00017
00018 #include "CL/sycl/context.hpp"
00019 #include "CL/sycl/detail/debug.hpp"
00020 #include "CL/sycl/detail/default_classes.hpp"
00021 #include "CL/sycl/detail/unimplemented.hpp"
00022 #include "CL/sycl/device.hpp"
00023 #include "CL/sycl/device_selector.hpp"
00024 #include "CL/sycl/exception.hpp"
00025 #include "CL/sycl/handler.hpp"
00026 #include "CL/sycl/handler_event.hpp"
00027 #include "CL/sycl/info/param_traits.hpp"
00028 #include "CL/sycl/parallelism.hpp"
00029 #include "CL/sycl/queue/detail/host_queue.hpp"
00030 #ifdef TRISYCL_OPENCL
00031 #include "CL/sycl/queue/detail/opengl_queue.hpp"
00032 #endif
00033
00034 namespace cl {
00035 namespace sycl {
00036
00037     class context;
00038     class device_selector;
00039
00040     /** \addtogroup execution Platforms, contexts, devices and queues
00041         @{
00042     */
00043
00044     namespace info {
00045
00046         using queue_profiling = bool;
00047
00048         /** Queue information descriptors
00049
00050             From specification C.4
00051
00052             \todo unsigned int?
00053
00054             \todo To be implemented
00055         */
00056         enum class queue : int {
00057             context,
00058             device,
00059             reference_count,
00060             properties
00061         };
00062
00063         /** Dummy example for get_info() on queue::context that would return a
00064             context
00065
00066             \todo Describe all the types
00067         */
00068         TRISYCL_INFO_PARAM_TRAITS(queue::context,
00069                                     context)
00070     }
00071
00072     /** SYCL queue, similar to the OpenCL queue concept.
00073
00074         \todo The implementation is quite minimal for now. :-)
00075
00076         \todo All the queue methods should return a queue& instead of void
00077         to it is possible to chain ooperations
00078     */
00079     class queue
00080     /* Use the underlying queue implementation that can be shared in
00081        the SYCL model */
00082     : public detail::shared_ptr_implementation<queue, detail::queue>,

```

```

00084     detail::debug<queue> {
00085     // The type encapsulating the implementation
00086     using implementation_t =
00087     detail::shared_ptr_implementation<queue, detail::queue>
00088     ;
00089     // Make the implementation member directly accessible in this class
00090     using implementation_t::implementation;
00091
00092 public:
00093
00094     /** Default constructor for platform which is the host platform
00095
00096     Returns errors via the SYCL exception class.
00097     */
00098     queue() : implementation_t { new detail::host_queue } {}
00099
00100
00101     /** This constructor creates a SYCL queue from an OpenCL queue
00102
00103     At construction it does a retain on the queue memory object.
00104
00105     Retain a reference to the cl_command_queue object. Caller should
00106     release the passed cl_command_queue object when it is no longer
00107     needed.
00108
00109     Return synchronous errors regarding the creation of the queue and
00110     report asynchronous errors via the async_handler callback function
00111     in conjunction with the synchronization and throw methods.
00112
00113     Note that the default case asyncHandler = nullptr is handled by the
00114     default constructor.
00115     */
00116     explicit queue(async_handler asyncHandler) : queue { } {
00117         detail::unimplemented();
00118     }
00119
00120
00121
00122     /** Creates a queue for the device provided by the device selector
00123
00124     If no device is selected, an error is reported.
00125
00126     Return synchronous errors regarding the creation of the queue and
00127     report asynchronous errors via the async_handler callback
00128     function if and only if there is an async_handler provided.
00129     */
00130     queue(const device_selector &deviceSelector,
00131           async_handler asyncHandler = nullptr) : queue { } {
00132         detail::unimplemented();
00133     }
00134
00135
00136     /** A queue is created for syclDevice
00137
00138     Return asynchronous errors via the async_handler callback function.
00139     */
00140     queue(const device &syclDevice,
00141           async_handler asyncHandler = nullptr) : queue { } {
00142         detail::unimplemented();
00143     };
00144
00145
00146     /** This constructor chooses a device based on the provided
00147     device_selector, which needs to be in the given context.
00148
00149     If no device is selected, an error is reported.
00150
00151     Return synchronous errors regarding the creation of the queue.
00152
00153     If and only if there is an asyncHandler provided, it reports
00154     asynchronous errors via the async_handler callback function in
00155     conjunction with the synchronization and throw methods.
00156     */
00157     queue(const context &syclContext,
00158           const device_selector &deviceSelector,
00159           async_handler asyncHandler = nullptr) : queue { } {
00160         detail::unimplemented();
00161     }
00162
00163
00164     /** Creates a command queue using clCreateCommandQueue from a context
00165     and a device
00166
00167     Return synchronous errors regarding the creation of the queue.
00168
00169     If and only if there is an asyncHandler provided, it reports

```

```

00170     asynchronous errors via the async_handler callback function in
00171     conjunction with the synchronization and throw methods.
00172 */
00173 queue(const context &syclContext,
00174        const device &syclDevice,
00175        async_handler asyncHandler = nullptr) : queue { } {
00176     detail::unimplemented();
00177 }
00178
00179 /** Creates a command queue using clCreateCommandQueue from a context
00180     and a device
00181
00182     It enables profiling on the queue if the profilingFlag is set to
00183     true.
00184
00185     Return synchronous errors regarding the creation of the queue. If
00186     and only if there is an asyncHandler provided, it reports
00187     asynchronous errors via the async_handler callback function in
00188     conjunction with the synchronization and throw methods.
00189 */
00190 queue(const context &syclContext,
00191        const device &syclDevice,
00192        info::queue_profiling profilingFlag,
00193        async_handler asyncHandler = nullptr) : queue { } {
00194     detail::unimplemented();
00195 }
00196
00197
00198
00199 #ifndef TRISYCL_OPENCL
00200 /** This constructor creates a SYCL queue from an OpenCL queue
00201
00202     At construction it does a retain on the queue memory object.
00203
00204     Return synchronous errors regarding the creation of the queue. If
00205     and only if there is an async_handler provided, it reports
00206     asynchronous errors via the async_handler callback function in
00207     conjunction with the synchronization and throw methods.
00208 */
00209 queue(const cl_command_queue &q, async_handler ah = nullptr)
00210 : queue { boost::compute::command_queue { q }, ah } {}
00211
00212
00213 /** Construct a queue instance using a boost::compute::command_queue
00214
00215     This is a triSYCL extension for boost::compute interoperation.
00216
00217     Return synchronous errors via the SYCL exception class.
00218
00219     \todo Deal with handler
00220 */
00221 queue(const boost::compute::command_queue &q, async_handler ah = nullptr)
00222 : implementation_t { detail::opencl_queue::instance(q) }
00223 {}
00224 #endif
00225
00226 #ifndef TRISYCL_OPENCL
00227 /** Return the underlying OpenCL command queue after doing a retain
00228
00229     This memory object is expected to be released by the developer.
00230
00231     Retain a reference to the returned cl_command_queue object.
00232
00233     Caller should release it when finished.
00234
00235     If the queue is a SYCL host queue then an exception is thrown.
00236 */
00237 cl_command_queue get() const {
00238     return implementation->get();
00239 }
00240 #endif
00241
00242
00243 /** Return the SYCL queue's context
00244
00245     Report errors using SYCL exception classes.
00246 */
00247 context get_context() const {
00248     return implementation->get_context();
00249 }
00250
00251
00252 /** Return the SYCL device the queue is associated with
00253
00254     Report errors using SYCL exception classes.
00255 */

```

```

00256 device get_device() const {
00257     return implementation->get_device();
00258 }
00259
00260
00261 /// Return whether the queue is executing on a SYCL host device
00262 bool is_host() const {
00263     return implementation->is_host();
00264 }
00265
00266
00267 /** Performs a blocking wait for the completion all enqueued tasks in
00268     the queue
00269
00270     Synchronous errors will be reported through SYCL exceptions.
00271 */
00272 void wait() {
00273     implementation->wait_for_kernel_execution();
00274 }
00275
00276
00277 /** Perform a blocking wait for the completion all enqueued tasks in the queue
00278
00279     Synchronous errors will be reported via SYCL exceptions.
00280
00281     Asynchronous errors will be passed to the async_handler passed to the
00282     queue on construction.
00283
00284     If no async_handler was provided then asynchronous exceptions will
00285     be lost.
00286 */
00287 void wait_and_throw() {
00288     detail::unimplemented();
00289 }
00290
00291
00292 /** Checks to see if any asynchronous errors have been produced by the
00293     queue and if so reports them by passing them to the async_handler
00294     passed to the queue on construction
00295
00296     If no async_handler was provided then asynchronous exceptions will
00297     be lost.
00298 */
00299 void throw_asynchronous() {
00300     detail::unimplemented();
00301 }
00302
00303
00304 /// Queries the platform for cl_command_queue info
00305 template <info::queue param>
00306 typename info::param_traits<info::queue, param>::type
00307 get_info() const {
00308     detail::unimplemented();
00309     return {};
00310 }
00311
00312 /** Submit a command group functor to the queue, in order to be
00313     scheduled for execution on the device
00314
00315     Use an explicit functor parameter taking a handler& so we can use
00316     "auto" in submit() lambda parameter.
00317
00318     \todo Add in the spec an implicit conversion of handler_event to
00319     queue& so it is possible to chain operations on the queue
00320
00321     \todo Update the spec to replace std::function by a templated
00322     type to avoid memory allocation
00323 */
00324 handler_event submit(std::function<void(handler &)> cgf) {
00325     handler command_group_handler { implementation };
00326     cgf(command_group_handler);
00327     return {};
00328 }
00329
00330
00331 /** Submit a command group functor to the queue, in order to be
00332     scheduled for execution on the device
00333
00334     On kernel error, this command group functor, then it is scheduled
00335     for execution on the secondary queue.
00336
00337     Return a command group functor event, which is corresponds to the
00338     queue the command group functor is being enqueued on.
00339 */
00340 handler_event submit(std::function<void(handler &)> cgf,
00341     queue &secondaryQueue) {

```

```

00341     detail::unimplemented();
00342     // Since it is not implemented, always submit on the main queue
00343     return submit(cgf);
00344 }
00345
00346 };
00347
00348 /// @} to end the execution Doxygen group
00349
00350 }
00351 }
00352
00353 /* Inject a custom specialization of std::hash to have the buffer
00354    usable into an unordered associative container
00355
00356    \todo Add this to the spec
00357 */
00358 namespace std {
00359
00360 template <> struct hash<cl::sycl::queue> {
00361
00362     auto operator() (const cl::sycl::queue &q) const {
00363         // Forward the hashing to the implementation
00364         return q.hash();
00365     }
00366 };
00367 };
00368
00369 }
00370
00371 /*
00372     # Some Emacs stuff:
00373     ### Local Variables:
00374     ### ispell-local-dictionary: "american"
00375     ### eval: (flyspell-prog-mode)
00376     ### End:
00377 */
00378
00379 #endif // TRISYCL_SYCL_QUEUE_HPP

```

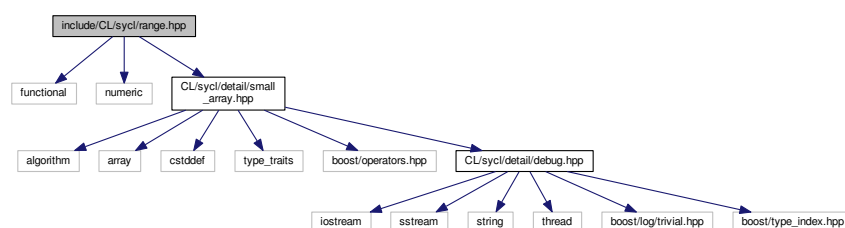
11.125 include/CL/sycl/range.hpp File Reference

```

#include <functional>
#include <numeric>
#include "CL/sycl/detail/small_array.hpp"

```

Include dependency graph for range.hpp:



11.126 range.hpp

```

00001 #ifndef TRISYCL_SYCL_RANGE_HPP
00002 #define TRISYCL_SYCL_RANGE_HPP
00003
00004 /** \file The OpenCL SYCL range<>
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 #include <functional>
00013 #include <numeric>
00014 #include "CL/sycl/detail/small_array.hpp"
00015
00016 namespace cl {
00017 namespace sycl {
00018
00019 /** \addtogroup parallelism Expressing parallelism through kernels
00020     @{
00021 */
00022
00023 /** A SYCL range defines a multi-dimensional index range that can be used
00024     to define launch parallel computation extent or buffer sizes.
00025
00026     \todo use std::size_t dims instead of int dims in the specification?
00027
00028     \todo add to the specification this default parameter value?
00029
00030     \todo add to the specification some way to specify an offset?
00031 */
00032 template <std::size_t dims = 1>
00033 class range : public detail::small_array_123<std::size_t, range<dims>, dims> {
00034
00035 public:
00036
00037     // Inherit of all the constructors
00038     using detail::small_array_123<std::size_t,
00039                                   range<dims>,
00040                                   dims>::small_array_123;
00041
00042
00043     /** Return the number of elements in the range
00044
00045         \todo Give back size() its real meaning in the specification
00046
00047         \todo add this method to the specification
00048     */
00049     size_t get_count() {
00050         // Return the product of the sizes in each dimension
00051         return std::accumulate(this->cbegin(),
00052                                this->cend(),
00053                                1,
00054                                std::multiplies<size_t> {});
00055     }
00056 };
00057
00058
00059 /** Implement a make_range to construct a range<> of the right dimension
00060     with implicit conversion from an initializer list for example.
00061
00062     Cannot use a template on the number of dimensions because the implicit
00063     conversion would not be tried.
00064 */
00065 inline auto make_range(range<1> r) { return r; }
00066 inline auto make_range(range<2> r) { return r; }
00067 inline auto make_range(range<3> r) { return r; }
00068
00069
00070 /** Construct a range<> from a function call with arguments, like
00071     make_range(1, 2, 3)
00072 */
00073 template<typename... BasicType>
00074 auto make_range(BasicType... Args) {
00075     // Call constructor directly to allow narrowing
00076     return range<sizeof...(Args)>(Args...);
00077 }
00078
00079 /// @} End the parallelism Doxygen group
00080
00081 }
00082 }
00083
00084 /*

```

```

00085     # Some Emacs stuff:
00086     ### Local Variables:
00087     ### ispell-local-dictionary: "american"
00088     ### eval: (flyspell-prog-mode)
00089     ### End:
00090 */
00091
00092 #endif // TRISYCL_SYCL_RANGE_HPP

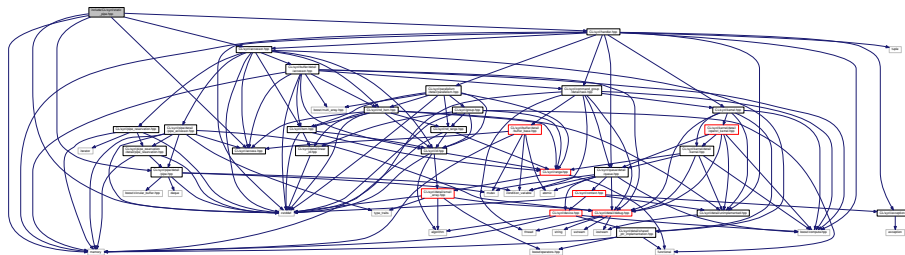
```

11.127 include/CL/sycl/static_pipe.hpp File Reference

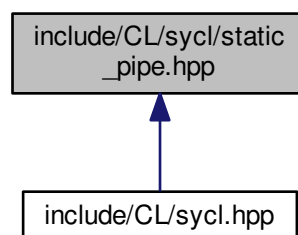
```

#include <cstdint>
#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

```

00001 #ifndef TRISYCL_SYCL_STATIC_PIPE_HPP
00002 #define TRISYCL_SYCL_STATIC_PIPE_HPP
00003
00004 /** \file The OpenCL SYCL static-scoped pipe equivalent to an OpenCL
00005     program-scoped pipe
00006
00007     Ronan at Keryell point FR
00008
00009     This file is distributed under the University of Illinois Open Source
00010     License. See LICENSE.TXT for details.
00011 */
00012
00013 #include <cstdlib>
00014 #include <memory>
00015
00016 #include "CL/sycl/access.hpp"
00017 #include "CL/sycl/accessor.hpp"
00018 #include "CL/sycl/handler.hpp"
00019 #include "CL/sycl/pipe/detail/pipe.hpp"
00020
00021 namespace cl {
00022 namespace sycl {
00023
00024 /** \addtogroup data Data access and storage in SYCL
00025     @{
00026 */
00027
00028 /** A SYCL static-scoped pipe equivalent to an OpenCL program-scoped
00029     pipe
00030
00031     Implement a FIFO-style object that can be used through accessors
00032     to send some objects T from the input to the output.
00033
00034     Compared to a normal pipe, a static_pipe takes a constexpr size
00035     and is expected to be declared in a compile-unit static context so
00036     the compiler can generate everything at compile time.
00037
00038     This is useful to generate a fixed and optimized hardware
00039     implementation on FPGA for example, where the interconnection
00040     graph can be also inferred at compile time.
00041
00042     It is not directly mapped to the OpenCL program-scoped pipe
00043     because in SYCL there is not this concept of separated
00044     program. But the SYCL device compiler is expected to generate some
00045     OpenCL program(s) with program-scoped pipes when a SYCL
00046     static-scoped pipe is used. These details are implementation
00047     defined.
00048 */
00049 template <typename T, std::size_t Capacity>
00050 class static_pipe
00051 {
00052     /* Use the underlying pipe implementation that can be shared in
00053     the SYCL model */
00054     : public detail::shared_ptr_implementation<static_pipe<T, Capacity>,
00055         detail::pipe<T>>,
00056         detail::debug<static_pipe<T, Capacity>> {
00057     // The type encapsulating the implementation
00058     using implementation_t =
00059         detail::shared_ptr_implementation<static_pipe<T, Capacity>
00060         , detail::pipe<T>>;
00061
00062     // Make the implementation member directly accessible in this class
00063     using implementation_t::implementation;
00064
00065 public:
00066
00067     /// The STL-like types
00068     using value_type = T;
00069
00070

```

```

00071  /// Construct a static-scoped pipe able to store up to Capacity T objects
00072  static_pipe()
00073      : implementation_t { new detail::pipe<T> { Capacity } } { }
00074
00075
00076  /** Get an accessor to the pipe with the required mode
00077
00078      \param Mode is the requested access mode
00079
00080      \param Target is the type of pipe access required
00081
00082      \param[in] command_group_handler is the command group handler in
00083          which the kernel is to be executed
00084  */
00085  template <access::mode Mode,
00086            access::target Target = access::target::pipe>
00087  accessor<value_type, 1, Mode, Target>
00088  get_access(handler &command_group_handler) {
00089      static_assert(Target == access::target::pipe
00090                    || Target == access::target::blocking_pipe,
00091                    "get_access(handler) with pipes can only deal with "
00092                    "access::pipe or access::blocking_pipe");
00093      return { implementation, command_group_handler };
00094  }
00095
00096
00097  /** Return the maximum number of elements that can fit in the pipe
00098
00099      This is a constexpr since the capacity is in the type.
00100  */
00101  std::size_t constexpr capacity() const {
00102      return Capacity;
00103  }
00104
00105  };
00106
00107  /// @} End the execution Doxygen group
00108
00109  }
00110  }
00111
00112  /*
00113      # Some Emacs stuff:
00114      ### Local Variables:
00115      ###  ispell-local-dictionary: "american"
00116      ###  eval: (flyspell-prog-mode)
00117      ###  End:
00118  */
00119
00120  #endif // TRISYCL_SYCL_STATIC_PIPE_HPP

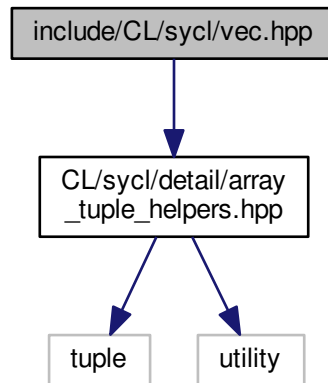
```

11.129 include/CL/sycl/vec.hpp File Reference

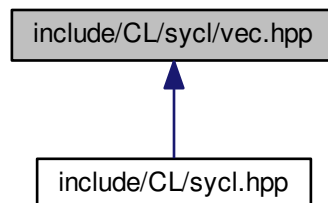
Implement the small OpenCL vector class.

```
#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

```

00001 #ifndef TRISYCL_SYCL_VEC_HPP
00002 #define TRISYCL_SYCL_VEC_HPP
00003
00004 /** \file
00005
00006     Implement the small OpenCL vector class
00007
00008     Ronan at Keryell point FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 #include "CL/sycl/detail/array_tuple_helpers.hpp"
00015
00016 namespace cl {
00017 namespace sycl {
00018
00019 /** \addtogroup vector Vector types in SYCL
00020
00021     @{
00022 */
00023
00024
00025 /** Small OpenCL vector class
00026
00027     \todo add [] operator
00028
00029     \todo add iterators on elements, with begin() and end()
00030
00031     \todo having vec<> sub-classing array<> instead would solve the
00032     previous issues
00033
00034     \todo move the implementation elsewhere
00035
00036     \todo simplify the helpers by removing some template types since there
00037     are now inside the vec<> class.
00038
00039     \todo rename in the specification element_type to value_type
00040 */
00041 template <typename DataType, size_t NumElements>
00042 class vec : public detail::small_array<DataType,
00043                                     vec<DataType, NumElements>,
00044                                     NumElements> {
00045     using basic_type = typename detail::small_array<DataType,
00046                                                     vec<DataType, NumElements>,
00047                                                     NumElements>;
00048
00049 public:

```

```

00050
00051 /** Construct a vec from anything from a scalar (to initialize all the
00052     elements with this value) up to an aggregate of scalar and vector
00053     types (in this case the total number of elements must match the size
00054     of the vector)
00055 */
00056 template <typename... Types>
00057 vec(const Types... args)
00058     : basic_type { detail::expand<vec>(flatten_to_tuple<vec>(args...)) } { }
00059
00060
00061 /// Use classical constructors too
00062 vec() = default;
00063
00064
00065 // Inherit of all the constructors
00066 using typename basic_type::small_array;
00067
00068 private:
00069
00070 /** Flattening helper that does not change scalar values but flatten a
00071     vec<T, n> v into a tuple<T, T,..., T>{ v[0], v[1],..., v[n-1] }
00072
00073     If we have a vector, just forward its array content since an array
00074     has also a tuple interface :-> (23.3.2.9 Tuple interface to class
00075     template array [array.tuple])
00076 */
00077 template <typename V, typename Element, size_t s>
00078 static auto flatten(const vec<Element, s> i) {
00079     static_assert(s <= V::dimension,
00080         "The element i will not fit in the vector");
00081     return static_cast<std::array<Element, s>>(i);
00082 }
00083
00084
00085 /** If we do not have a vector, just forward it as a tuple up to the
00086     final initialization.
00087
00088     \return typically tuple<double>{ 2.4 } from 2.4 input for example
00089 */
00090 template <typename V, typename Type>
00091 static auto flatten(const Type i) {
00092     return std::make_tuple(i);
00093 }
00094
00095
00096 /** Take some initializer values and apply flattening on each value
00097
00098     \return a tuple of scalar initializer values
00099 */
00100 template <typename V, typename... Types>
00101 static auto flatten_to_tuple(const Types... i) {
00102     // Concatenate the tuples returned by each flattening
00103     return std::tuple_cat(flatten<V>(i)...);
00104 }
00105
00106
00107 /// \todo To implement
00108 #if 0
00109 vec<dataT,
00110     numElements>
00111 operator+(const vec<dataT, numElements> &rhs) const;
00112 vec<dataT, numElements>
00113 operator-(const vec<dataT, numElements> &rhs) const;
00114 vec<dataT, numElements>
00115 operator*(const vec<dataT, numElements> &rhs) const;
00116 vec<dataT, numElements>
00117 operator/(const vec<dataT, numElements> &rhs) const;
00118 vec<dataT, numElements>
00119 operator+=(const vec<dataT, numElements> &rhs);
00120 vec<dataT, numElements>
00121 operator-=(const vec<dataT, numElements> &rhs);
00122 vec<dataT, numElements>
00123 operator*=(const vec<dataT, numElements> &rhs);
00124 vec<dataT, numElements>
00125 operator/=(const vec<dataT, numElements> &rhs);
00126 vec<dataT, numElements>
00127 operator+(const dataT &rhs) const;
00128 vec<dataT, numElements>
00129 operator-(const dataT &rhs) const;
00130 vec<dataT, numElements>
00131 operator*(const dataT &rhs) const;
00132 vec<dataT, numElements>
00133 operator/(const dataT &rhs) const;
00134 vec<dataT, numElements>
00135 operator+=(const dataT &rhs);
00136 vec<dataT, numElements>

```

```

00137     operator==(const dataT &rhs);
00138     vec<dataT, numElements>
00139     operator*=(const dataT &rhs);
00140     vec<dataT, numElements>
00141     operator/=(const dataT &rhs);
00142     vec<dataT, numElements> &operator=(const
vec<dataT, numElements> &rhs);
00143     vec<dataT, numElements> &operator=(const dataT &rhs);
00144     bool operator==(const vec<dataT, numElements> &rhs) const;
00145     bool operator!=(const vec<dataT, numElements> &rhs) const;
00146     // Swizzle methods (see notes)
00147     swizzled_vec<T, out_dims> swizzle<int s1, ...>();
00148 #ifdef SYCL_SIMPLE_SWIZZLES
00149     swizzled_vec<T, 4> xyzw();
00150     ...
00151 #endif // #ifdef SYCL_SIMPLE_SWIZZLES
00152 #endif
00153 };
00154
00155 /** A macro to define type alias, such as for type=uchar, size=4 and
00156     real_type=unsigned char, uchar4 is equivalent to vec<float, 4>
00157 */
00158 #define TRISYCL_DEFINE_VEC_TYPE_SIZE(type, size, actual_type) \
00159     using type##size = vec<actual_type, size>;
00160
00161 /// Declare the vector types of a type for all the sizes
00162 #define TRISYCL_DEFINE_VEC_TYPE(type, actual_type) \
00163     TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 1, actual_type) \
00164     TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 2, actual_type) \
00165     TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 3, actual_type) \
00166     TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 4, actual_type) \
00167     TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 8, actual_type) \
00168     TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 16, actual_type)
00169
00170 /// Declare all the possible vector type aliases
00171 TRISYCL_DEFINE_VEC_TYPE(char, char)
00172 TRISYCL_DEFINE_VEC_TYPE(uchar, unsigned char)
00173 TRISYCL_DEFINE_VEC_TYPE(short, short int)
00174 TRISYCL_DEFINE_VEC_TYPE(ushort, unsigned short int)
00175 TRISYCL_DEFINE_VEC_TYPE(int, int)
00176 TRISYCL_DEFINE_VEC_TYPE(uint, unsigned int)
00177 TRISYCL_DEFINE_VEC_TYPE(long, long int)
00178 TRISYCL_DEFINE_VEC_TYPE(ulong, unsigned long int)
00179 TRISYCL_DEFINE_VEC_TYPE(float, float)
00180 TRISYCL_DEFINE_VEC_TYPE(double, double)
00181
00182 /// @} End the vector Doxygen group
00183
00184
00185 }
00186 }
00187
00188 /*
00189     # Some Emacs stuff:
00190     ### Local Variables:
00191     ### ispell-local-dictionary: "american"
00192     ### eval: (flyspell-prog-mode)
00193     ### End:
00194 */
00195
00196 #endif // TRISYCL_SYCL_VEC_HPP

```

Index

- `__SYCL_SINGLE_SOURCE__`
 - Manage default configuration and types, [224](#)
- `__attribute__`
 - Platforms, contexts, devices and queues, [215](#), [216](#)
- `~buffer`
 - `cl::sycl::detail::buffer`, [65](#)
- `~buffer_base`
 - `cl::sycl::detail::buffer_base`, [354](#)
- `~buffer_waiter`
 - `cl::sycl::detail::buffer_waiter`, [70](#)
- `~device`
 - `cl::sycl::detail::device`, [153](#)
- `~device_selector`
 - `cl::sycl::device_selector`, [168](#)
- `~opencl_device`
 - `cl::sycl::detail::opencl_device`, [377](#)
- `~opencl_platform`
 - `cl::sycl::detail::opencl_platform`, [188](#)
- `~opencl_queue`
 - `cl::sycl::detail::opencl_queue`, [388](#)
- `~pipe_accessor`
 - `cl::sycl::detail::pipe_accessor`, [87](#)
- `~pipe_reservation`
 - `cl::sycl::detail::pipe_reservation`, [100](#)
- `~platform`
 - `cl::sycl::detail::platform`, [191](#)
- `~queue`
 - `cl::sycl::detail::queue`, [393](#)
- `accelerator`
 - Platforms, contexts, devices and queues, [212](#)
- `access`
 - `cl::sycl::detail::buffer`, [68](#)
- `accessor`
 - `cl::sycl::accessor`, [31](#), [32](#)
 - `cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >`, [43](#)
 - `cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >`, [41](#)
 - `cl::sycl::detail::accessor`, [48](#)
- `accessor_detail`
 - `cl::sycl::accessor`, [30](#)
 - `cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >`, [43](#)
 - `cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >`, [41](#)
 - `cl::sycl::pipe_reservation`, [107](#)
- `accessor_type`
 - `cl::sycl::detail::pipe_reservation`, [98](#)
 - `cl::sycl::pipe_reservation`, [107](#)
- `add_buffer`
 - `cl::sycl::detail::task`, [407](#)
- `add_buffer_to_task`
 - `cl::sycl::detail`, [334](#)
- `add_postlude`
 - `cl::sycl::detail::task`, [407](#)
- `add_prelude`
 - `cl::sycl::detail::task`, [407](#)
- `add_to_task`
 - `cl::sycl::detail::buffer_base`, [355](#)
- `addr_space`
 - Dealing with OpenCL address spaces, [140](#)
- `address_bits`
 - Platforms, contexts, devices and queues, [208](#)
- `address_space`
 - `cl::sycl::detail::address_space_base`, [137](#)
 - Dealing with OpenCL address spaces, [142](#)
- `address_space_array`
 - `cl::sycl::detail::address_space_array`, [127](#)
- `address_space_fundamental`
 - `cl::sycl::detail::address_space_fundamental`, [130](#)
- `address_space_object`
 - `cl::sycl::detail::address_space_object`, [132](#)
- `address_space_ptr`
 - `cl::sycl::detail::address_space_ptr`, [135](#)
- `address_space_variable`
 - `cl::sycl::detail::address_space_variable`, [139](#)
- `all`
 - Platforms, contexts, devices and queues, [212](#)
- `allocation`
 - `cl::sycl::detail::buffer`, [68](#)
- `allocator_type`
 - `cl::sycl::buffer`, [341](#)
- `array`
 - `cl::sycl::detail::accessor`, [59](#)
- `array_view_type`
 - `cl::sycl::detail::accessor`, [47](#)
- `assume_validity`
 - `cl::sycl::detail::pipe_reservation`, [101](#)
- `async_handler`
 - Error handling, [265](#)
- `atomic`
 - `cl::sycl::access`, [329](#)
- `barrier`
 - `cl::sycl::nd_item`, [287](#)
- `basic_type`
 - `cl::sycl::vec`, [319](#)
- `begin`
 - `cl::sycl::accessor`, [33](#)

- cl::sycl::detail::accessor, 49
 - cl::sycl::detail::pipe_reservation, 101
 - cl::sycl::pipe_reservation, 110
- blocking
 - cl::sycl::detail::pipe_accessor, 92
 - cl::sycl::detail::pipe_reservation, 105
 - cl::sycl::pipe_reservation, 115
- blocking_pipe
 - cl::sycl::access, 330
- buf
 - cl::sycl::detail::accessor, 59
- buffer
 - cl::sycl::buffer, 342–348
 - cl::sycl::detail::buffer, 63–65
- buffer_add_to_task
 - Data access and storage in SYCL, 118
- buffer_allocator
 - Data access and storage in SYCL, 118
- buffer_base
 - cl::sycl::detail::buffer_base, 354
- buffer_waiter
 - cl::sycl::detail::buffer_waiter, 70
- buffers_in_use
 - cl::sycl::detail::task, 413
- built_in_kernels
 - Platforms, contexts, devices and queues, 209
- c
 - cl::sycl::detail::cache, 361
- CL_SYCL_LANGUAGE_VERSION
 - Manage default configuration and types, 224
- CL_TRISYCL_LANGUAGE_VERSION
 - Manage default configuration and types, 224
- cache
 - cl::sycl::detail::opencl_device, 381
 - cl::sycl::detail::opencl_kernel, 385
 - cl::sycl::detail::opencl_platform, 190
 - cl::sycl::detail::opencl_queue, 391
- capacity
 - cl::sycl::detail::pipe, 75
 - cl::sycl::detail::pipe_accessor, 87
 - cl::sycl::pipe, 95
 - cl::sycl::static_pipe, 117
- cb
 - cl::sycl::detail::pipe, 82
- cb_mutex
 - cl::sycl::detail::pipe, 82
- cbegin
 - cl::sycl::accessor, 33
 - cl::sycl::detail::accessor, 50
 - cl::sycl::pipe_reservation, 110
- cend
 - cl::sycl::accessor, 33
 - cl::sycl::detail::accessor, 50
 - cl::sycl::pipe_reservation, 111
- cl, 323
- cl::sycl, 323
 - function_class, 327
 - mutex_class, 327
 - shared_ptr_class, 327
 - string_class, 327
 - unique_ptr_class, 327
 - vector_class, 328
 - weak_ptr_class, 328
- cl::sycl::access, 328
 - atomic, 329
 - blocking_pipe, 330
 - constant_buffer, 330
 - discard_read_write, 329
 - discard_write, 329
 - fence_space, 329
 - global_and_local, 329
 - global_buffer, 330
 - global_space, 329
 - host_buffer, 330
 - host_image, 330
 - image, 330
 - image_array, 330
 - local, 330
 - local_space, 329
 - mode, 329
 - pipe, 330
 - read, 329
 - read_write, 329
 - target, 329
 - write, 329
- cl::sycl::accessor, 28
 - accessor, 31, 32
 - accessor_detail, 30
 - begin, 33
 - cbegin, 33
 - cend, 33
 - const_reference, 30
 - crbegin, 34
 - crend, 34
 - dimensionality, 39
 - end, 34
 - implementation_t, 30
 - operator*, 34, 35
 - operator[], 35–38
 - rbegin, 38
 - reference, 30
 - rend, 39
 - value_type, 30
- cl::sycl::accessor< DataType, 1, AccessMode, access_mode::target::blocking_pipe >, 42
 - accessor, 43
 - accessor_detail, 43
 - get_pipe_detail, 43
 - reserve, 43
- cl::sycl::accessor< DataType, 1, AccessMode, access_mode::target::pipe >, 39
 - accessor, 41
 - accessor_detail, 41
 - get_pipe_detail, 41
 - reserve, 41
- cl::sycl::accessor_error, 250

- cl::sycl::async_exception, 247
- cl::sycl::buffer
 - allocator_type, 341
 - buffer, 342–348
 - const_reference, 341
 - get_access, 349, 350
 - get_count, 350
 - get_range, 350
 - get_size, 351
 - implementation_t, 341
 - is_read_only, 351
 - reference, 342
 - set_final_data, 351
 - use_count, 352
 - value_type, 342
- cl::sycl::buffer< T, Dimensions, Allocator >, 339
- cl::sycl::cl_exception, 246
 - cl_code, 247
 - cl_exception, 247
 - get_cl_code, 247
- cl::sycl::compile_program_error, 256
- cl::sycl::context, 147
 - context, 148–150
 - get, 151
 - get_devices, 151
 - get_info, 151
 - get_platform, 152
 - is_host, 152
- cl::sycl::detail, 330
 - add_buffer_to_task, 334
- cl::sycl::detail::accessor, 44
 - accessor, 48
 - array, 59
 - array_view_type, 47
 - begin, 49
 - buf, 59
 - cbegin, 50
 - cend, 50
 - cl_buf, 60
 - const_iterator, 47
 - const_reference, 47
 - const_reverse_iterator, 47
 - copy_back_cl_buffer, 50
 - copy_in_cl_buffer, 51
 - crbegin, 52
 - crend, 52
 - dimensionality, 60
 - element, 47
 - end, 52
 - get_buffer, 52
 - get_cl_buffer, 53
 - get_size, 53
 - handler, 60
 - is_read_access, 54
 - is_write_access, 54
 - iterator, 47
 - operator*, 55
 - operator[], 56–58
 - rbegin, 59
 - reference, 47
 - rend, 59
 - reverse_iterator, 48
 - task, 60
 - value_type, 48
 - writable_array_view_type, 48
- cl::sycl::detail::address_space_array, 125
 - address_space_array, 127
 - super, 126
- cl::sycl::detail::address_space_base, 135
 - address_space, 137
 - opencl_type, 136
 - type, 136
- cl::sycl::detail::address_space_fundamental, 127
 - address_space_fundamental, 130
 - super, 129
- cl::sycl::detail::address_space_object, 130
 - address_space_object, 132
 - opencl_type, 132
 - operator opencl_type &, 132
- cl::sycl::detail::address_space_ptr, 132
 - address_space_ptr, 135
 - super, 134
- cl::sycl::detail::address_space_variable, 137
 - address_space_variable, 139
 - get_address, 139
 - opencl_type, 139
 - operator opencl_type &, 139
 - super, 139
 - variable, 140
- cl::sycl::detail::buffer, 60
 - ~buffer, 65
 - access, 68
 - allocation, 68
 - buffer, 63–65
 - detail::accessor, 68
 - element, 63
 - final_data, 69
 - get_access, 65
 - get_count, 66
 - get_destructor_future, 66
 - get_range, 67
 - get_size, 67
 - host_write_back, 69
 - set_final_data, 68
 - shared_data, 69
 - value_type, 63
- cl::sycl::detail::buffer_base, 353
 - ~buffer_base, 354
 - add_to_task, 355
 - buffer_base, 354
 - get_latest_producer, 355
 - latest_producer, 357
 - latest_producer_mutex, 357
 - notify_buffer_destructor, 357
 - number_of_users, 357
 - read_only, 358

- ready, 358
- ready_mutex, 358
- release, 356
- set_latest_producer, 356
- use, 356
- wait, 356
- cl::sycl::detail::buffer_waiter, 69
 - ~buffer_waiter, 70
 - buffer_waiter, 70
 - implementation_t, 70
- cl::sycl::detail::cache
 - c, 361
 - get_or_register, 359
 - key_type, 359
 - m, 361
 - remove, 360
 - value_type, 359
- cl::sycl::detail::cache< Key, Value >, 358
- cl::sycl::detail::debug, 221
- cl::sycl::detail::device, 152
 - ~device, 153
 - get, 154
 - get_platform, 154
 - has_extension, 154
 - is_accelerator, 154
 - is_cpu, 154
 - is_gpu, 154
 - is_host, 154
- cl::sycl::detail::display_vector, 221
 - display, 222
- cl::sycl::detail::expand_to_vector, 217
- cl::sycl::detail::expand_to_vector< V, Tuple, true >, 217
- cl::sycl::detail::host_device, 369
 - get, 371
 - get_platform, 371
 - has_extension, 371
 - is_accelerator, 372
 - is_cpu, 372
 - is_gpu, 372
 - is_host, 373
- cl::sycl::detail::host_platform, 183
 - get, 184
 - get_info_string, 184
 - has_extension, 185
 - is_host, 185
 - platform_extensions, 186
- cl::sycl::detail::host_queue, 373
 - get, 374
 - get_boost_compute, 374
 - get_context, 375
 - get_device, 375
 - is_host, 375
- cl::sycl::detail::kernel, 178
 - get, 180
 - get_boost_compute, 180
 - TRISYCL_ParallelForKernel_RANGE, 180
- cl::sycl::detail::opencil_device, 376
 - ~opencil_device, 377
- cache, 381
- d, 381
- get, 378
- get_platform, 378
- has_extension, 378
- instance, 379
- is_accelerator, 380
- is_cpu, 380
- is_gpu, 380
- is_host, 380
- opencil_device, 377
- cl::sycl::detail::opencil_kernel, 382
 - cache, 385
 - get, 383
 - get_boost_compute, 383
 - instance, 384
 - k, 385
 - opencil_kernel, 383
 - TRISYCL_ParallelForKernel_RANGE, 385
- cl::sycl::detail::opencil_platform, 186
 - ~opencil_platform, 188
 - cache, 190
 - get, 188
 - get_info_string, 188
 - has_extension, 189
 - instance, 189
 - is_host, 190
 - opencil_platform, 188
 - p, 190
- cl::sycl::detail::opencil_queue, 386
 - ~opencil_queue, 388
 - cache, 391
 - get, 389
 - get_boost_compute, 389
 - get_context, 389
 - get_device, 389
 - instance, 390
 - is_host, 390
 - opencil_queue, 388
 - q, 391
- cl::sycl::detail::opencil_type, 122
 - type, 123
- cl::sycl::detail::opencil_type< T, constant_address_space >, 123
 - type, 123
- cl::sycl::detail::opencil_type< T, generic_address_space >, 123
 - type, 123
- cl::sycl::detail::opencil_type< T, global_address_space >, 123
 - type, 124
- cl::sycl::detail::opencil_type< T, local_address_space >, 124
 - type, 124
- cl::sycl::detail::opencil_type< T, private_address_space >, 124
 - type, 125
- cl::sycl::detail::parallel_OpenMP_for_iterate, 304

- parallel_OpenMP_for_iterate, 304
- cl::sycl::detail::parallel_for_iterate, 303
 - parallel_for_iterate, 304
- cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >, 305
 - parallel_for_iterate, 305
- cl::sycl::detail::pipe, 72
 - capacity, 75
 - cb, 82
 - cb_mutex, 82
 - debug_mode, 83
 - empty, 75
 - empty_with_lock, 75
 - full, 76
 - full_with_lock, 76
 - implementation_t, 74
 - move_read_reservation_forward, 76
 - move_write_reservation_forward, 77
 - pipe, 75
 - r_rid_q, 83
 - read, 77
 - read_done, 83
 - read_reserved_frozen, 83
 - reserve_read, 78
 - reserve_write, 79
 - reserved_for_reading, 80
 - reserved_for_writing, 80
 - rid_iterator, 74
 - size, 81
 - size_with_lock, 81
 - used_for_reading, 83
 - used_for_writing, 83
 - value_type, 75
 - w_rid_q, 83
 - write, 82
 - write_done, 83
- cl::sycl::detail::pipe_accessor, 84
 - ~pipe_accessor, 87
 - blocking, 92
 - capacity, 87
 - const_reference, 86
 - empty, 87
 - full, 87
 - get_pipe_detail, 88
 - implementation, 92
 - mode, 92
 - ok, 93
 - operator bool, 88
 - operator<<, 88
 - operator>>, 89
 - pipe_accessor, 86
 - rank, 93
 - read, 89, 90
 - reference, 86
 - reserve, 90
 - set_debug, 91
 - size, 91
 - target, 93
 - value_type, 86
 - write, 91
- cl::sycl::detail::pipe_reservation, 95
 - ~pipe_reservation, 100
 - accessor_type, 98
 - assume_validity, 101
 - begin, 101
 - blocking, 105
 - commit, 102
 - const_iterator, 98
 - end, 103
 - iterator, 98
 - mode, 105
 - ok, 105
 - operator bool, 103
 - operator[], 103
 - p, 105
 - pipe_reservation, 98, 99
 - reference, 98
 - rid, 105
 - size, 104
 - target, 105
 - value_type, 98
- cl::sycl::detail::platform, 191
 - ~platform, 191
 - get, 192
 - get_info_string, 192
 - has_extension, 192
 - is_host, 192
- cl::sycl::detail::queue, 392
 - ~queue, 393
 - finished, 398
 - finished_mutex, 398
 - get, 394
 - get_boost_compute, 394
 - get_context, 395
 - get_device, 395
 - is_host, 396
 - kernel_end, 396
 - kernel_start, 397
 - queue, 393
 - running_kernels, 398
 - wait_for_kernel_execution, 397
- cl::sycl::detail::reserve_id, 71
 - ready, 72
 - reserve_id, 71
 - size, 72
 - start, 72
- cl::sycl::detail::shared_ptr_implementation
 - hash, 401
 - implementation, 403
 - operator<, 402
 - operator==, 402
 - shared_ptr_implementation, 400, 401
- cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >, 399
- cl::sycl::detail::singleton
 - instance, 403

- cl::sycl::detail::singleton< T >, 403
- cl::sycl::detail::small_array, 226
 - dimension, 230
 - dimensionality, 230
 - element_type, 228
 - get, 230
 - operator FinalType, 230
 - small_array, 228, 229
- cl::sycl::detail::small_array_123, 230
- cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >, 231
 - operator BasicType, 233
 - small_array_123, 233
- cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >, 233
 - small_array_123, 234
- cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >, 235
 - small_array_123, 236
- cl::sycl::detail::task, 404
 - add_buffer, 407
 - add_postlude, 407
 - add_prelude, 407
 - buffers_in_use, 413
 - epilogues, 413
 - execution_ended, 413
 - get_kernel, 408
 - get_queue, 408
 - kernel, 413
 - notify_consumers, 408
 - owner_queue, 414
 - postlude, 409
 - prelude, 409
 - producer_tasks, 414
 - prologues, 414
 - ready, 414
 - ready_mutex, 414
 - release_buffers, 410
 - schedule, 410
 - set_kernel, 412
 - task, 407
 - wait, 412
 - wait_for_producers, 412
- cl::sycl::device, 154
 - device, 157
 - get, 158
 - get_info, 158, 159
 - get_platform, 159
 - has_extension, 159
 - implementation_t, 156
 - is_accelerator, 160
 - is_cpu, 160
 - is_gpu, 160
 - is_host, 161
 - type, 161
- cl::sycl::device_error, 255
- cl::sycl::device_selector, 167
 - ~device_selector, 168
- operator(), 169
 - select_device, 169
- cl::sycl::device_type_selector, 162
 - default_device, 165
 - device_type, 165
 - device_type_selector, 164
 - operator(), 165
- cl::sycl::device_type_name_selector, 166
 - device_type_name_selector, 167
- cl::sycl::error_handler, 241
 - default_handler, 242
 - report_error, 242
- cl::sycl::event, 363
 - event, 363
- cl::sycl::event_error, 253
- cl::sycl::exception, 243
 - exception, 245
 - message, 246
 - what, 245
- cl::sycl::exception_list, 242
- cl::sycl::feature_not_supported, 263
- cl::sycl::group, 267
 - dimensionality, 276
 - get, 270
 - get_global_range, 270
 - get_group_range, 271
 - get_linear, 271
 - get_local_range, 271, 272
 - get_nd_range, 272
 - get_offset, 273
 - group, 269
 - group_id, 276
 - ndr, 276
 - operator[], 274
 - parallel_for_work_item, 274, 275
- cl::sycl::handler, 169
 - Dimensions, 178
 - dispatch_set_arg, 171
 - handler, 171
 - parallel_for, 172
 - parallel_for_work_group, 172
 - set_arg, 174, 175
 - set_args, 175
 - single_task, 176
 - TRISYCL_ParallelForKernel_RANGE, 178
 - TRISYCL_parallel_for_functor_GLOBAL, 177
 - task, 178
- cl::sycl::id, 276
 - id, 277
- cl::sycl::image, 70
- cl::sycl::info, 334
 - context, 336, 337
 - device, 337
 - gl_context_interop, 336
 - gl_interop, 336
 - num_devices, 336
 - properties, 337
 - queue, 336

- queue_profiling, 336
- reference_count, 336, 337
- cl::sycl::info::param_traits< T, Param >, 391
- cl::sycl::invalid_object_error, 258
- cl::sycl::invalid_parameter_error, 254
- cl::sycl::item, 277
 - dimensionality, 283
 - display, 280
 - get, 280, 281
 - get_linear_id, 281
 - get_offset, 281
 - get_range, 282
 - global_index, 283
 - global_range, 283
 - item, 279
 - offset, 283
 - operator[], 282
 - set, 283
- cl::sycl::kernel, 180
 - get, 183
 - handler, 183
 - implementation_t, 181
 - kernel, 182
- cl::sycl::kernel_error, 249
- cl::sycl::link_program_error, 257
- cl::sycl::memory_allocation_error, 259
- cl::sycl::nd_item, 283
 - barrier, 287
 - dimensionality, 299
 - get_global, 287, 288
 - get_global_linear_id, 289
 - get_global_range, 289
 - get_group, 290, 291
 - get_group_linear_id, 292
 - get_item, 292
 - get_local, 293
 - get_local_linear_id, 294
 - get_local_range, 295
 - get_nd_range, 296
 - get_num_groups, 296, 297
 - get_offset, 298
 - global_index, 299
 - local_index, 299
 - ND_range, 299
 - nd_item, 285, 286
 - set_global, 298
 - set_local, 299
- cl::sycl::nd_range, 299
 - dimensionality, 303
 - display, 301
 - get_global, 301
 - get_group, 302
 - get_local, 302
 - get_offset, 302
 - global_range, 303
 - local_range, 303
 - nd_range, 301
 - offset, 303
- cl::sycl::nd_range_error, 251
- cl::sycl::non_cl_error, 264
- cl::sycl::pipe, 93
 - capacity, 95
 - get_access, 95
 - implementation_t, 94
 - pipe, 95
 - value_type, 94
- cl::sycl::pipe_error, 260
- cl::sycl::pipe_reservation, 105
 - accessor_detail, 107
 - accessor_type, 107
 - begin, 110
 - blocking, 115
 - cbegin, 110
 - cend, 111
 - commit, 111
 - const_iterator, 108
 - const_pointer, 108
 - const_reference, 108
 - const_reverse_iterator, 108
 - crbegin, 112
 - crend, 112
 - difference_type, 108
 - end, 112
 - implementation, 115
 - iterator, 108
 - operator bool, 113
 - operator[], 113
 - pipe_reservation, 109
 - pointer, 108
 - rbegin, 113
 - reference, 108
 - rend, 114
 - reverse_iterator, 108
 - size, 114
 - size_type, 108
 - value_type, 109
- cl::sycl::platform, 192
 - get, 196
 - get_info, 196
 - get_platforms, 196
 - has_extension, 197
 - implementation_t, 194
 - is_host, 197
 - platform, 194, 195
- cl::sycl::platform_error, 261
- cl::sycl::profiling_error, 262
- cl::sycl::queue, 197
 - get, 203
 - get_context, 203
 - get_device, 204
 - get_info, 204
 - implementation_t, 199
 - is_host, 204
 - queue, 199–203
 - submit, 205
 - throw_asynchronous, 205

- wait, 206
- wait_and_throw, 206
- cl::sycl::range, 305
 - get_count, 307
- cl::sycl::runtime_error, 248
- cl::sycl::static_pipe, 115
 - capacity, 117
 - get_access, 117
 - implementation_t, 117
 - static_pipe, 117
 - value_type, 117
- cl::sycl::trisycl, 337
- cl::sycl::trisycl::default_error_handler, 362
 - report_error, 363
- cl::sycl::vec, 317
 - basic_type, 319
 - flatten, 320
 - flatten_to_tuple, 320
 - vec, 319
- cl_buf
 - cl::sycl::detail::accessor, 60
- cl_code
 - cl::sycl::cl_exception, 247
- cl_exception
 - cl::sycl::cl_exception, 247
- commit
 - cl::sycl::detail::pipe_reservation, 102
 - cl::sycl::pipe_reservation, 111
- const_iterator
 - cl::sycl::detail::accessor, 47
 - cl::sycl::detail::pipe_reservation, 98
 - cl::sycl::pipe_reservation, 108
- const_pointer
 - cl::sycl::pipe_reservation, 108
- const_reference
 - cl::sycl::accessor, 30
 - cl::sycl::buffer, 341
 - cl::sycl::detail::accessor, 47
 - cl::sycl::detail::pipe_accessor, 86
 - cl::sycl::pipe_reservation, 108
- const_reverse_iterator
 - cl::sycl::detail::accessor, 47
 - cl::sycl::pipe_reservation, 108
- constant
 - Dealing with OpenCL address spaces, 140
- constant_address_space
 - Dealing with OpenCL address spaces, 142
- constant_buffer
 - cl::sycl::access, 330
- context
 - cl::sycl::context, 148–150
 - cl::sycl::info, 336, 337
- copy_back_cl_buffer
 - cl::sycl::detail::accessor, 50
- copy_in_cl_buffer
 - cl::sycl::detail::accessor, 51
- correctly_rounded_divide_sqrt
 - Platforms, contexts, devices and queues, 213
- cpu
 - Platforms, contexts, devices and queues, 212
- cpu_selector
 - Platforms, contexts, devices and queues, 207
- crbegin
 - cl::sycl::accessor, 34
 - cl::sycl::detail::accessor, 52
 - cl::sycl::pipe_reservation, 112
- crend
 - cl::sycl::accessor, 34
 - cl::sycl::detail::accessor, 52
 - cl::sycl::pipe_reservation, 112
- custom
 - Platforms, contexts, devices and queues, 212
- d
 - cl::sycl::detail::opencl_device, 381
- Data access and storage in SYCL, 27
 - buffer_add_to_task, 118
 - buffer_allocator, 118
 - get_pipe_detail, 119
 - waiter, 119
- Dealing with OpenCL address spaces, 121
 - addr_space, 140
 - address_space, 142
 - constant, 140
 - constant_address_space, 142
 - generic, 140
 - generic_address_space, 142
 - global, 141
 - global_address_space, 142
 - local, 141
 - local_address_space, 142
 - make_multi, 142
 - multi_ptr, 141
 - priv, 142
 - private_address_space, 142
- debug.hpp
 - TRISYCL_DUMP_T, 478
 - TRISYCL_DUMP, 478
 - TRISYCL_INTERNAL_DUMP, 478
- debug_mode
 - cl::sycl::detail::pipe, 83
- Debugging and tracing support, 221
 - trace_kernel, 222
- default_device
 - cl::sycl::device_type_selector, 165
- default_handler
 - cl::sycl::error_handler, 242
- default_selector
 - Platforms, contexts, devices and queues, 207
- defaults
 - Platforms, contexts, devices and queues, 212
- denorm
 - Platforms, contexts, devices and queues, 213
- detail::accessor
 - cl::sycl::detail::buffer, 68
- device
 - cl::sycl::device, 157

- cl::sycl::info, [337](#)
 - Platforms, contexts, devices and queues, [208](#)
- device_affinity_domain
 - Platforms, contexts, devices and queues, [210](#)
- device_exec_capabilities
 - Platforms, contexts, devices and queues, [207](#)
- device_execution_capabilities
 - Platforms, contexts, devices and queues, [211](#)
- device_fp_config
 - Platforms, contexts, devices and queues, [207](#)
- device_partition_property
 - Platforms, contexts, devices and queues, [211](#)
- device_partition_type
 - Platforms, contexts, devices and queues, [211](#)
- device_queue_properties
 - Platforms, contexts, devices and queues, [207](#)
- device_type
 - cl::sycl::device_type_selector, [165](#)
 - Platforms, contexts, devices and queues, [208](#), [212](#)
- device_type_selector
 - cl::sycl::device_type_selector, [164](#)
- device_type_name_selector
 - cl::sycl::device_type_name_selector, [167](#)
- device_version
 - Platforms, contexts, devices and queues, [209](#)
- difference_type
 - cl::sycl::pipe_reservation, [108](#)
- dimension
 - cl::sycl::detail::small_array, [230](#)
- dimensionality
 - cl::sycl::accessor, [39](#)
 - cl::sycl::detail::accessor, [60](#)
 - cl::sycl::detail::small_array, [230](#)
 - cl::sycl::group, [276](#)
 - cl::sycl::item, [283](#)
 - cl::sycl::nd_item, [299](#)
 - cl::sycl::nd_range, [303](#)
- Dimensions
 - cl::sycl::handler, [178](#)
- discard_read_write
 - cl::sycl::access, [329](#)
- discard_write
 - cl::sycl::access, [329](#)
- dispatch_set_arg
 - cl::sycl::handler, [171](#)
- display
 - cl::sycl::detail::display_vector, [222](#)
 - cl::sycl::item, [280](#)
 - cl::sycl::nd_range, [301](#)
- double_fp_config
 - Platforms, contexts, devices and queues, [209](#)
- driver_version
 - Platforms, contexts, devices and queues, [209](#)
- element
 - cl::sycl::detail::accessor, [47](#)
 - cl::sycl::detail::buffer, [63](#)
- element_type
 - cl::sycl::detail::small_array, [228](#)
- empty
 - cl::sycl::detail::pipe, [75](#)
 - cl::sycl::detail::pipe_accessor, [87](#)
- empty_with_lock
 - cl::sycl::detail::pipe, [75](#)
- end
 - cl::sycl::accessor, [34](#)
 - cl::sycl::detail::accessor, [52](#)
 - cl::sycl::detail::pipe_reservation, [103](#)
 - cl::sycl::pipe_reservation, [112](#)
- endian_little
 - Platforms, contexts, devices and queues, [209](#)
- epilogues
 - cl::sycl::detail::task, [413](#)
- Error handling, [240](#)
 - async_handler, [265](#)
 - exception_ptr, [265](#)
- error_correction_support
 - Platforms, contexts, devices and queues, [209](#)
- event
 - cl::sycl::event, [363](#)
- exception
 - cl::sycl::exception, [245](#)
- exception_ptr
 - Error handling, [265](#)
- exec_kernel
 - Platforms, contexts, devices and queues, [211](#)
- exec_native_kernel
 - Platforms, contexts, devices and queues, [211](#)
- execution_capabilities
 - Platforms, contexts, devices and queues, [209](#)
- execution_ended
 - cl::sycl::detail::task, [413](#)
- expand
 - Helpers to do array and tuple conversion, [218](#)
- Expressing parallelism through kernels, [266](#)
 - make_id, [307](#), [308](#)
 - make_range, [308](#)
 - parallel_for, [309](#), [310](#)
 - parallel_for_global_offset, [312](#)
 - parallel_for_work_item, [312](#)
 - parallel_for_workgroup, [313](#)
 - parallel_for_workitem, [314](#)
- extensions
 - Platforms, contexts, devices and queues, [209](#)
- fence_space
 - cl::sycl::access, [329](#)
- fill_tuple
 - Helpers to do array and tuple conversion, [219](#)
- final_data
 - cl::sycl::detail::buffer, [69](#)
- finished
 - cl::sycl::detail::queue, [398](#)
- finished_mutex
 - cl::sycl::detail::queue, [398](#)
- flatten
 - cl::sycl::vec, [320](#)
- flatten_to_tuple

- cl::sycl::vec, [320](#)
- fma
 - Platforms, contexts, devices and queues, [213](#)
- fp_config
 - Platforms, contexts, devices and queues, [212](#)
- full
 - cl::sycl::detail::pipe, [76](#)
 - cl::sycl::detail::pipe_accessor, [87](#)
- full_with_lock
 - cl::sycl::detail::pipe, [76](#)
- function_class
 - cl::sycl, [327](#)
- generic
 - Dealing with OpenCL address spaces, [140](#)
- generic_address_space
 - Dealing with OpenCL address spaces, [142](#)
- get
 - cl::sycl::context, [151](#)
 - cl::sycl::detail::device, [154](#)
 - cl::sycl::detail::host_device, [371](#)
 - cl::sycl::detail::host_platform, [184](#)
 - cl::sycl::detail::host_queue, [374](#)
 - cl::sycl::detail::kernel, [180](#)
 - cl::sycl::detail::opencl_device, [378](#)
 - cl::sycl::detail::opencl_kernel, [383](#)
 - cl::sycl::detail::opencl_platform, [188](#)
 - cl::sycl::detail::opencl_queue, [389](#)
 - cl::sycl::detail::platform, [192](#)
 - cl::sycl::detail::queue, [394](#)
 - cl::sycl::detail::small_array, [230](#)
 - cl::sycl::device, [158](#)
 - cl::sycl::group, [270](#)
 - cl::sycl::item, [280](#), [281](#)
 - cl::sycl::kernel, [183](#)
 - cl::sycl::platform, [196](#)
 - cl::sycl::queue, [203](#)
- get_access
 - cl::sycl::buffer, [349](#), [350](#)
 - cl::sycl::detail::buffer, [65](#)
 - cl::sycl::pipe, [95](#)
 - cl::sycl::static_pipe, [117](#)
- get_address
 - cl::sycl::detail::address_space_variable, [139](#)
- get_boost_compute
 - cl::sycl::detail::host_queue, [374](#)
 - cl::sycl::detail::kernel, [180](#)
 - cl::sycl::detail::opencl_kernel, [383](#)
 - cl::sycl::detail::opencl_queue, [389](#)
 - cl::sycl::detail::queue, [394](#)
- get_buffer
 - cl::sycl::detail::accessor, [52](#)
- get_cl_buffer
 - cl::sycl::detail::accessor, [53](#)
- get_cl_code
 - cl::sycl::cl_exception, [247](#)
- get_context
 - cl::sycl::detail::host_queue, [375](#)
 - cl::sycl::detail::opencl_queue, [389](#)
 - cl::sycl::detail::queue, [395](#)
 - cl::sycl::queue, [203](#)
- get_count
 - cl::sycl::buffer, [350](#)
 - cl::sycl::detail::buffer, [66](#)
 - cl::sycl::range, [307](#)
- get_destructor_future
 - cl::sycl::detail::buffer, [66](#)
- get_device
 - cl::sycl::detail::host_queue, [375](#)
 - cl::sycl::detail::opencl_queue, [389](#)
 - cl::sycl::detail::queue, [395](#)
 - cl::sycl::queue, [204](#)
- get_devices
 - cl::sycl::context, [151](#)
 - Platforms, contexts, devices and queues, [215](#)
- get_global
 - cl::sycl::nd_item, [287](#), [288](#)
 - cl::sycl::nd_range, [301](#)
- get_global_linear_id
 - cl::sycl::nd_item, [289](#)
- get_global_range
 - cl::sycl::group, [270](#)
 - cl::sycl::nd_item, [289](#)
- get_group
 - cl::sycl::nd_item, [290](#), [291](#)
 - cl::sycl::nd_range, [302](#)
- get_group_linear_id
 - cl::sycl::nd_item, [292](#)
- get_group_range
 - cl::sycl::group, [271](#)
- get_info
 - cl::sycl::context, [151](#)
 - cl::sycl::device, [158](#), [159](#)
 - cl::sycl::platform, [196](#)
 - cl::sycl::queue, [204](#)
- get_info_string
 - cl::sycl::detail::host_platform, [184](#)
 - cl::sycl::detail::opencl_platform, [188](#)
 - cl::sycl::detail::platform, [192](#)
- get_item
 - cl::sycl::nd_item, [292](#)
- get_kernel
 - cl::sycl::detail::task, [408](#)
- get_latest_producer
 - cl::sycl::detail::buffer_base, [355](#)
- get_linear
 - cl::sycl::group, [271](#)
- get_linear_id
 - cl::sycl::item, [281](#)
- get_local
 - cl::sycl::nd_item, [293](#)
 - cl::sycl::nd_range, [302](#)
- get_local_linear_id
 - cl::sycl::nd_item, [294](#)
- get_local_range
 - cl::sycl::group, [271](#), [272](#)
 - cl::sycl::nd_item, [295](#)

- get_nd_range
 - cl::sycl::group, 272
 - cl::sycl::nd_item, 296
- get_num_groups
 - cl::sycl::nd_item, 296, 297
- get_offset
 - cl::sycl::group, 273
 - cl::sycl::item, 281
 - cl::sycl::nd_item, 298
 - cl::sycl::nd_range, 302
- get_or_register
 - cl::sycl::detail::cache, 359
- get_pipe_detail
 - cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >, 43
 - cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >, 41
 - cl::sycl::detail::pipe_accessor, 88
 - Data access and storage in SYCL, 119
- get_platform
 - cl::sycl::context, 152
 - cl::sycl::detail::device, 154
 - cl::sycl::detail::host_device, 371
 - cl::sycl::detail::opencl_device, 378
 - cl::sycl::device, 159
- get_platforms
 - cl::sycl::platform, 196
- get_queue
 - cl::sycl::detail::task, 408
- get_range
 - cl::sycl::buffer, 350
 - cl::sycl::detail::buffer, 67
 - cl::sycl::item, 282
- get_size
 - cl::sycl::buffer, 351
 - cl::sycl::detail::accessor, 53
 - cl::sycl::detail::buffer, 67
- gl_context_interop
 - cl::sycl::info, 336
- gl_interop
 - cl::sycl::info, 336
- global
 - Dealing with OpenCL address spaces, 141
 - Platforms, contexts, devices and queues, 214
- global_address_space
 - Dealing with OpenCL address spaces, 142
- global_and_local
 - cl::sycl::access, 329
- global_buffer
 - cl::sycl::access, 330
- global_index
 - cl::sycl::item, 283
 - cl::sycl::nd_item, 299
- global_mem_cache_line_size
 - Platforms, contexts, devices and queues, 209
- global_mem_cache_size
 - Platforms, contexts, devices and queues, 209
- global_mem_cache_type
 - Platforms, contexts, devices and queues, 209, 213
- global_mem_size
 - Platforms, contexts, devices and queues, 209
- global_range
 - cl::sycl::item, 283
 - cl::sycl::nd_range, 303
- global_space
 - cl::sycl::access, 329
- gpu
 - Platforms, contexts, devices and queues, 212
- gpu_selector
 - Platforms, contexts, devices and queues, 207
- group
 - cl::sycl::group, 269
- group_id
 - cl::sycl::group, 276
- handler
 - cl::sycl::detail::accessor, 60
 - cl::sycl::handler, 171
 - cl::sycl::kernel, 183
- handler.hpp
 - TRISYCL_ParallelForFunctor_GLOBAL_OFFSET, 539
 - TRISYCL_ParallelForKernel_RANGE, 539
 - TRISYCL_parallel_for_functor_GLOBAL, 539
- handler_event, 363
- has_extension
 - cl::sycl::detail::device, 154
 - cl::sycl::detail::host_device, 371
 - cl::sycl::detail::host_platform, 185
 - cl::sycl::detail::opencl_device, 378
 - cl::sycl::detail::opencl_platform, 189
 - cl::sycl::detail::platform, 192
 - cl::sycl::device, 159
 - cl::sycl::platform, 197
- hash
 - cl::sycl::detail::shared_ptr_implementation, 401
- Helpers to do array and tuple conversion, 217
 - expand, 218
 - fill_tuple, 219
 - tuple_to_array, 220
 - tuple_to_array_iterate, 220
- host
 - Platforms, contexts, devices and queues, 212
- host_buffer
 - cl::sycl::access, 330
- host_image
 - cl::sycl::access, 330
- host_selector
 - Platforms, contexts, devices and queues, 207
- host_unified_memory
 - Platforms, contexts, devices and queues, 209
- host_write_back
 - cl::sycl::detail::buffer, 69
- id
 - cl::sycl::id, 277
- image

- cl::sycl::access, 330
- image2d_max_height
 - Platforms, contexts, devices and queues, 209
- image2d_max_width
 - Platforms, contexts, devices and queues, 209
- image3d_max_depth
 - Platforms, contexts, devices and queues, 209
- image3d_max_height
 - Platforms, contexts, devices and queues, 209
- image3d_max_width
 - Platforms, contexts, devices and queues, 209
- image_array
 - cl::sycl::access, 330
- image_max_array_size
 - Platforms, contexts, devices and queues, 209
- image_max_buffer_size
 - Platforms, contexts, devices and queues, 209
- image_support
 - Platforms, contexts, devices and queues, 208
- implementation
 - cl::sycl::detail::pipe_accessor, 92
 - cl::sycl::detail::shared_ptr_implementation, 403
 - cl::sycl::pipe_reservation, 115
- implementation_t
 - cl::sycl::accessor, 30
 - cl::sycl::buffer, 341
 - cl::sycl::detail::buffer_waiter, 70
 - cl::sycl::detail::pipe, 74
 - cl::sycl::device, 156
 - cl::sycl::kernel, 181
 - cl::sycl::pipe, 94
 - cl::sycl::platform, 194
 - cl::sycl::queue, 199
 - cl::sycl::static_pipe, 117
- include/CL/sycl.hpp, 415, 416
- include/CL/sycl/access.hpp, 417, 418
- include/CL/sycl/accessor.hpp, 419, 420
- include/CL/sycl/address_space.hpp, 438, 440
- include/CL/sycl/address_space/detail/address_↔
space.hpp, 431, 433
- include/CL/sycl/buffer.hpp, 446, 448
- include/CL/sycl/buffer/detail/accessor.hpp, 425, 426
- include/CL/sycl/buffer/detail/buffer.hpp, 442, 443
- include/CL/sycl/buffer/detail/buffer_base.hpp, 454, 455
- include/CL/sycl/buffer/detail/buffer_waiter.hpp, 457, 459
- include/CL/sycl/buffer_allocator.hpp, 460, 461
- include/CL/sycl/command_group/detail/task.hpp, 462,
463
- include/CL/sycl/context.hpp, 466, 468
- include/CL/sycl/detail/array_tuple_helpers.hpp, 470,
472
- include/CL/sycl/detail/cache.hpp, 474, 475
- include/CL/sycl/detail/debug.hpp, 477, 479
- include/CL/sycl/detail/default_classes.hpp, 481, 483
- include/CL/sycl/detail/global_config.hpp, 484, 485
- include/CL/sycl/detail/linear_id.hpp, 486, 488
- include/CL/sycl/detail/shared_ptr_implementation.hpp,
488, 490
- include/CL/sycl/detail/singleton.hpp, 491, 492
- include/CL/sycl/detail/small_array.hpp, 493, 495
- include/CL/sycl/detail/unimplemented.hpp, 498, 499
- include/CL/sycl/device.hpp, 502, 504
- include/CL/sycl/device/detail/device.hpp, 500, 501
- include/CL/sycl/device/detail/device_tail.hpp, 512, 513
- include/CL/sycl/device/detail/host_device.hpp, 513, 515
- include/CL/sycl/device/detail/ocl_device.hpp, 516,
518
- include/CL/sycl/device_selector.hpp, 519, 521
- include/CL/sycl/device_selector/detail/device_selector_↔
tail.hpp, 522, 523
- include/CL/sycl/error_handler.hpp, 524, 526
- include/CL/sycl/event.hpp, 527
- include/CL/sycl/exception.hpp, 528, 530
- include/CL/sycl/group.hpp, 533, 535
- include/CL/sycl/handler.hpp, 537, 540
- include/CL/sycl/handler_event.hpp, 545, 546
- include/CL/sycl/id.hpp, 546, 548
- include/CL/sycl/image.hpp, 549, 550
- include/CL/sycl/info/device.hpp, 507, 510
- include/CL/sycl/info/param_traits.hpp, 551, 552
- include/CL/sycl/info/platform.hpp, 553, 555
- include/CL/sycl/item.hpp, 562, 564
- include/CL/sycl/kernel.hpp, 568, 569
- include/CL/sycl/kernel/detail/kernel.hpp, 565, 567
- include/CL/sycl/kernel/detail/ocl_kernel.hpp, 571,
573
- include/CL/sycl/nd_item.hpp, 575, 576
- include/CL/sycl/nd_range.hpp, 579, 580
- include/CL/sycl/parallelism.hpp, 588, 590
- include/CL/sycl/parallelism/detail/parallelism.hpp, 581,
584
- include/CL/sycl/pipe.hpp, 597, 598
- include/CL/sycl/pipe/detail/pipe.hpp, 590, 592
- include/CL/sycl/pipe/detail/pipe_accessor.hpp, 600, 601
- include/CL/sycl/pipe_reservation.hpp, 608, 609
- include/CL/sycl/pipe_reservation/detail/pipe_reservation_↔
hpp, 604, 606
- include/CL/sycl/platform.hpp, 558, 560
- include/CL/sycl/platform/detail/host_platform.hpp, 611,
613
- include/CL/sycl/platform/detail/ocl_platform.hpp,
614, 616
- include/CL/sycl/platform/detail/platform.hpp, 556, 557
- include/CL/sycl/queue.hpp, 625, 627
- include/CL/sycl/queue/detail/host_queue.hpp, 618, 619
- include/CL/sycl/queue/detail/ocl_queue.hpp, 620,
621
- include/CL/sycl/queue/detail/queue.hpp, 622, 623
- include/CL/sycl/range.hpp, 631, 633
- include/CL/sycl/static_pipe.hpp, 634, 635
- include/CL/sycl/vec.hpp, 636, 638
- inf_nan
 - Platforms, contexts, devices and queues, 213
- instance
 - cl::sycl::detail::ocl_device, 379
 - cl::sycl::detail::ocl_kernel, 384

- cl::sycl::detail::opencl_platform, 189
 - cl::sycl::detail::opencl_queue, 390
 - cl::sycl::detail::singleton, 403
- is_accelerator
 - cl::sycl::detail::device, 154
 - cl::sycl::detail::host_device, 372
 - cl::sycl::detail::opencl_device, 380
 - cl::sycl::device, 160
- is_available
 - Platforms, contexts, devices and queues, 209
- is_compiler_available
 - Platforms, contexts, devices and queues, 209
- is_cpu
 - cl::sycl::detail::device, 154
 - cl::sycl::detail::host_device, 372
 - cl::sycl::detail::opencl_device, 380
 - cl::sycl::device, 160
- is_gpu
 - cl::sycl::detail::device, 154
 - cl::sycl::detail::host_device, 372
 - cl::sycl::detail::opencl_device, 380
 - cl::sycl::device, 160
- is_host
 - cl::sycl::context, 152
 - cl::sycl::detail::device, 154
 - cl::sycl::detail::host_device, 373
 - cl::sycl::detail::host_platform, 185
 - cl::sycl::detail::host_queue, 375
 - cl::sycl::detail::opencl_device, 380
 - cl::sycl::detail::opencl_platform, 190
 - cl::sycl::detail::opencl_queue, 390
 - cl::sycl::detail::platform, 192
 - cl::sycl::detail::queue, 396
 - cl::sycl::device, 161
 - cl::sycl::platform, 197
 - cl::sycl::queue, 204
- is_linker_available
 - Platforms, contexts, devices and queues, 209
- is_read_access
 - cl::sycl::detail::accessor, 54
- is_read_only
 - cl::sycl::buffer, 351
- is_write_access
 - cl::sycl::detail::accessor, 54
- item
 - cl::sycl::item, 279
- iterator
 - cl::sycl::detail::accessor, 47
 - cl::sycl::detail::pipe_reservation, 98
 - cl::sycl::pipe_reservation, 108
- k
 - cl::sycl::detail::opencl_kernel, 385
- kernel
 - cl::sycl::detail::task, 413
 - cl::sycl::kernel, 182
- kernel/detail/kernel.hpp
 - TRISYCL_ParallelForKernel_RANGE, 567
- kernel_end
 - cl::sycl::detail::queue, 396
- kernel_start
 - cl::sycl::detail::queue, 397
- key_type
 - cl::sycl::detail::cache, 359
- L1_cache
 - Platforms, contexts, devices and queues, 212
- L2_cache
 - Platforms, contexts, devices and queues, 211, 212
- L3_cache
 - Platforms, contexts, devices and queues, 211, 212
- L4_cache
 - Platforms, contexts, devices and queues, 211, 212
- latest_producer
 - cl::sycl::detail::buffer_base, 357
- latest_producer_mutex
 - cl::sycl::detail::buffer_base, 357
- linear_id
 - Some helpers for the implementation, 237
- local
 - cl::sycl::access, 330
 - Dealing with OpenCL address spaces, 141
 - Platforms, contexts, devices and queues, 214
- local_address_space
 - Dealing with OpenCL address spaces, 142
- local_index
 - cl::sycl::nd_item, 299
- local_mem_size
 - Platforms, contexts, devices and queues, 209
- local_mem_type
 - Platforms, contexts, devices and queues, 209, 213
- local_range
 - cl::sycl::nd_range, 303
- local_space
 - cl::sycl::access, 329
- m
 - cl::sycl::detail::cache, 361
- make_id
 - Expressing parallelism through kernels, 307, 308
- make_multi
 - Dealing with OpenCL address spaces, 142
- make_range
 - Expressing parallelism through kernels, 308
- Manage default configuration and types, 224
 - __SYCL_SINGLE_SOURCE__, 224
 - CL_SYCL_LANGUAGE_VERSION, 224
 - CL_TRISYCL_LANGUAGE_VERSION, 224
 - TRISYCL_ASYNC, 225
 - TRISYCL_SKIP_OPENCL, 225
- max_clock_frequency
 - Platforms, contexts, devices and queues, 208
- max_compute_units
 - Platforms, contexts, devices and queues, 208
- max_constant_args
 - Platforms, contexts, devices and queues, 209
- max_constant_buffer_size
 - Platforms, contexts, devices and queues, 209

- max_mem_alloc_size
 - Platforms, contexts, devices and queues, [208](#)
- max_parameter_size
 - Platforms, contexts, devices and queues, [209](#)
- max_read_image_args
 - Platforms, contexts, devices and queues, [208](#)
- max_samplers
 - Platforms, contexts, devices and queues, [209](#)
- max_work_group_size
 - Platforms, contexts, devices and queues, [208](#)
- max_work_item_dimensions
 - Platforms, contexts, devices and queues, [208](#)
- max_work_item_sizes
 - Platforms, contexts, devices and queues, [208](#)
- max_write_image_args
 - Platforms, contexts, devices and queues, [208](#)
- mem_base_addr_align
 - Platforms, contexts, devices and queues, [209](#)
- message
 - cl::sycl::exception, [246](#)
- mode
 - cl::sycl::access, [329](#)
 - cl::sycl::detail::pipe_accessor, [92](#)
 - cl::sycl::detail::pipe_reservation, [105](#)
- move_read_reservation_forward
 - cl::sycl::detail::pipe, [76](#)
- move_write_reservation_forward
 - cl::sycl::detail::pipe, [77](#)
- multi_ptr
 - Dealing with OpenCL address spaces, [141](#)
- mutex_class
 - cl::sycl, [327](#)
- ND_range
 - cl::sycl::nd_item, [299](#)
- name
 - Platforms, contexts, devices and queues, [209](#)
- native_vector_width_char
 - Platforms, contexts, devices and queues, [208](#)
- native_vector_width_double
 - Platforms, contexts, devices and queues, [208](#)
- native_vector_width_float
 - Platforms, contexts, devices and queues, [208](#)
- native_vector_width_half
 - Platforms, contexts, devices and queues, [208](#)
- native_vector_width_int
 - Platforms, contexts, devices and queues, [208](#)
- native_vector_width_long_long
 - Platforms, contexts, devices and queues, [208](#)
- native_vector_width_short
 - Platforms, contexts, devices and queues, [208](#)
- nd_item
 - cl::sycl::nd_item, [285](#), [286](#)
- nd_range
 - cl::sycl::nd_range, [301](#)
- ndr
 - cl::sycl::group, [276](#)
- next_partitionable
 - Platforms, contexts, devices and queues, [211](#)
- no_partition
 - Platforms, contexts, devices and queues, [212](#)
- none
 - Platforms, contexts, devices and queues, [213](#), [214](#)
- notify_buffer_destructor
 - cl::sycl::detail::buffer_base, [357](#)
- notify_consumers
 - cl::sycl::detail::task, [408](#)
- num_devices
 - cl::sycl::info, [336](#)
- numa
 - Platforms, contexts, devices and queues, [211](#), [212](#)
- number_of_users
 - cl::sycl::detail::buffer_base, [357](#)
- offset
 - cl::sycl::item, [283](#)
 - cl::sycl::nd_range, [303](#)
- ok
 - cl::sycl::detail::pipe_accessor, [93](#)
 - cl::sycl::detail::pipe_reservation, [105](#)
- opengl
 - Platforms, contexts, devices and queues, [212](#)
- opengl_device
 - cl::sycl::detail::opengl_device, [377](#)
- opengl_kernel
 - cl::sycl::detail::opengl_kernel, [383](#)
- opengl_kernel.hpp
 - TRISYCL_ParallelForKernel_RANGE, [573](#)
- opengl_platform
 - cl::sycl::detail::opengl_platform, [188](#)
- opengl_queue
 - cl::sycl::detail::opengl_queue, [388](#)
- opengl_type
 - cl::sycl::detail::address_space_base, [136](#)
 - cl::sycl::detail::address_space_object, [132](#)
 - cl::sycl::detail::address_space_variable, [139](#)
- opengl_version
 - Platforms, contexts, devices and queues, [209](#)
- operator BasicType
 - cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >, [233](#)
- operator bool
 - cl::sycl::detail::pipe_accessor, [88](#)
 - cl::sycl::detail::pipe_reservation, [103](#)
 - cl::sycl::pipe_reservation, [113](#)
- operator FinalType
 - cl::sycl::detail::small_array, [230](#)
- operator opengl_type &
 - cl::sycl::detail::address_space_object, [132](#)
 - cl::sycl::detail::address_space_variable, [139](#)
- operator<
 - cl::sycl::detail::shared_ptr_implementation, [402](#)
- operator<<
 - cl::sycl::detail::pipe_accessor, [88](#)
- operator>>
 - cl::sycl::detail::pipe_accessor, [89](#)
- operator*
 - cl::sycl::accessor, [34](#), [35](#)

- cl::sycl::detail::accessor, 55
- operator()
 - cl::sycl::device_selector, 169
 - cl::sycl::device_type_selector, 165
 - std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >, 364
 - std::hash< cl::sycl::device >, 365
 - std::hash< cl::sycl::kernel >, 366
 - std::hash< cl::sycl::platform >, 367
 - std::hash< cl::sycl::queue >, 368
- operator==
 - cl::sycl::detail::shared_ptr_implementation, 402
- operator[]
 - cl::sycl::accessor, 35–38
 - cl::sycl::detail::accessor, 56–58
 - cl::sycl::detail::pipe_reservation, 103
 - cl::sycl::group, 274
 - cl::sycl::item, 282
 - cl::sycl::pipe_reservation, 113
- owner_queue
 - cl::sycl::detail::task, 414
- P
 - cl::sycl::detail::opencl_platform, 190
 - cl::sycl::detail::pipe_reservation, 105
- parallel_OpenMP_for_iterate
 - cl::sycl::detail::parallel_OpenMP_for_iterate, 304
- parallel_for
 - cl::sycl::handler, 172
 - Expressing parallelism through kernels, 309, 310
- parallel_for_global_offset
 - Expressing parallelism through kernels, 312
- parallel_for_iterate
 - cl::sycl::detail::parallel_for_iterate, 304
 - cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >, 305
- parallel_for_work_group
 - cl::sycl::handler, 172
- parallel_for_work_item
 - cl::sycl::group, 274, 275
 - Expressing parallelism through kernels, 312
- parallel_for_workgroup
 - Expressing parallelism through kernels, 313
- parallel_for_workitem
 - Expressing parallelism through kernels, 314
- param_traits.hpp
 - TRISYCL_INFO_PARAM_TRAITS_ANY_T, 552
 - TRISYCL_INFO_PARAM_TRAITS, 552
- parent_device
 - Platforms, contexts, devices and queues, 209
- partition_affinity_domain
 - Platforms, contexts, devices and queues, 209
- partition_affinity_domain_next_partitionable
 - Platforms, contexts, devices and queues, 211
- partition_by_affinity_domain
 - Platforms, contexts, devices and queues, 211
- partition_by_counts
 - Platforms, contexts, devices and queues, 211
- partition_equally
 - Platforms, contexts, devices and queues, 211
- partition_max_sub_devices
 - Platforms, contexts, devices and queues, 209
- partition_properties
 - Platforms, contexts, devices and queues, 209
- partition_type
 - Platforms, contexts, devices and queues, 209
- pipe
 - cl::sycl::access, 330
 - cl::sycl::detail::pipe, 75
 - cl::sycl::pipe, 95
- pipe_accessor
 - cl::sycl::detail::pipe_accessor, 86
- pipe_reservation
 - cl::sycl::detail::pipe_reservation, 98, 99
 - cl::sycl::pipe_reservation, 109
- platform
 - cl::sycl::platform, 194, 195
 - Platforms, contexts, devices and queues, 209, 214
- platform_extensions
 - cl::sycl::detail::host_platform, 186
- Platforms, contexts, devices and queues, 144
 - __attribute__, 215, 216
 - accelerator, 212
 - address_bits, 208
 - all, 212
 - built_in_kernels, 209
 - correctly_rounded_divide_sqrt, 213
 - cpu, 212
 - cpu_selector, 207
 - custom, 212
 - default_selector, 207
 - defaults, 212
 - denorm, 213
 - device, 208
 - device_affinity_domain, 210
 - device_exec_capabilities, 207
 - device_execution_capabilities, 211
 - device_fp_config, 207
 - device_partition_property, 211
 - device_partition_type, 211
 - device_queue_properties, 207
 - device_type, 208, 212
 - device_version, 209
 - double_fp_config, 209
 - driver_version, 209
 - endian_little, 209
 - error_correction_support, 209
 - exec_kernel, 211
 - exec_native_kernel, 211
 - execution_capabilities, 209
 - extensions, 209
 - fma, 213
 - fp_config, 212
 - get_devices, 215
 - global, 214
 - global_mem_cache_line_size, 209
 - global_mem_cache_size, 209

- global_mem_cache_type, 209, 213
- global_mem_size, 209
- gpu, 212
- gpu_selector, 207
- host, 212
- host_selector, 207
- host_unified_memory, 209
- image2d_max_height, 209
- image2d_max_width, 209
- image3d_max_depth, 209
- image3d_max_height, 209
- image3d_max_width, 209
- image_max_array_size, 209
- image_max_buffer_size, 209
- image_support, 208
- inf_nan, 213
- is_available, 209
- is_compiler_available, 209
- is_linker_available, 209
- L1_cache, 212
- L2_cache, 211, 212
- L3_cache, 211, 212
- L4_cache, 211, 212
- local, 214
- local_mem_size, 209
- local_mem_type, 209, 213
- max_clock_frequency, 208
- max_compute_units, 208
- max_constant_args, 209
- max_constant_buffer_size, 209
- max_mem_alloc_size, 208
- max_parameter_size, 209
- max_read_image_args, 208
- max_samplers, 209
- max_work_group_size, 208
- max_work_item_dimensions, 208
- max_work_item_sizes, 208
- max_write_image_args, 208
- mem_base_addr_align, 209
- name, 209
- native_vector_width_char, 208
- native_vector_width_double, 208
- native_vector_width_float, 208
- native_vector_width_half, 208
- native_vector_width_int, 208
- native_vector_width_long_long, 208
- native_vector_width_short, 208
- next_partitionable, 211
- no_partition, 212
- none, 213, 214
- numa, 211, 212
- opencl, 212
- opencl_version, 209
- parent_device, 209
- partition_affinity_domain, 209
- partition_affinity_domain_next_partitionable, 211
- partition_by_affinity_domain, 211
- partition_by_counts, 211
- partition_equally, 211
- partition_max_sub_devices, 209
- partition_properties, 209
- partition_type, 209
- platform, 209, 214
- preferred_interop_user_sync, 209
- preferred_vector_width_char, 208
- preferred_vector_width_double, 208
- preferred_vector_width_float, 208
- preferred_vector_width_half, 208
- preferred_vector_width_int, 208
- preferred_vector_width_long_long, 208
- preferred_vector_width_short, 208
- printf_buffer_size, 209
- profile, 209
- profiling_timer_resolution, 209
- queue_properties, 209
- read_only, 213
- reference_count, 210
- round_to_inf, 213
- round_to_nearest, 213
- round_to_zero, 213
- single_fp_config, 209
- soft_float, 213
- TRISYCL_SKIP_OPENCL, 214
- unsupported, 211
- vendor, 209
- vendor_id, 208
- write_only, 213
- pointer
 - cl::sycl::pipe_reservation, 108
- postlude
 - cl::sycl::detail::task, 409
- preferred_interop_user_sync
 - Platforms, contexts, devices and queues, 209
- preferred_vector_width_char
 - Platforms, contexts, devices and queues, 208
- preferred_vector_width_double
 - Platforms, contexts, devices and queues, 208
- preferred_vector_width_float
 - Platforms, contexts, devices and queues, 208
- preferred_vector_width_half
 - Platforms, contexts, devices and queues, 208
- preferred_vector_width_int
 - Platforms, contexts, devices and queues, 208
- preferred_vector_width_long_long
 - Platforms, contexts, devices and queues, 208
- preferred_vector_width_short
 - Platforms, contexts, devices and queues, 208
- prelude
 - cl::sycl::detail::task, 409
- printf_buffer_size
 - Platforms, contexts, devices and queues, 209
- priv
 - Dealing with OpenCL address spaces, 142
- private_address_space
 - Dealing with OpenCL address spaces, 142
- producer_tasks

- cl::sycl::detail::task, [414](#)
- profile
 - Platforms, contexts, devices and queues, [209](#)
- profiling_timer_resolution
 - Platforms, contexts, devices and queues, [209](#)
- prologues
 - cl::sycl::detail::task, [414](#)
- properties
 - cl::sycl::info, [337](#)
- q
 - cl::sycl::detail::opencl_queue, [391](#)
- queue
 - cl::sycl::detail::queue, [393](#)
 - cl::sycl::info, [336](#)
 - cl::sycl::queue, [199–203](#)
- queue_profiling
 - cl::sycl::info, [336](#)
- queue_properties
 - Platforms, contexts, devices and queues, [209](#)
- r_id_q
 - cl::sycl::detail::pipe, [83](#)
- rank
 - cl::sycl::detail::pipe_accessor, [93](#)
- rbegin
 - cl::sycl::accessor, [38](#)
 - cl::sycl::detail::accessor, [59](#)
 - cl::sycl::pipe_reservation, [113](#)
- read
 - cl::sycl::access, [329](#)
 - cl::sycl::detail::pipe, [77](#)
 - cl::sycl::detail::pipe_accessor, [89, 90](#)
- read_done
 - cl::sycl::detail::pipe, [83](#)
- read_only
 - cl::sycl::detail::buffer_base, [358](#)
 - Platforms, contexts, devices and queues, [213](#)
- read_reserved_frozen
 - cl::sycl::detail::pipe, [83](#)
- read_write
 - cl::sycl::access, [329](#)
- ready
 - cl::sycl::detail::buffer_base, [358](#)
 - cl::sycl::detail::reserve_id, [72](#)
 - cl::sycl::detail::task, [414](#)
- ready_mutex
 - cl::sycl::detail::buffer_base, [358](#)
 - cl::sycl::detail::task, [414](#)
- reference
 - cl::sycl::accessor, [30](#)
 - cl::sycl::buffer, [342](#)
 - cl::sycl::detail::accessor, [47](#)
 - cl::sycl::detail::pipe_accessor, [86](#)
 - cl::sycl::detail::pipe_reservation, [98](#)
 - cl::sycl::pipe_reservation, [108](#)
- reference_count
 - cl::sycl::info, [336, 337](#)
 - Platforms, contexts, devices and queues, [210](#)
- release
 - cl::sycl::detail::buffer_base, [356](#)
- release_buffers
 - cl::sycl::detail::task, [410](#)
- remove
 - cl::sycl::detail::cache, [360](#)
- rend
 - cl::sycl::accessor, [39](#)
 - cl::sycl::detail::accessor, [59](#)
 - cl::sycl::pipe_reservation, [114](#)
- report_error
 - cl::sycl::error_handler, [242](#)
 - cl::sycl::trisycl::default_error_handler, [363](#)
- reserve
 - cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >, [43](#)
 - cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >, [41](#)
 - cl::sycl::detail::pipe_accessor, [90](#)
- reserve_id
 - cl::sycl::detail::reserve_id, [71](#)
- reserve_read
 - cl::sycl::detail::pipe, [78](#)
- reserve_write
 - cl::sycl::detail::pipe, [79](#)
- reserved_for_reading
 - cl::sycl::detail::pipe, [80](#)
- reserved_for_writing
 - cl::sycl::detail::pipe, [80](#)
- reverse_iterator
 - cl::sycl::detail::accessor, [48](#)
 - cl::sycl::pipe_reservation, [108](#)
- rid
 - cl::sycl::detail::pipe_reservation, [105](#)
- rid_iterator
 - cl::sycl::detail::pipe, [74](#)
- round_to_inf
 - Platforms, contexts, devices and queues, [213](#)
- round_to_nearest
 - Platforms, contexts, devices and queues, [213](#)
- round_to_zero
 - Platforms, contexts, devices and queues, [213](#)
- running_kernels
 - cl::sycl::detail::queue, [398](#)
- schedule
 - cl::sycl::detail::task, [410](#)
- select_device
 - cl::sycl::device_selector, [169](#)
- set
 - cl::sycl::item, [283](#)
- set_arg
 - cl::sycl::handler, [174, 175](#)
- set_args
 - cl::sycl::handler, [175](#)
- set_debug
 - cl::sycl::detail::pipe_accessor, [91](#)
- set_final_data
 - cl::sycl::buffer, [351](#)

- cl::sycl::detail::buffer, 68
- set_global
 - cl::sycl::nd_item, 298
- set_kernel
 - cl::sycl::detail::task, 412
- set_latest_producer
 - cl::sycl::detail::buffer_base, 356
- set_local
 - cl::sycl::nd_item, 299
- shared_data
 - cl::sycl::detail::buffer, 69
- shared_ptr_class
 - cl::sycl, 327
- shared_ptr_implementation
 - cl::sycl::detail::shared_ptr_implementation, 400, 401
- single_fp_config
 - Platforms, contexts, devices and queues, 209
- single_task
 - cl::sycl::handler, 176
- size
 - cl::sycl::detail::pipe, 81
 - cl::sycl::detail::pipe_accessor, 91
 - cl::sycl::detail::pipe_reservation, 104
 - cl::sycl::detail::reserve_id, 72
 - cl::sycl::pipe_reservation, 114
- size_type
 - cl::sycl::pipe_reservation, 108
- size_with_lock
 - cl::sycl::detail::pipe, 81
- small_array
 - cl::sycl::detail::small_array, 228, 229
- small_array_123
 - cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >, 233
 - cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >, 234
 - cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >, 236
- soft_float
 - Platforms, contexts, devices and queues, 213
- Some helpers for the implementation, 226
 - linear_id, 237
 - TRISYCL_BOOST_OPERATOR_VECTOR_OP, 237
 - unimplemented, 238
- start
 - cl::sycl::detail::reserve_id, 72
- static_pipe
 - cl::sycl::static_pipe, 117
- std, 337
- std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >, 364
 - operator(), 364
- std::hash< cl::sycl::device >, 365
 - operator(), 365
- std::hash< cl::sycl::kernel >, 366
 - operator(), 366
- std::hash< cl::sycl::platform >, 367
 - operator(), 367
- std::hash< cl::sycl::queue >, 368
 - operator(), 368
- string_class
 - cl::sycl, 327
- submit
 - cl::sycl::queue, 205
- super
 - cl::sycl::detail::address_space_array, 126
 - cl::sycl::detail::address_space_fundamental, 129
 - cl::sycl::detail::address_space_ptr, 134
 - cl::sycl::detail::address_space_variable, 139
- TRISYCL_ASYNC
 - Manage default configuration and types, 225
- TRISYCL_BOOST_OPERATOR_VECTOR_OP
 - Some helpers for the implementation, 237
- TRISYCL_DEFINE_VEC_TYPE_SIZE
 - Vector types in SYCL, 321
- TRISYCL_DEFINE_VEC_TYPE
 - Vector types in SYCL, 321
- TRISYCL_DUMP_T
 - debug.hpp, 478
- TRISYCL_DUMP
 - debug.hpp, 478
- TRISYCL_INFO_PARAM_TRAITS_ANY_T
 - param_traits.hpp, 552
- TRISYCL_INFO_PARAM_TRAITS
 - param_traits.hpp, 552
- TRISYCL_INTERNAL_DUMP
 - debug.hpp, 478
- TRISYCL_ParallelForFunctor_GLOBAL_OFFSET
 - handler.hpp, 539
- TRISYCL_ParallelForKernel_RANGE
 - cl::sycl::detail::kernel, 180
 - cl::sycl::detail::opencl_kernel, 385
 - cl::sycl::handler, 178
 - handler.hpp, 539
 - kernel/detail/kernel.hpp, 567
 - opencl_kernel.hpp, 573
- TRISYCL_SKIP_OPENCL
 - Manage default configuration and types, 225
 - Platforms, contexts, devices and queues, 214
- TRISYCL_parallel_for_functor_GLOBAL
 - cl::sycl::handler, 177
 - handler.hpp, 539
- target
 - cl::sycl::access, 329
 - cl::sycl::detail::pipe_accessor, 93
 - cl::sycl::detail::pipe_reservation, 105
- task
 - cl::sycl::detail::accessor, 60
 - cl::sycl::detail::task, 407
 - cl::sycl::handler, 178
- throw_asynchronous
 - cl::sycl::queue, 205
- trace_kernel
 - Debugging and tracing support, 222

- tuple_to_array
 - Helpers to do array and tuple conversion, [220](#)
- tuple_to_array_iterate
 - Helpers to do array and tuple conversion, [220](#)
- type
 - cl::sycl::detail::address_space_base, [136](#)
 - cl::sycl::detail::opencl_type, [123](#)
 - cl::sycl::detail::opencl_type< T, constant_↵
address_space >, [123](#)
 - cl::sycl::detail::opencl_type< T, generic_address_↵
_space >, [123](#)
 - cl::sycl::detail::opencl_type< T, global_address_↵
space >, [124](#)
 - cl::sycl::detail::opencl_type< T, local_address_↵
space >, [124](#)
 - cl::sycl::detail::opencl_type< T, private_address_↵
_space >, [125](#)
 - cl::sycl::device, [161](#)
- unimplemented
 - Some helpers for the implementation, [238](#)
- unique_ptr_class
 - cl::sycl, [327](#)
- unsupported
 - Platforms, contexts, devices and queues, [211](#)
- use
 - cl::sycl::detail::buffer_base, [356](#)
- use_count
 - cl::sycl::buffer, [352](#)
- used_for_reading
 - cl::sycl::detail::pipe, [83](#)
- used_for_writing
 - cl::sycl::detail::pipe, [83](#)
- value_type
 - cl::sycl::accessor, [30](#)
 - cl::sycl::buffer, [342](#)
 - cl::sycl::detail::accessor, [48](#)
 - cl::sycl::detail::buffer, [63](#)
 - cl::sycl::detail::cache, [359](#)
 - cl::sycl::detail::pipe, [75](#)
 - cl::sycl::detail::pipe_accessor, [86](#)
 - cl::sycl::detail::pipe_reservation, [98](#)
 - cl::sycl::pipe, [94](#)
 - cl::sycl::pipe_reservation, [109](#)
 - cl::sycl::static_pipe, [117](#)
- variable
 - cl::sycl::detail::address_space_variable, [140](#)
- vec
 - cl::sycl::vec, [319](#)
- Vector types in SYCL, [317](#)
 - TRISYCL_DEFINE_VEC_TYPE_SIZE, [321](#)
 - TRISYCL_DEFINE_VEC_TYPE, [321](#)
- vector_class
 - cl::sycl, [328](#)
- vendor
 - Platforms, contexts, devices and queues, [209](#)
- vendor_id
 - Platforms, contexts, devices and queues, [208](#)
- w_rid_q
 - cl::sycl::detail::pipe, [83](#)
- wait
 - cl::sycl::detail::buffer_base, [356](#)
 - cl::sycl::detail::task, [412](#)
 - cl::sycl::queue, [206](#)
- wait_and_throw
 - cl::sycl::queue, [206](#)
- wait_for_kernel_execution
 - cl::sycl::detail::queue, [397](#)
- wait_for_producers
 - cl::sycl::detail::task, [412](#)
- waiter
 - Data access and storage in SYCL, [119](#)
- weak_ptr_class
 - cl::sycl, [328](#)
- what
 - cl::sycl::exception, [245](#)
- writable_array_view_type
 - cl::sycl::detail::accessor, [48](#)
- write
 - cl::sycl::access, [329](#)
 - cl::sycl::detail::pipe, [82](#)
 - cl::sycl::detail::pipe_accessor, [91](#)
- write_done
 - cl::sycl::detail::pipe, [83](#)
- write_only
 - Platforms, contexts, devices and queues, [213](#)