

triSYCL implementation of OpenCL SYCL

Generated by Doxygen 1.8.13



# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Main Page</b>  | <b>1</b>  |
| <b>2</b> | <b>Todo List</b>  | <b>3</b>  |
| <b>3</b> | <b>Module Index</b>   | <b>15</b> |
| 3.1      | Modules . . . . .   | 15        |
| <b>4</b> | <b>Namespace Index</b>  | <b>17</b> |
| 4.1      | Namespace List . . . . .  | 17        |
| <b>5</b> | <b>Hierarchical Index</b>   | <b>19</b> |
| 5.1      | Class Hierarchy . . . . .   | 19        |
| <b>6</b> | <b>Class Index</b>  | <b>25</b> |
| 6.1      | Class List . . . . .  | 25        |
| <b>7</b> | <b>File Index</b>   | <b>27</b> |
| 7.1      | File List . . . . .   | 27        |
| <b>8</b> | <b>Module Documentation</b>   | <b>29</b> |
| 8.1      | Data access and storage in SYCL . . . . .   | 29        |
| 8.1.1    | Detailed Description . . . . .  | 30        |
| 8.1.2    | Class Documentation . . . . .   | 30        |
| 8.1.2.1  | class <code>cl::sycl::detail::accessor&lt; T, Dimensions, Mode, access::target::local &gt;</code> . .                 | 30        |
| 8.1.2.2  | class <code>cl::sycl::accessor</code> . . . . .   | 48        |
| 8.1.2.3  | class <code>cl::sycl::accessor&lt; DataType, 1, AccessMode, access::target::pipe &gt;</code> . . .                    | 64        |
| 8.1.2.4  | class <code>cl::sycl::accessor&lt; DataType, 1, AccessMode, access::target::blocking_↔<br/>pipe &gt;</code> . . . . . | 67        |

|          |   |     |
|----------|---|-----|
| 8.1.2.5  | <code>class cl::sycl::detail::accessor</code>                                     | 70  |
| 8.1.2.6  | <code>class cl::sycl::detail::buffer</code>                                       | 92  |
| 8.1.2.7  | <code>struct cl::sycl::detail::buffer_base</code>                                 | 109 |
| 8.1.2.8  | <code>class cl::sycl::detail::buffer_waiter</code>                                | 121 |
| 8.1.2.9  | <code>class cl::sycl::buffer</code>   | 123 |
| 8.1.2.10 | <code>struct cl::sycl::image</code>   | 141 |
| 8.1.2.11 | <code>struct cl::sycl::detail::reserve_id</code>                                  | 142 |
| 8.1.2.12 | <code>class cl::sycl::detail::pipe</code>   | 144 |
| 8.1.2.13 | <code>class cl::sycl::detail::pipe_accessor</code>                                | 158 |
| 8.1.2.14 | <code>class cl::sycl::pipe</code>   | 169 |
| 8.1.2.15 | <code>class cl::sycl::detail::pipe_reservation</code>                             | 172 |
| 8.1.2.16 | <code>struct cl::sycl::pipe_reservation</code>                                    | 183 |
| 8.1.2.17 | <code>class cl::sycl::static_pipe</code>  | 195 |
| 8.1.3    | Typedef Documentation   | 199 |
| 8.1.3.1  | <code>buffer_allocator</code>   | 199 |
| 8.1.3.2  | <code>image_allocator</code>  | 200 |
| 8.1.3.3  | <code>map_allocator</code>  | 200 |
| 8.1.4    | Function Documentation  | 200 |
| 8.1.4.1  | <code>add_buffer_to_task()</code>   | 200 |
| 8.1.4.2  | <code>buffer_add_to_task()</code>   | 201 |
| 8.1.4.3  | <code>get_pipe_detail()</code>  | 201 |
| 8.1.4.4  | <code>waiter()</code>   | 202 |
| 8.2      | Dealing with OpenCL address spaces  | 203 |
| 8.2.1    | Detailed Description  | 205 |
| 8.2.2    | Class Documentation   | 205 |
| 8.2.2.1  | <code>struct cl::sycl::detail::ocl_type</code>                                    | 205 |
| 8.2.2.2  | <code>struct cl::sycl::detail::ocl_type&lt; T, constant_address_space &gt;</code> | 205 |
| 8.2.2.3  | <code>struct cl::sycl::detail::ocl_type&lt; T, generic_address_space &gt;</code>  | 206 |
| 8.2.2.4  | <code>struct cl::sycl::detail::ocl_type&lt; T, global_address_space &gt;</code>   | 206 |
| 8.2.2.5  | <code>struct cl::sycl::detail::ocl_type&lt; T, local_address_space &gt;</code>    | 207 |

|          |  |     |
|----------|--|-----|
| 8.2.2.6  | <code>struct cl::sycl::detail::ocl_type&lt; T, private_address_space &gt;</code> | 207 |
| 8.2.2.7  | <code>struct cl::sycl::detail::address_space_array</code>                        | 207 |
| 8.2.2.8  | <code>struct cl::sycl::detail::address_space_fundamental</code>                  | 210 |
| 8.2.2.9  | <code>struct cl::sycl::detail::address_space_object</code>                       | 213 |
| 8.2.2.10 | <code>struct cl::sycl::detail::address_space_ptr</code>                          | 216 |
| 8.2.2.11 | <code>struct cl::sycl::detail::address_space_base</code>                         | 220 |
| 8.2.2.12 | <code>struct cl::sycl::detail::address_space_variable</code>                     | 222 |
| 8.2.3    | Typedef Documentation  | 226 |
| 8.2.3.1  | <code>addr_space</code>  | 226 |
| 8.2.3.2  | <code>constant</code>  | 227 |
| 8.2.3.3  | <code>constant_ptr</code>  | 227 |
| 8.2.3.4  | <code>generic</code>   | 227 |
| 8.2.3.5  | <code>global</code>  | 228 |
| 8.2.3.6  | <code>global_ptr</code>  | 228 |
| 8.2.3.7  | <code>local</code>   | 228 |
| 8.2.3.8  | <code>local_ptr</code>   | 229 |
| 8.2.3.9  | <code>multi_ptr</code>   | 229 |
| 8.2.3.10 | <code>priv</code>  | 229 |
| 8.2.3.11 | <code>private_ptr</code>   | 230 |
| 8.2.4    | Enumeration Type Documentation   | 230 |
| 8.2.4.1  | <code>address_space</code>   | 230 |
| 8.2.5    | Function Documentation   | 231 |
| 8.2.5.1  | <code>make_multi()</code>  | 231 |
| 8.3      | Platforms, contexts, devices and queues  | 232 |
| 8.3.1    | Detailed Description   | 235 |
| 8.3.2    | Class Documentation  | 235 |
| 8.3.2.1  | <code>class cl::sycl::detail::context</code>                                     | 235 |
| 8.3.2.2  | <code>class cl::sycl::detail::host_context</code>                                | 238 |
| 8.3.2.3  | <code>class cl::sycl::context</code>   | 242 |
| 8.3.2.4  | <code>class cl::sycl::detail::device</code>                                      | 252 |

|          |  |     |
|----------|--|-----|
| 8.3.2.5  | <a href="#">class cl::sycl::device</a>                   | 255 |
| 8.3.2.6  | <a href="#">class cl::sycl::device_type_selector</a>     | 266 |
| 8.3.2.7  | <a href="#">class cl::sycl::device_typename_selector</a> | 269 |
| 8.3.2.8  | <a href="#">class cl::sycl::device_selector</a>          | 271 |
| 8.3.2.9  | <a href="#">class cl::sycl::handler</a>                  | 273 |
| 8.3.2.10 | <a href="#">class cl::sycl::detail::kernel</a>           | 282 |
| 8.3.2.11 | <a href="#">class cl::sycl::kernel</a>                   | 285 |
| 8.3.2.12 | <a href="#">class cl::sycl::detail::host_platform</a>    | 289 |
| 8.3.2.13 | <a href="#">class cl::sycl::detail::opencl_platform</a>  | 293 |
| 8.3.2.14 | <a href="#">class cl::sycl::detail::platform</a>         | 298 |
| 8.3.2.15 | <a href="#">class cl::sycl::platform</a>                 | 300 |
| 8.3.2.16 | <a href="#">class cl::sycl::queue</a>                    | 308 |
| 8.3.3    | <a href="#">Typedef Documentation</a>                    | 319 |
| 8.3.3.1  | <a href="#">cpu_selector</a>                             | 320 |
| 8.3.3.2  | <a href="#">default_selector</a>                         | 320 |
| 8.3.3.3  | <a href="#">device_exec_capabilities</a>                 | 320 |
| 8.3.3.4  | <a href="#">device_fp_config</a>                         | 321 |
| 8.3.3.5  | <a href="#">device_queue_properties</a>                  | 321 |
| 8.3.3.6  | <a href="#">gl_context_interop</a>                       | 321 |
| 8.3.3.7  | <a href="#">gpu_selector</a>                             | 321 |
| 8.3.3.8  | <a href="#">host_selector</a>                            | 321 |
| 8.3.4    | <a href="#">Enumeration Type Documentation</a>           | 322 |
| 8.3.4.1  | <a href="#">context</a>                                  | 322 |
| 8.3.4.2  | <a href="#">device</a>                                   | 322 |
| 8.3.4.3  | <a href="#">device_affinity_domain</a>                   | 325 |
| 8.3.4.4  | <a href="#">device_execution_capabilities</a>            | 326 |
| 8.3.4.5  | <a href="#">device_partition_property</a>                | 326 |
| 8.3.4.6  | <a href="#">device_partition_type</a>                    | 326 |
| 8.3.4.7  | <a href="#">device_type</a>                              | 327 |
| 8.3.4.8  | <a href="#">fp_config</a>                                | 328 |

|          |   |     |
|----------|---|-----|
| 8.3.4.9  | <a href="#">global_mem_cache_type</a>   | 328 |
| 8.3.4.10 | <a href="#">local_mem_type</a>  | 329 |
| 8.3.4.11 | <a href="#">platform</a>  | 329 |
| 8.3.5    | <a href="#">Function Documentation</a>  | 330 |
| 8.3.5.1  | <a href="#">device::get_info&lt; info::device::device_type &gt;()</a>             | 330 |
| 8.3.5.2  | <a href="#">device::get_info&lt; info::device::local_mem_size &gt;()</a>          | 331 |
| 8.3.5.3  | <a href="#">device::get_info&lt; info::device::max_compute_units &gt;()</a>       | 331 |
| 8.3.5.4  | <a href="#">device::get_info&lt; info::device::max_mem_alloc_size &gt;()</a>      | 331 |
| 8.3.5.5  | <a href="#">device::get_info&lt; info::device::max_work_group_size &gt;()</a>     | 332 |
| 8.3.5.6  | <a href="#">device::get_info&lt; info::device::name &gt;()</a>                    | 332 |
| 8.3.5.7  | <a href="#">device::get_info&lt; info::device::profile &gt;()</a>                 | 332 |
| 8.3.5.8  | <a href="#">device::get_info&lt; info::device::vendor &gt;()</a>                  | 333 |
| 8.3.5.9  | <a href="#">get_devices()</a> [1/3]   | 333 |
| 8.3.5.10 | <a href="#">get_devices()</a> [2/3]   | 334 |
| 8.3.5.11 | <a href="#">get_devices()</a> [3/3]   | 335 |
| 8.3.6    | <a href="#">Variable Documentation</a>  | 336 |
| 8.3.6.1  | <a href="#">TRISYCL_WEAK_ATTRIB_SUFFIX</a>  | 336 |
| 8.4      | <a href="#">Helpers to do array and tuple conversion</a>                          | 337 |
| 8.4.1    | <a href="#">Detailed Description</a>  | 337 |
| 8.4.2    | <a href="#">Class Documentation</a>   | 337 |
| 8.4.2.1  | <a href="#">struct cl::sycl::detail::expand_to_vector</a>                         | 337 |
| 8.4.2.2  | <a href="#">struct cl::sycl::detail::expand_to_vector&lt; V, Tuple, true &gt;</a> | 338 |
| 8.4.3    | <a href="#">Function Documentation</a>  | 338 |
| 8.4.3.1  | <a href="#">expand()</a> [1/3]  | 338 |
| 8.4.3.2  | <a href="#">expand()</a> [2/3]  | 339 |
| 8.4.3.3  | <a href="#">expand()</a> [3/3]  | 339 |
| 8.4.3.4  | <a href="#">fill_tuple()</a>  | 340 |
| 8.4.3.5  | <a href="#">tuple_to_array()</a>  | 340 |
| 8.4.3.6  | <a href="#">tuple_to_array_iterate()</a>  | 341 |
| 8.5      | <a href="#">Some helpers for the implementation</a>                               | 342 |

|         |   |     |
|---------|---|-----|
| 8.5.1   | Detailed Description  | 342 |
| 8.5.2   | Class Documentation   | 342 |
| 8.5.2.1 | struct cl::sycl::detail::container_element_aspect                   | 342 |
| 8.5.2.2 | struct cl::sycl::detail::small_array                                | 343 |
| 8.5.2.3 | struct cl::sycl::detail::small_array_123                            | 351 |
| 8.5.2.4 | struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 > | 352 |
| 8.5.2.5 | struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 > | 354 |
| 8.5.2.6 | struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 > | 357 |
| 8.5.3   | Macro Definition Documentation                                      | 359 |
| 8.5.3.1 | TRISYCL_BOOST_OPERATOR_VECTOR_OP                                    | 359 |
| 8.5.3.2 | TRISYCL_LOGICAL_OPERATOR_VECTOR_OP                                  | 360 |
| 8.5.4   | Function Documentation  | 360 |
| 8.5.4.1 | linear_id()   | 360 |
| 8.5.4.2 | unimplemented()   | 361 |
| 8.6     | Debugging and tracing support                                       | 363 |
| 8.6.1   | Detailed Description  | 363 |
| 8.6.2   | Class Documentation   | 363 |
| 8.6.2.1 | struct cl::sycl::detail::debug                                      | 363 |
| 8.6.2.2 | struct cl::sycl::detail::display_vector                             | 363 |
| 8.6.3   | Function Documentation  | 364 |
| 8.6.3.1 | trace_kernel()  | 365 |
| 8.7     | Manage default configuration and types                              | 366 |
| 8.7.1   | Detailed Description  | 366 |
| 8.7.2   | Macro Definition Documentation                                      | 366 |
| 8.7.2.1 | __SYCL_SINGLE_SOURCE__  | 366 |
| 8.7.2.2 | CL_SYCL_LANGUAGE_VERSION  | 367 |
| 8.7.2.3 | TRISYCL_CL_LANGUAGE_VERSION   | 367 |
| 8.7.2.4 | TRISYCL_MAKE_BOOST_CIRCULARBUFFER_THREAD_SAFE                       | 367 |
| 8.7.2.5 | TRISYCL_SKIP_OPENCL   | 367 |
| 8.8     | Error handling  | 368 |



|          |   |     |
|----------|---|-----|
| 8.8.1    | Detailed Description                    | 369 |
| 8.8.2    | Class Documentation                     | 369 |
| 8.8.2.1  | struct cl::sycl::error_handler          | 369 |
| 8.8.2.2  | struct cl::sycl::exception_list         | 371 |
| 8.8.2.3  | class cl::sycl::exception               | 371 |
| 8.8.2.4  | class cl::sycl::cl_exception            | 374 |
| 8.8.2.5  | struct cl::sycl::async_exception        | 376 |
| 8.8.2.6  | class cl::sycl::runtime_error           | 377 |
| 8.8.2.7  | class cl::sycl::kernel_error            | 378 |
| 8.8.2.8  | class cl::sycl::accessor_error          | 379 |
| 8.8.2.9  | class cl::sycl::nd_range_error          | 380 |
| 8.8.2.10 | class cl::sycl::event_error             | 381 |
| 8.8.2.11 | class cl::sycl::invalid_parameter_error | 382 |
| 8.8.2.12 | class cl::sycl::device_error            | 383 |
| 8.8.2.13 | class cl::sycl::compile_program_error   | 384 |
| 8.8.2.14 | class cl::sycl::link_program_error      | 386 |
| 8.8.2.15 | class cl::sycl::invalid_object_error    | 387 |
| 8.8.2.16 | class cl::sycl::memory_allocation_error | 388 |
| 8.8.2.17 | class cl::sycl::pipe_error              | 389 |
| 8.8.2.18 | class cl::sycl::platform_error          | 390 |
| 8.8.2.19 | class cl::sycl::profiling_error         | 391 |
| 8.8.2.20 | class cl::sycl::feature_not_supported   | 392 |
| 8.8.2.21 | class cl::sycl::non_cl_error            | 393 |
| 8.8.3    | Typedef Documentation                   | 394 |
| 8.8.3.1  | async_handler                           | 394 |
| 8.8.3.2  | exception_ptr                           | 394 |
| 8.9      | Expressing parallelism through kernels  | 395 |
| 8.9.1    | Detailed Description                    | 396 |
| 8.9.2    | Class Documentation                     | 396 |
| 8.9.2.1  | struct cl::sycl::group                  | 396 |

|          |  |     |
|----------|--|-----|
| 8.9.2.2  | <code>class cl::sycl::id</code>  | 407 |
| 8.9.2.3  | <code>class cl::sycl::item</code>  | 409 |
| 8.9.2.4  | <code>struct cl::sycl::nd_item</code>  | 416 |
| 8.9.2.5  | <code>struct cl::sycl::nd_range</code>   | 434 |
| 8.9.2.6  | <code>struct cl::sycl::detail::parallel_for_iterate</code>   | 438 |
| 8.9.2.7  | <code>struct cl::sycl::detail::parallel_OpenMP_for_iterate</code>                                    | 439 |
| 8.9.2.8  | <code>struct cl::sycl::detail::parallel_for_iterate&lt; 0, Range, ParallelForFunctor, Id &gt;</code> | 440 |
| 8.9.2.9  | <code>class cl::sycl::range</code>   | 441 |
| 8.9.3    | Function Documentation   | 443 |
| 8.9.3.1  | <code>make_id()</code> [1/4]   | 443 |
| 8.9.3.2  | <code>make_id()</code> [2/4]   | 443 |
| 8.9.3.3  | <code>make_id()</code> [3/4]   | 444 |
| 8.9.3.4  | <code>make_id()</code> [4/4]   | 444 |
| 8.9.3.5  | <code>make_range()</code> [1/4]  | 444 |
| 8.9.3.6  | <code>make_range()</code> [2/4]  | 445 |
| 8.9.3.7  | <code>make_range()</code> [3/4]  | 445 |
| 8.9.3.8  | <code>make_range()</code> [4/4]  | 445 |
| 8.9.3.9  | <code>parallel_for()</code> [1/4]  | 446 |
| 8.9.3.10 | <code>parallel_for()</code> [2/4]  | 447 |
| 8.9.3.11 | <code>parallel_for()</code> [3/4]  | 447 |
| 8.9.3.12 | <code>parallel_for()</code> [4/4]  | 448 |
| 8.9.3.13 | <code>parallel_for_global_offset()</code>  | 449 |
| 8.9.3.14 | <code>parallel_for_work_item()</code>  | 450 |
| 8.9.3.15 | <code>parallel_for_workgroup()</code>  | 451 |
| 8.9.3.16 | <code>parallel_for_workitem()</code>   | 452 |
| 8.10     | Vector types in SYCL   | 455 |
| 8.10.1   | Detailed Description   | 455 |
| 8.10.2   | Class Documentation  | 455 |
| 8.10.2.1 | <code>class cl::sycl::vec</code>   | 455 |
| 8.10.3   | Macro Definition Documentation   | 459 |
| 8.10.3.1 | <code>TRISYCL_DEFINE_VEC_TYPE</code>   | 460 |
| 8.10.3.2 | <code>TRISYCL_DEFINE_VEC_TYPE_SIZE</code>  | 460 |

|   |            |
|---|------------|
| <b>9 Namespace Documentation</b>          | <b>461</b> |
| 9.1 cl Namespace Reference                | 461        |
| 9.1.1 Detailed Description                | 461        |
| 9.2 cl::sycl Namespace Reference          | 461        |
| 9.2.1 Typedef Documentation               | 466        |
| 9.2.1.1 function_class                    | 466        |
| 9.2.1.2 mutex_class                       | 466        |
| 9.2.1.3 shared_ptr_class                  | 466        |
| 9.2.1.4 string_class                      | 467        |
| 9.2.1.5 unique_ptr_class                  | 467        |
| 9.2.1.6 vector_class                      | 467        |
| 9.2.1.7 weak_ptr_class                    | 467        |
| 9.2.2 Function Documentation              | 467        |
| 9.2.2.1 min()                             | 467        |
| 9.2.2.2 TRISYCL_MATH_WRAP()               | 468        |
| 9.2.2.3 TRISYCL_MATH_WRAP2s()             | 468        |
| 9.2.3 Variable Documentation              | 469        |
| 9.2.3.1 y                                 | 469        |
| 9.2.3.2 z                                 | 469        |
| 9.3 cl::sycl::access Namespace Reference  | 469        |
| 9.3.1 Detailed Description                | 469        |
| 9.3.2 Enumeration Type Documentation      | 469        |
| 9.3.2.1 fence_space                       | 469        |
| 9.3.2.2 mode                              | 470        |
| 9.3.2.3 target                            | 470        |
| 9.4 cl::sycl::detail Namespace Reference  | 471        |
| 9.4.1 Function Documentation              | 475        |
| 9.4.1.1 add_buffer_to_task()              | 475        |
| 9.5 cl::sycl::info Namespace Reference    | 475        |
| 9.5.1 Typedef Documentation               | 477        |
| 9.5.1.1 queue_profiling                   | 477        |
| 9.5.2 Enumeration Type Documentation      | 477        |
| 9.5.2.1 queue                             | 477        |
| 9.6 cl::sycl::trisycl Namespace Reference | 478        |
| 9.6.1 Detailed Description                | 478        |
| 9.7 std Namespace Reference               | 478        |

|   |            |
|---|------------|
| <b>10 Class Documentation</b>   | <b>479</b> |
| 10.1 <code>cl::sycl::detail::cache&lt; Key, Value &gt;</code> Class Template Reference                            | 479        |
| 10.1.1 Detailed Description   | 480        |
| 10.1.2 Member Typedef Documentation   | 480        |
| 10.1.2.1 <code>key_type</code>  | 480        |
| 10.1.2.2 <code>value_type</code>  | 480        |
| 10.1.3 Member Function Documentation  | 481        |
| 10.1.3.1 <code>get_or_register()</code>   | 481        |
| 10.1.3.2 <code>remove()</code>  | 482        |
| 10.1.4 Member Data Documentation  | 483        |
| 10.1.4.1 <code>c</code>   | 483        |
| 10.1.4.2 <code>m</code>   | 483        |
| 10.2 <code>cl::sycl::trisycl::default_error_handler</code> Struct Reference                                       | 484        |
| 10.2.1 Detailed Description   | 484        |
| 10.2.2 Member Function Documentation  | 485        |
| 10.2.2.1 <code>report_error()</code>  | 485        |
| 10.3 <code>cl::sycl::event</code> Class Reference   | 485        |
| 10.3.1 Detailed Description   | 485        |
| 10.3.2 Constructor & Destructor Documentation   | 485        |
| 10.3.2.1 <code>event()</code>   | 486        |
| 10.4 <code>handler_event</code> Class Reference   | 486        |
| 10.4.1 Detailed Description   | 486        |
| 10.5 <code>std::hash&lt; cl::sycl::buffer&lt; T, Dimensions, Allocator &gt; &gt;</code> Struct Template Reference | 486        |
| 10.5.1 Detailed Description   | 486        |
| 10.5.2 Member Function Documentation  | 487        |
| 10.5.2.1 <code>operator()()</code>  | 487        |
| 10.6 <code>std::hash&lt; cl::sycl::context &gt;</code> Struct Template Reference                                  | 487        |
| 10.6.1 Detailed Description   | 488        |
| 10.6.2 Member Function Documentation  | 488        |
| 10.6.2.1 <code>operator()()</code>  | 488        |

|           |   |     |
|-----------|---|-----|
| 10.7      | <a href="#">std::hash&lt; cl::sycl::device &gt; Struct Template Reference</a>   | 488 |
| 10.7.1    | <a href="#">Detailed Description</a>  | 489 |
| 10.7.2    | <a href="#">Member Function Documentation</a>                                   | 489 |
| 10.7.2.1  | <a href="#">operator()()</a>  | 489 |
| 10.8      | <a href="#">std::hash&lt; cl::sycl::kernel &gt; Struct Template Reference</a>   | 489 |
| 10.8.1    | <a href="#">Detailed Description</a>  | 490 |
| 10.8.2    | <a href="#">Member Function Documentation</a>                                   | 490 |
| 10.8.2.1  | <a href="#">operator()()</a>  | 490 |
| 10.9      | <a href="#">std::hash&lt; cl::sycl::platform &gt; Struct Template Reference</a> | 490 |
| 10.9.1    | <a href="#">Detailed Description</a>  | 491 |
| 10.9.2    | <a href="#">Member Function Documentation</a>                                   | 491 |
| 10.9.2.1  | <a href="#">operator()()</a>  | 491 |
| 10.10     | <a href="#">std::hash&lt; cl::sycl::queue &gt; Struct Template Reference</a>    | 491 |
| 10.10.1   | <a href="#">Detailed Description</a>  | 492 |
| 10.10.2   | <a href="#">Member Function Documentation</a>                                   | 492 |
| 10.10.2.1 | <a href="#">operator()()</a>  | 492 |
| 10.11     | <a href="#">cl::sycl::detail::host_device Class Reference</a>                   | 493 |
| 10.11.1   | <a href="#">Detailed Description</a>  | 494 |
| 10.11.2   | <a href="#">Member Function Documentation</a>                                   | 494 |
| 10.11.2.1 | <a href="#">get()</a>   | 494 |
| 10.11.2.2 | <a href="#">get_boost_compute()</a>   | 495 |
| 10.11.2.3 | <a href="#">get_platform()</a>  | 495 |
| 10.11.2.4 | <a href="#">has_extension()</a>   | 496 |
| 10.11.2.5 | <a href="#">is_accelerator()</a>  | 496 |
| 10.11.2.6 | <a href="#">is_cpu()</a>  | 497 |
| 10.11.2.7 | <a href="#">is_gpu()</a>  | 497 |
| 10.11.2.8 | <a href="#">is_host()</a>   | 497 |
| 10.12     | <a href="#">cl::sycl::detail::host_queue Class Reference</a>                    | 498 |
| 10.12.1   | <a href="#">Detailed Description</a>  | 499 |
| 10.12.2   | <a href="#">Member Function Documentation</a>                                   | 499 |

|  |     |
|--|-----|
| 10.12.2.1 <code>get()</code> . . . . .   | 499 |
| 10.12.2.2 <code>get_boost_compute()</code> . . . . .                                   | 500 |
| 10.12.2.3 <code>get_context()</code> . . . . .   | 500 |
| 10.12.2.4 <code>get_device()</code> . . . . .  | 500 |
| 10.12.2.5 <code>is_host()</code> . . . . .   | 501 |
| 10.13 <code>cl::sycl::is_wrapper&lt; T &gt;</code> Struct Template Reference . . . . . | 501 |
| 10.13.1 Detailed Description . . . . .   | 502 |
| 10.14 <code>cl::sycl::detail::opencl_context</code> Class Reference . . . . .          | 502 |
| 10.14.1 Detailed Description . . . . .   | 503 |
| 10.14.2 Constructor & Destructor Documentation . . . . .                               | 504 |
| 10.14.2.1 <code>opencl_context()</code> . . . . .                                      | 504 |
| 10.14.2.2 <code>~opencl_context()</code> . . . . .                                     | 504 |
| 10.14.3 Member Function Documentation . . . . .  | 504 |
| 10.14.3.1 <code>get()</code> . . . . .   | 505 |
| 10.14.3.2 <code>get_boost_compute()</code> . . . . .                                   | 505 |
| 10.14.3.3 <code>get_boost_queue()</code> . . . . .                                     | 505 |
| 10.14.3.4 <code>get_devices()</code> . . . . .   | 506 |
| 10.14.3.5 <code>get_platform()</code> . . . . .  | 506 |
| 10.14.3.6 <code>instance()</code> . . . . .  | 507 |
| 10.14.3.7 <code>is_host()</code> . . . . .   | 508 |
| 10.14.4 Member Data Documentation . . . . .  | 508 |
| 10.14.4.1 <code>c</code> . . . . .   | 509 |
| 10.14.4.2 <code>cache</code> . . . . .   | 509 |
| 10.14.4.3 <code>q</code> . . . . .   | 509 |
| 10.15 <code>cl::sycl::detail::opencl_device</code> Class Reference . . . . .           | 510 |
| 10.15.1 Detailed Description . . . . .   | 511 |
| 10.15.2 Constructor & Destructor Documentation . . . . .                               | 511 |
| 10.15.2.1 <code>opencl_device()</code> . . . . .                                       | 511 |
| 10.15.2.2 <code>~opencl_device()</code> . . . . .                                      | 512 |
| 10.15.3 Member Function Documentation . . . . .  | 512 |

|           |   |     |
|-----------|---|-----|
| 10.15.3.1 | <a href="#">get()</a>   | 512 |
| 10.15.3.2 | <a href="#">get_boost_compute()</a>                             | 513 |
| 10.15.3.3 | <a href="#">get_platform()</a>                                  | 513 |
| 10.15.3.4 | <a href="#">has_extension()</a>                                 | 514 |
| 10.15.3.5 | <a href="#">instance()</a>                                      | 514 |
| 10.15.3.6 | <a href="#">is_accelerator()</a>                                | 515 |
| 10.15.3.7 | <a href="#">is_cpu()</a>  | 515 |
| 10.15.3.8 | <a href="#">is_gpu()</a>  | 516 |
| 10.15.3.9 | <a href="#">is_host()</a>                                       | 516 |
| 10.15.4   | <a href="#">Member Data Documentation</a>                       | 516 |
| 10.15.4.1 | <a href="#">cache</a>   | 516 |
| 10.15.4.2 | <a href="#">d</a>   | 517 |
| 10.16     | <a href="#">cl::sycl::detail::opencl_kernel Class Reference</a> | 517 |
| 10.16.1   | <a href="#">Detailed Description</a>                            | 519 |
| 10.16.2   | <a href="#">Constructor &amp; Destructor Documentation</a>      | 519 |
| 10.16.2.1 | <a href="#">opencl_kernel()</a>                                 | 519 |
| 10.16.3   | <a href="#">Member Function Documentation</a>                   | 519 |
| 10.16.3.1 | <a href="#">get()</a>   | 519 |
| 10.16.3.2 | <a href="#">get_boost_compute()</a>                             | 520 |
| 10.16.3.3 | <a href="#">instance()</a>                                      | 520 |
| 10.16.3.4 | <a href="#">single_task()</a>                                   | 521 |
| 10.16.3.5 | <a href="#">TRISYCL_ParallelForKernel_RANGE()</a>               | 522 |
| 10.16.4   | <a href="#">Member Data Documentation</a>                       | 522 |
| 10.16.4.1 | <a href="#">cache</a>   | 522 |
| 10.16.4.2 | <a href="#">k</a>   | 523 |
| 10.17     | <a href="#">cl::sycl::detail::opencl_queue Class Reference</a>  | 523 |
| 10.17.1   | <a href="#">Detailed Description</a>                            | 525 |
| 10.17.2   | <a href="#">Constructor &amp; Destructor Documentation</a>      | 525 |
| 10.17.2.1 | <a href="#">opencl_queue()</a>                                  | 525 |
| 10.17.2.2 | <a href="#">~opencl_queue()</a>                                 | 525 |

|   |     |
|---|-----|
| 10.17.3 Member Function Documentation . . . . .                                   | 526 |
| 10.17.3.1 get() . . . . .   | 526 |
| 10.17.3.2 get_boost_compute() . . . . .   | 526 |
| 10.17.3.3 get_context() . . . . .   | 526 |
| 10.17.3.4 get_device() . . . . .  | 527 |
| 10.17.3.5 instance() [1/2] . . . . .  | 527 |
| 10.17.3.6 instance() [2/2] . . . . .  | 528 |
| 10.17.3.7 is_host() . . . . .   | 528 |
| 10.17.4 Member Data Documentation . . . . .                                       | 528 |
| 10.17.4.1 cache . . . . .   | 529 |
| 10.17.4.2 q . . . . .   | 529 |
| 10.18cl::sycl::info::param_traits< T, Param > Struct Template Reference . . . . . | 529 |
| 10.18.1 Detailed Description . . . . .  | 529 |
| 10.19cl::sycl::detail::queue Struct Reference . . . . .                           | 530 |
| 10.19.1 Detailed Description . . . . .  | 531 |
| 10.19.2 Constructor & Destructor Documentation . . . . .                          | 531 |
| 10.19.2.1 queue() . . . . .   | 531 |
| 10.19.2.2 ~queue() . . . . .  | 532 |
| 10.19.3 Member Function Documentation . . . . .                                   | 532 |
| 10.19.3.1 get() . . . . .   | 532 |
| 10.19.3.2 get_boost_compute() . . . . .   | 533 |
| 10.19.3.3 get_context() . . . . .   | 533 |
| 10.19.3.4 get_device() . . . . .  | 534 |
| 10.19.3.5 is_host() . . . . .   | 534 |
| 10.19.3.6 kernel_end() . . . . .  | 535 |
| 10.19.3.7 kernel_start() . . . . .  | 535 |
| 10.19.3.8 wait_for_kernel_execution() . . . . .                                   | 536 |
| 10.19.4 Member Data Documentation . . . . .                                       | 536 |
| 10.19.4.1 finished . . . . .  | 536 |
| 10.19.4.2 finished_mutex . . . . .  | 536 |



|  |     |
|--|-----|
| 10.19.4.3 <code>running_kernels</code>   | 537 |
| 10.20 <code>cl::sycl::detail::shared_ptr_implementation&lt; Parent, Implementation &gt; Struct Template Reference</code> | 537 |
| 10.20.1 Detailed Description   | 538 |
| 10.20.2 Constructor & Destructor Documentation   | 539 |
| 10.20.2.1 <code>shared_ptr_implementation()</code> [1/3]   | 539 |
| 10.20.2.2 <code>shared_ptr_implementation()</code> [2/3]   | 539 |
| 10.20.2.3 <code>shared_ptr_implementation()</code> [3/3]   | 539 |
| 10.20.3 Member Function Documentation  | 540 |
| 10.20.3.1 <code>hash()</code>  | 540 |
| 10.20.3.2 <code>operator&lt;()</code>  | 541 |
| 10.20.3.3 <code>operator==()</code>  | 541 |
| 10.20.4 Member Data Documentation  | 541 |
| 10.20.4.1 <code>implementation</code>  | 541 |
| 10.21 <code>cl::sycl::detail::singleton&lt; T &gt; Struct Template Reference</code>                                      | 542 |
| 10.21.1 Detailed Description   | 542 |
| 10.21.2 Member Function Documentation  | 542 |
| 10.21.2.1 <code>instance()</code>  | 542 |
| 10.22 <code>cl::sycl::detail::task Struct Reference</code>   | 543 |
| 10.22.1 Detailed Description   | 545 |
| 10.22.2 Constructor & Destructor Documentation   | 546 |
| 10.22.2.1 <code>task()</code>  | 546 |
| 10.22.3 Member Function Documentation  | 546 |
| 10.22.3.1 <code>add_buffer()</code>  | 546 |
| 10.22.3.2 <code>add_postlude()</code>  | 547 |
| 10.22.3.3 <code>add_prelude()</code>   | 547 |
| 10.22.3.4 <code>get_kernel()</code>  | 547 |
| 10.22.3.5 <code>get_queue()</code>   | 548 |
| 10.22.3.6 <code>notify_consumers()</code>  | 548 |
| 10.22.3.7 <code>postlude()</code>  | 549 |
| 10.22.3.8 <code>prelude()</code>   | 549 |

|            |   |            |
|------------|---|------------|
| 10.22.3.9  | <a href="#">release_buffers()</a>   | 550        |
| 10.22.3.10 | <a href="#">schedule()</a>  | 550        |
| 10.22.3.11 | <a href="#">set_kernel()</a>  | 552        |
| 10.22.3.12 | <a href="#">wait()</a>  | 552        |
| 10.22.3.13 | <a href="#">wait_for_producers()</a>  | 553        |
| 10.22.4    | Member Data Documentation   | 553        |
| 10.22.4.1  | <a href="#">buffers_in_use</a>  | 553        |
| 10.22.4.2  | <a href="#">epilogues</a>   | 554        |
| 10.22.4.3  | <a href="#">execution_ended</a>   | 554        |
| 10.22.4.4  | <a href="#">kernel</a>  | 554        |
| 10.22.4.5  | <a href="#">owner_queue</a>   | 554        |
| 10.22.4.6  | <a href="#">producer_tasks</a>  | 554        |
| 10.22.4.7  | <a href="#">prologues</a>   | 555        |
| 10.22.4.8  | <a href="#">ready</a>   | 555        |
| 10.22.4.9  | <a href="#">ready_mutex</a>   | 555        |
| <b>11</b>  | <b>File Documentation</b>   | <b>557</b> |
| 11.1       | <a href="#">include/CL/sycl.hpp File Reference</a>                                    | 557        |
| 11.2       | <a href="#">sycl.hpp</a>  | 558        |
| 11.3       | <a href="#">include/CL/sycl/access.hpp File Reference</a>                             | 559        |
| 11.4       | <a href="#">access.hpp</a>  | 560        |
| 11.5       | <a href="#">include/CL/sycl/accessor.hpp File Reference</a>                           | 561        |
| 11.6       | <a href="#">accessor.hpp</a>  | 562        |
| 11.7       | <a href="#">include/CL/sycl/buffer/detail/accessor.hpp File Reference</a>             | 568        |
| 11.8       | <a href="#">accessor.hpp</a>  | 569        |
| 11.9       | <a href="#">include/CL/sycl/accessor/detail/local_accessor.hpp File Reference</a>     | 575        |
| 11.10      | <a href="#">local_accessor.hpp</a>  | 577        |
| 11.11      | <a href="#">include/CL/sycl/address_space/detail/address_space.hpp File Reference</a> | 581        |
| 11.11.1    | <a href="#">Detailed Description</a>  | 583        |
| 11.12      | <a href="#">address_space.hpp</a>   | 583        |
| 11.13      | <a href="#">include/CL/sycl/address_space.hpp File Reference</a>                      | 588        |

|   |     |
|---|-----|
| 11.13.1 Detailed Description . . . . .  | 590 |
| 11.14address_space.hpp . . . . .  | 590 |
| 11.15include/CL/sycl/allocator.hpp File Reference . . . . .                     | 592 |
| 11.16allocator.hpp . . . . .  | 593 |
| 11.17include/CL/sycl/buffer/detail/buffer.hpp File Reference . . . . .          | 594 |
| 11.18buffer.hpp . . . . .   | 596 |
| 11.19include/CL/sycl/buffer.hpp File Reference . . . . .                        | 601 |
| 11.20buffer.hpp . . . . .   | 602 |
| 11.21include/CL/sycl/buffer/detail/buffer_base.hpp File Reference . . . . .     | 609 |
| 11.22buffer_base.hpp . . . . .  | 610 |
| 11.23include/CL/sycl/buffer/detail/buffer_waiter.hpp File Reference . . . . .   | 614 |
| 11.24buffer_waiter.hpp . . . . .  | 616 |
| 11.25include/CL/sycl/buffer_allocator.hpp File Reference . . . . .              | 617 |
| 11.26buffer_allocator.hpp . . . . .   | 618 |
| 11.27include/CL/sycl/command_group/detail/task.hpp File Reference . . . . .     | 619 |
| 11.28task.hpp . . . . .   | 620 |
| 11.29include/CL/sycl/context/detail/context.hpp File Reference . . . . .        | 623 |
| 11.30context.hpp . . . . .  | 625 |
| 11.31include/CL/sycl/context.hpp File Reference . . . . .                       | 626 |
| 11.32context.hpp . . . . .  | 627 |
| 11.33include/CL/sycl/info/context.hpp File Reference . . . . .                  | 630 |
| 11.34context.hpp . . . . .  | 632 |
| 11.35include/CL/sycl/context/detail/host_context.hpp File Reference . . . . .   | 632 |
| 11.36host_context.hpp . . . . .   | 634 |
| 11.37include/CL/sycl/context/detail/opencl_context.hpp File Reference . . . . . | 635 |
| 11.38opencl_context.hpp . . . . .   | 637 |
| 11.39include/CL/sycl/detail/array_tuple_helpers.hpp File Reference . . . . .    | 638 |
| 11.39.1 Detailed Description . . . . .  | 640 |
| 11.40array_tuple_helpers.hpp . . . . .  | 640 |
| 11.41include/CL/sycl/detail/cache.hpp File Reference . . . . .                  | 642 |

|           |   |     |
|-----------|---|-----|
| 11.42     | <a href="#">cache.hpp</a>   | 643 |
| 11.43     | <a href="#">include/CL/sycl/detail/container_element_aspect.hpp</a> File Reference  | 645 |
| 11.44     | <a href="#">container_element_aspect.hpp</a>  | 645 |
| 11.45     | <a href="#">include/CL/sycl/detail/debug.hpp</a> File Reference                     | 646 |
| 11.45.1   | Macro Definition Documentation  | 648 |
| 11.45.1.1 | <a href="#">TRISYCL_DUMP</a>  | 648 |
| 11.45.1.2 | <a href="#">TRISYCL_DUMP_T</a>  | 648 |
| 11.45.1.3 | <a href="#">TRISYCL_INTERNAL_DUMP</a>   | 648 |
| 11.46     | <a href="#">debug.hpp</a>   | 649 |
| 11.47     | <a href="#">include/CL/sycl/detail/default_classes.hpp</a> File Reference           | 651 |
| 11.48     | <a href="#">default_classes.hpp</a>   | 653 |
| 11.49     | <a href="#">include/CL/sycl/detail/global_config.hpp</a> File Reference             | 654 |
| 11.49.1   | Macro Definition Documentation  | 656 |
| 11.49.1.1 | <a href="#">TRISYCL_WEAK_ATTRIB_PREFIX</a>  | 656 |
| 11.49.1.2 | <a href="#">TRISYCL_WEAK_ATTRIB_SUFFIX</a>  | 656 |
| 11.50     | <a href="#">global_config.hpp</a>   | 656 |
| 11.51     | <a href="#">include/CL/sycl/detail/linear_id.hpp</a> File Reference                 | 657 |
| 11.52     | <a href="#">linear_id.hpp</a>   | 658 |
| 11.53     | <a href="#">include/CL/sycl/detail/shared_ptr_implementation.hpp</a> File Reference | 659 |
| 11.54     | <a href="#">shared_ptr_implementation.hpp</a>                                       | 660 |
| 11.55     | <a href="#">include/CL/sycl/detail/singleton.hpp</a> File Reference                 | 662 |
| 11.56     | <a href="#">singleton.hpp</a>   | 663 |
| 11.57     | <a href="#">include/CL/sycl/detail/small_array.hpp</a> File Reference               | 663 |
| 11.58     | <a href="#">small_array.hpp</a>   | 665 |
| 11.59     | <a href="#">include/CL/sycl/detail/unimplemented.hpp</a> File Reference             | 669 |
| 11.60     | <a href="#">unimplemented.hpp</a>   | 670 |
| 11.61     | <a href="#">include/CL/sycl/device/detail/device.hpp</a> File Reference             | 671 |
| 11.62     | <a href="#">device.hpp</a>  | 672 |
| 11.63     | <a href="#">include/CL/sycl/device.hpp</a> File Reference                           | 673 |
| 11.64     | <a href="#">device.hpp</a>  | 675 |

|   |     |
|---|-----|
| 11.65include/CL/sycl/info/device.hpp File Reference . . . . .                                 | 678 |
| 11.66device.hpp . . . . .   | 681 |
| 11.67include/CL/sycl/device/detail/device_tail.hpp File Reference . . . . .                   | 684 |
| 11.68device_tail.hpp . . . . .  | 684 |
| 11.69include/CL/sycl/device/detail/host_device.hpp File Reference . . . . .                   | 685 |
| 11.70host_device.hpp . . . . .  | 686 |
| 11.71include/CL/sycl/device/detail/ocl_device.hpp File Reference . . . . .                    | 688 |
| 11.72ocl_device.hpp . . . . .   | 689 |
| 11.73include/CL/sycl/device_selector.hpp File Reference . . . . .                             | 691 |
| 11.74device_selector.hpp . . . . .  | 693 |
| 11.75include/CL/sycl/device_selector/detail/device_selector_tail.hpp File Reference . . . . . | 693 |
| 11.76device_selector_tail.hpp . . . . .   | 695 |
| 11.77include/CL/sycl/error_handler.hpp File Reference . . . . .                               | 697 |
| 11.78error_handler.hpp . . . . .  | 698 |
| 11.79include/CL/sycl/event.hpp File Reference . . . . .                                       | 699 |
| 11.80event.hpp . . . . .  | 700 |
| 11.81include/CL/sycl/exception.hpp File Reference . . . . .                                   | 700 |
| 11.82exception.hpp . . . . .  | 703 |
| 11.83include/CL/sycl/group.hpp File Reference . . . . .                                       | 705 |
| 11.84group.hpp . . . . .  | 707 |
| 11.85include/CL/sycl/handler.hpp File Reference . . . . .                                     | 709 |
| 11.85.1 Macro Definition Documentation . . . . .  | 711 |
| 11.85.1.1 TRISYCL_parallel_for_functor_GLOBAL . . . . .                                       | 711 |
| 11.85.1.2 TRISYCL_ParallelForFunctor_GLOBAL_OFFSET . . . . .                                  | 711 |
| 11.85.1.3 TRISYCL_ParallelForKernel_RANGE . . . . .   | 712 |
| 11.86handler.hpp . . . . .  | 712 |
| 11.87include/CL/sycl/handler_event.hpp File Reference . . . . .                               | 718 |
| 11.88handler_event.hpp . . . . .  | 718 |
| 11.89include/CL/sycl/id.hpp File Reference . . . . .  | 719 |
| 11.90id.hpp . . . . .   | 721 |

|  |     |
|--|-----|
| 11.91include/CL/sycl/image.hpp File Reference . . . . .                        | 722 |
| 11.91.1 Detailed Description . . . . .   | 722 |
| 11.92image.hpp . . . . .   | 723 |
| 11.93include/CL/sycl/info/param_traits.hpp File Reference . . . . .            | 724 |
| 11.93.1 Macro Definition Documentation . . . . .                               | 725 |
| 11.93.1.1 TRISYCL_INFO_PARAM_TRAITS . . . . .                                  | 725 |
| 11.93.1.2 TRISYCL_INFO_PARAM_TRAITS_ANY_T . . . . .                            | 725 |
| 11.94param_traits.hpp . . . . .  | 726 |
| 11.95include/CL/sycl/info/platform.hpp File Reference . . . . .                | 726 |
| 11.96platform.hpp . . . . .  | 729 |
| 11.97include/CL/sycl/platform/detail/platform.hpp File Reference . . . . .     | 730 |
| 11.98platform.hpp . . . . .  | 731 |
| 11.99include/CL/sycl/platform.hpp File Reference . . . . .                     | 733 |
| 11.100platform.hpp . . . . .   | 734 |
| 11.101include/CL/sycl/item.hpp File Reference . . . . .                        | 736 |
| 11.102item.hpp . . . . .   | 738 |
| 11.103include/CL/sycl/kernel/detail/kernel.hpp File Reference . . . . .        | 739 |
| 11.103.1 Macro Definition Documentation . . . . .                              | 741 |
| 11.103.1.1 TRISYCL_ParallelForKernel_RANGE . . . . .                           | 741 |
| 11.104kernel.hpp . . . . .   | 741 |
| 11.105include/CL/sycl/kernel.hpp File Reference . . . . .                      | 742 |
| 11.106kernel.hpp . . . . .   | 744 |
| 11.107include/CL/sycl/kernel/detail/opencl_kernel.hpp File Reference . . . . . | 745 |
| 11.107.1 Macro Definition Documentation . . . . .                              | 747 |
| 11.107.1.1 TRISYCL_ParallelForKernel_RANGE . . . . .                           | 747 |
| 11.108opencl_kernel.hpp . . . . .  | 748 |
| 11.109include/CL/sycl/math.hpp File Reference . . . . .                        | 749 |
| 11.109.1 Detailed Description . . . . .  | 751 |
| 11.109.2 Macro Definition Documentation . . . . .                              | 751 |
| 11.109.2.1 TRISYCL_MATH_WRAP . . . . .   | 751 |

|             |   |     |
|-------------|---|-----|
| 11.109.2.2  | TRISYCL_MATH_WRAP2 . . . . .  | 752 |
| 11.109.2.3  | TRISYCL_MATH_WRAP2s . . . . .   | 752 |
| 11.109.2.4  | TRISYCL_MATH_WRAP3 . . . . .  | 752 |
| 11.109.2.5  | TRISYCL_MATH_WRAP3s . . . . .   | 753 |
| 11.109.2.6  | TRISYCL_MATH_WRAP3ss . . . . .  | 753 |
| 11.110      | math.hpp . . . . .  | 753 |
| 11.111      | include/CL/sycl/nd_item.hpp File Reference . . . . .                        | 755 |
| 11.112      | nd_item.hpp . . . . .   | 757 |
| 11.113      | include/CL/sycl/nd_range.hpp File Reference . . . . .                       | 759 |
| 11.114      | nd_range.hpp . . . . .  | 761 |
| 11.115      | include/CL/sycl/opencl_types.hpp File Reference . . . . .                   | 762 |
| 11.115.1    | Detailed Description . . . . .  | 764 |
| 11.115.2    | Macro Definition Documentation . . . . .                                    | 764 |
| 11.115.2.1  | TRISYCL_BOOST_COMPUTE_NAME . . . . .  | 764 |
| 11.115.2.2  | TRISYCL_DECLARE_CL_TYPES . . . . .  | 764 |
| 11.115.2.3  | TRISYCL_DEFINE_TYPES . . . . .  | 764 |
| 11.115.2.4  | TRISYCL_H_DEFINE_TYPE . . . . .   | 765 |
| 11.115.2.5  | TRISYCL_IS_WRAPPER_TRAIT . . . . .  | 765 |
| 11.115.2.6  | TRISYCL_SCALAR_TYPES . . . . .  | 765 |
| 11.115.2.7  | TRISYCL_SIZED_NAME . . . . .  | 766 |
| 11.115.2.8  | TRISYCL_TYPE_ACTUAL_NAME . . . . .  | 766 |
| 11.115.2.9  | TRISYCL_TYPE_CL_NAME . . . . .  | 766 |
| 11.115.2.10 | TRISYCL_TYPE_NAME . . . . .   | 766 |
| 11.115.2.11 | TRISYCL_TYPEDEF_TYPE . . . . .  | 766 |
| 11.115.2.12 | TRISYCL_WRAPPER_CLASS_2 . . . . .   | 767 |
| 11.115.2.13 | TRISYCL_WRAPPER_CLASS_3 . . . . .   | 767 |
| 11.115.2.14 | TRISYCL_WRAPPER_CLASS_4 . . . . .   | 768 |
| 11.116      | opencl_types.hpp . . . . .  | 768 |
| 11.117      | include/CL/sycl/parallelism/detail/parallelism.hpp File Reference . . . . . | 770 |
| 11.117.1    | Detailed Description . . . . .  | 772 |

|  |     |
|--|-----|
| 11.11parallelism.hpp . . . . .   | 773 |
| 11.11include/CL/sycl/parallelism.hpp File Reference . . . . .                              | 777 |
| 11.119.Detailed Description . . . . .  | 778 |
| 11.12parallelism.hpp . . . . .   | 779 |
| 11.12include/CL/sycl/pipe/detail/pipe.hpp File Reference . . . . .                         | 779 |
| 11.12pipe.hpp . . . . .  | 781 |
| 11.12include/CL/sycl/pipe.hpp File Reference . . . . .                                     | 786 |
| 11.12pipe.hpp . . . . .  | 787 |
| 11.12include/CL/sycl/pipe/detail/pipe_accessor.hpp File Reference . . . . .                | 789 |
| 11.12pipe_accessor.hpp . . . . .   | 790 |
| 11.12include/CL/sycl/pipe_reservation/detail/pipe_reservation.hpp File Reference . . . . . | 793 |
| 11.12pipe_reservation.hpp . . . . .  | 795 |
| 11.12include/CL/sycl/pipe_reservation.hpp File Reference . . . . .                         | 797 |
| 11.13pipe_reservation.hpp . . . . .  | 798 |
| 11.13include/CL/sycl/platform/detail/host_platform.hpp File Reference . . . . .            | 800 |
| 11.13host_platform.hpp . . . . .   | 802 |
| 11.13include/CL/sycl/platform/detail/host_platform_tail.hpp File Reference . . . . .       | 804 |
| 11.13host_platform_tail.hpp . . . . .  | 804 |
| 11.13include/CL/sycl/platform/detail/opencl_platform.hpp File Reference . . . . .          | 805 |
| 11.13opencl_platform.hpp . . . . .   | 807 |
| 11.13include/CL/sycl/platform/detail/opencl_platform_tail.hpp File Reference . . . . .     | 809 |
| 11.13opencl_platform_tail.hpp . . . . .  | 809 |
| 11.13include/CL/sycl/queue/detail/host_queue.hpp File Reference . . . . .                  | 810 |
| 11.14host_queue.hpp . . . . .  | 811 |
| 11.14include/CL/sycl/queue/detail/opencl_queue.hpp File Reference . . . . .                | 812 |
| 11.14opencl_queue.hpp . . . . .  | 814 |
| 11.14include/CL/sycl/queue/detail/queue.hpp File Reference . . . . .                       | 815 |
| 11.14queue.hpp . . . . .   | 816 |
| 11.14include/CL/sycl/queue.hpp File Reference . . . . .                                    | 818 |
| 11.14queue.hpp . . . . .   | 820 |
| 11.14include/CL/sycl/range.hpp File Reference . . . . .                                    | 824 |
| 11.14range.hpp . . . . .   | 826 |
| 11.14include/CL/sycl/static_pipe.hpp File Reference . . . . .                              | 827 |
| 11.15static_pipe.hpp . . . . .   | 828 |
| 11.15include/CL/sycl/vec.hpp File Reference . . . . .                                      | 829 |
| 11.151.Detailed Description . . . . .  | 831 |
| 11.15vec.hpp . . . . .   | 831 |



# Chapter 1

## Main Page

This is the main 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/triSYCL/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 2015–2017 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, 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 >::get_count () const`

Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

### Member `cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::get_pointer () const`

Should it be named `data()` instead?

### Member `cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::get_range () const`

Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

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

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

Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

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

Add in the specification

**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[] (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 >`**

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

Update the specification to have a non-const allocator for const buffer? Or do we rely on `rebind_alloc<T>`. But does this work with `astate-full` allocator?

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 (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 >::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 (const T *host_data, const range< Dimensions > &r, Allocator allocator={})`**

Actually this is redundant.

**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 >::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`** (`shared_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, access::target::local >`**

Use the `access::mode`

**Member `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::accessor`** (`const range< Dimensions > &allocation_size, handler &command_group_handler`)

fix the specification to rename `target` that shadows template param

**Member `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::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, access::target::local >::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, access::target::local >::get_count () const`**

Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

**Member `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::get_range () const`**

Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

**Member `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::get_size () const`**

Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

**Member `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::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, access::target::local >::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, access::target::local >::iterator`**

Add iterators to accessors in the specification

**Member `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator* ()`**

Add in the specification

**Member `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::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, access::target::local >::operator[] (nd_item< dimensionality > index)`**

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

**Member `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::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, access::target::local >::value_type`**

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

**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 >::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_count () const`**

Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

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

Implement the various pointer address spaces

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

Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

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

Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

**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 >::register_accessor ()`**

Double-check with the C++ committee on this issue.

**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 >::alloc`**

Implement user-provided allocator

**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 >::call_update_buffer_state (cl::sycl::context ctx, access↵  
::mode mode, size_t size, DataType *data, std::enable_if_t<!std::is_const< BaseType > ::value > * = 0)`**

Use `if constexpr` when it is available with C++17

**Member `cl::sycl::detail::buffer< T, Dimensions >::get_destructor_future ()`**

Make the function private again

**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 >::~~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::buffer_base ()`**

Use lazy allocation for the context tracking set

**Member `cl::sycl::detail::context::get_devices () const = 0`**

virtual cannot be templated template `<info::context Param> typename info::param_traits<info::context, Param>::type get_info() const = 0;`

**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;`

**Member `cl::sycl::detail::host_context::get_devices () const override`**

To be implemented

**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

**Member `cl::sycl::detail::opencl_context::get_devices()` const override**

To be implemented

**Member `cl::sycl::detail::opencl_context::get_platform()` const override**

To be implemented

**Member `cl::sycl::detail::opencl_device::has_extension` (const string\_class &extension) const override**

To be implemented

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

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

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

**Member `cl::sycl::detail::opencl_kernel::single_task` (std::shared\_ptr< detail::task > task, std::shared\_ptr< detail::queue > q) override**

Remove either task or q

**Member `cl::sycl::detail::opencl_queue::instance` (const cl::sycl::device &d)**

Check with SYCL committee what is the expected behaviour here about the context. Is this a new context everytime, or always the same for a given device?

**Member `cl::sycl::detail::parallel_for` (nd\_range< Dimensions > r, ParallelForFunctor f)**

Add an OpenMP implementation

Deal with incomplete work-groups

Implement with `parallel_for_workgroup()/parallel_for_workitem()`

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

Better type the functor

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

provide a && version

**Member `cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::pipe_accessor` (const std::shared\_ptr< detail::pipe< T >> &p, handler &command\_group\_handler)**

Use pipe\_exception instead

**Member `cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::write` (const value\_type &value) const**

provide a && version

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

Throw exception instead

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

Add to the specification that for simplicity a reservation can be 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::detail::task::schedule (std::function< void(void)> f)`

This is an issue if there is an exception in the kernel

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

Make it non-explicit in the specification?

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

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

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

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

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

Remove this from specification

**Class** `cl::sycl::device_type_selector`

To be added to the specification

**Class** `cl::sycl::device_typename_selector< DeviceType >`

To be added to the specification

**Member** `cl::sycl::error_handler::default_handler`

add this concept to the specification?

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

Add "virtual void" to the specification

**Class** `cl::sycl::exception_list`

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

**Member** `cl::sycl::exception_ptr`

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

**Member** `cl::sycl::group< Dimensions >::dimensionality`

add this Boost::multi\_array or STL concept to the specification?

**Member** `cl::sycl::group< Dimensions >::get_group_range () const`

Fix this comment and the specification

**Member** `cl::sycl::group< Dimensions >::get_local_range () const`

Add to the specification

**Member** `cl::sycl::group< Dimensions >::get_local_range (int dimension) const`

Add to the specification

**Member** `cl::sycl::group< Dimensions >::get_nd_range () const`

Also provide this access to the current `nd_range`

**Member `cl::sycl::group< Dimensions >::get_offset` (int dimension) const**

Add to the specification

**Member `cl::sycl::group< Dimensions >::get_offset` () const**

Add to the specification

**Member `cl::sycl::group< Dimensions >::group` (const id< Dimensions > &i, const nd\_range< Dimensions > &ndr)**

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

**Member `cl::sycl::group< Dimensions >::group` (const nd\_range< Dimensions > &ndr)**

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

**Member `cl::sycl::group< Dimensions >::group` ()=default**

Make most of them protected, reserved to implementation

**Member `cl::sycl::group< Dimensions >::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< Dimensions >::parallel_for_work_item` (std::function< void(nd\_item< dimensionality >)> f) const**

Add this method in the specification

**Member `cl::sycl::group< Dimensions >::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_args` (Ts &&... args)**

Update the specification to add this function according to [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15978](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15978) proposal

**Member `cl::sycl::handler::single_task` (kernel sycl\_kernel)**

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

**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< Dimensions >::dimensionality`**

add this Boost::multi\_array or STL concept to the specification?

**Member `cl::sycl::item< Dimensions >::item ()=default`**

Make most of them protected, reserved to implementation

**Member `cl::sycl::item< Dimensions >::set (id< Dimensions > 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::map_allocator`**

: implement and clarify the specification. It looks like it is not really an allocator according the current spec

**Member `cl::sycl::nd_item< Dimensions >::dimensionality`**

add this Boost::multi\_array or STL concept to the specification?

**Member `cl::sycl::nd_item< Dimensions >::get_item () const`**

Add to the specification

**Member `cl::sycl::nd_item< Dimensions >::nd_item (id< Dimensions > global_index, nd_range< Dimensions > ndr)`**

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

**Member `cl::sycl::nd_item< Dimensions >::nd_item (nd_range< Dimensions > 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.

**Member `cl::sycl::nd_item< Dimensions >::nd_item ()=default`**

Make most of them protected, reserved to implementation

**Class `cl::sycl::nd_range< Dimensions >`**

add copy constructors in the specification

**Member `cl::sycl::nd_range< Dimensions >::dimensionality`**

add this Boost::multi\_array or STL concept to the specification?

**Member `cl::sycl::nd_range< Dimensions >::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::queue ()`**

Check with the specification if it is the host queue or the one related to the default device selector.

**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< Dimensions >`**

use `std::size_t Dimensions` instead of `int Dimensions` 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< Dimensions >::get_count () const`**

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`

**Module `execution`**

The implementation is quite minimal for now. :-)

**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

Remove either task or q



## Chapter 3

# Module Index

### 3.1 Modules

Here is a list of all modules:

|  |     |
|--|-----|
| Data access and storage in SYCL . . . . .          | 29  |
| Dealing with OpenCL address spaces . . . . .       | 203 |
| Platforms, contexts, devices and queues . . . . .  | 232 |
| Helpers to do array and tuple conversion . . . . . | 337 |
| Some helpers for the implementation . . . . .      | 342 |
| Debugging and tracing support . . . . .            | 363 |
| Manage default configuration and types . . . . .   | 366 |
| Error handling . . . . .                           | 368 |
| Expressing parallelism through kernels . . . . .   | 395 |
| Vector types in SYCL . . . . .                     | 455 |





## Chapter 4

# Namespace Index

### 4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

|                                   |   |     |
|-----------------------------------|---|-----|
| <a href="#">cl</a>                | The vector type to be used as SYCL vector . . . . . | 461 |
| <a href="#">cl::sycl</a>          | . . . . .   | 461 |
| <a href="#">cl::sycl::access</a>  | Describe the type of access by kernels . . . . .    | 469 |
| <a href="#">cl::sycl::detail</a>  | . . . . .   | 471 |
| <a href="#">cl::sycl::info</a>    | . . . . .   | 475 |
| <a href="#">cl::sycl::trisycl</a> | . . . . .   | 478 |
| <a href="#">std</a>               | . . . . .   | 478 |



## 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 > . . . . .  | 203 |
| cl::sycl::detail::address_space_object< T, AS > . . . . .                                      | 203 |
| cl::sycl::detail::address_space_variable< T, AS > . . . . .                                    | 203 |
| cl::sycl::detail::address_space_array< T, AS > . . . . .                                       | 203 |
| cl::sycl::detail::address_space_fundamental< T, AS > . . . . .                                 | 203 |
| cl::sycl::detail::address_space_ptr< T, AS > . . . . .   | 203 |
| array  |     |
| cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > . . . . .   | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 1 > . . . . .                             | 342 |
| cl::sycl::detail::small_array_123< BasicType, FinalType, 1 > . . . . .                         | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 2 > . . . . .                             | 342 |
| cl::sycl::detail::small_array_123< BasicType, FinalType, 2 > . . . . .                         | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 3 > . . . . .                             | 342 |
| cl::sycl::detail::small_array_123< BasicType, FinalType, 3 > . . . . .                         | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, Dims > . . . . .                          | 342 |
| cl::sycl::detail::small_array_123< BasicType, FinalType, Dims > . . . . .                      | 342 |
| cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > . . . . . | 342 |
| cl::sycl::vec< DataType, NumElements > . . . . .   | 455 |
| cl::sycl::detail::small_array< std::size_t, id< Dimensions >, Dims > . . . . .                 | 342 |
| cl::sycl::detail::small_array_123< std::size_t, id< Dimensions >, Dimensions > . . . . .       | 342 |
| cl::sycl::id< Dimensions > . . . . .   | 395 |
| cl::sycl::id< dimensionality > . . . . .   | 395 |
| cl::sycl::detail::small_array< std::size_t, range< Dimensions >, Dims > . . . . .              | 342 |
| cl::sycl::detail::small_array_123< std::size_t, range< Dimensions >, Dimensions > . . . . .    | 342 |
| cl::sycl::range< Dimensions > . . . . .  | 395 |
| cl::sycl::range< dimensionality > . . . . .  | 395 |
| bitwise  |     |
| cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > . . . . .   | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 1 > . . . . .                             | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 2 > . . . . .                             | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 3 > . . . . .                             | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, Dims > . . . . .                          | 342 |
| cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > . . . . . | 342 |
| cl::sycl::detail::small_array< std::size_t, id< Dimensions >, Dims > . . . . .                 | 342 |

|  |     |
|--|-----|
| cl::sycl::detail::small_array< std::size_t, range< Dimensions >, Dims > . . . . .                | 342 |
| cl::sycl::detail::cache< Key, Value > . . . . .  | 479 |
| cl::sycl::detail::cache< cl_command_queue, cl::sycl::detail::opencl_queue > . . . . .            | 479 |
| cl::sycl::detail::cache< cl_context, cl::sycl::detail::opencl_context > . . . . .                | 479 |
| cl::sycl::detail::cache< cl_device_id, cl::sycl::detail::opencl_device > . . . . .               | 479 |
| cl::sycl::detail::cache< cl_kernel, cl::sycl::detail::opencl_kernel > . . . . .                  | 479 |
| cl::sycl::detail::cache< cl_platform_id, cl::sycl::detail::opencl_platform > . . . . .           | 479 |
| cl::sycl::detail::container_element_aspect< T > . . . . .  | 342 |
| cl::sycl::detail::container_element_aspect< DataType > . . . . .                                 | 342 |
| cl::sycl::accessor< DataType, Dimensions, AccessMode, Target > . . . . .                         | 29  |
| cl::sycl::detail::context . . . . .  | 232 |
| cl::sycl::detail::host_context . . . . .   | 232 |
| cl::sycl::detail::opencl_context . . . . .   | 502 |
| cl::sycl::detail::debug< T > . . . . .   | 363 |
| cl::sycl::detail::debug< accessor< T, Dimensions, Mode, access::target::local > > . . . . .      | 363 |
| cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local > . . . . .               | 29  |
| cl::sycl::detail::debug< accessor< T, Dimensions, Mode, Target > > . . . . .                     | 363 |
| cl::sycl::detail::accessor< T, Dimensions, Mode, Target > . . . . .                              | 29  |
| cl::sycl::detail::debug< buffer< T, Dimensions > > . . . . .                                     | 363 |
| cl::sycl::detail::buffer< T, Dimensions > . . . . .  | 29  |
| cl::sycl::detail::debug< buffer< T, Dimensions, Allocator > > . . . . .                          | 363 |
| cl::sycl::buffer< T, Dimensions, Allocator > . . . . .   | 29  |
| cl::sycl::detail::debug< buffer_waiter< T, Dimensions, Allocator > > . . . . .                   | 363 |
| cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator > . . . . .                            | 29  |
| cl::sycl::detail::debug< detail::kernel > . . . . .  | 363 |
| cl::sycl::detail::kernel . . . . .   | 232 |
| cl::sycl::detail::opencl_kernel . . . . .  | 517 |
| cl::sycl::detail::debug< detail::pipe_accessor< DataType, AccessMode, Target > > . . . . .       | 363 |
| cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe > . . . . . | 29  |
| cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe > . . . . .           | 29  |
| cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::pipe > . . . . .          | 29  |
| cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe > . . . . .                    | 29  |
| cl::sycl::detail::debug< detail::pipe_accessor< T, AccessMode, Target > > . . . . .              | 363 |
| cl::sycl::detail::pipe_accessor< T, AccessMode, Target > . . . . .                               | 29  |
| cl::sycl::detail::debug< detail::pipe_reservation< PipeAccessor > > . . . . .                    | 363 |
| cl::sycl::detail::pipe_reservation< PipeAccessor > . . . . .                                     | 29  |
| cl::sycl::detail::debug< detail::queue > . . . . .   | 363 |
| cl::sycl::detail::queue . . . . .  | 530 |
| cl::sycl::detail::host_queue . . . . .   | 498 |
| cl::sycl::detail::opencl_queue . . . . .   | 523 |
| cl::sycl::detail::debug< handler > . . . . .   | 363 |
| cl::sycl::handler . . . . .  | 232 |
| cl::sycl::detail::debug< host_queue > . . . . .  | 363 |
| cl::sycl::detail::host_queue . . . . .   | 498 |
| cl::sycl::detail::debug< opencl_kernel > . . . . .   | 363 |
| cl::sycl::detail::opencl_kernel . . . . .  | 517 |
| cl::sycl::detail::debug< opencl_queue > . . . . .  | 363 |
| cl::sycl::detail::opencl_queue . . . . .   | 523 |
| cl::sycl::detail::debug< pipe< T > > . . . . .   | 363 |
| cl::sycl::detail::pipe< T > . . . . .  | 29  |
| cl::sycl::pipe< T > . . . . .  | 29  |
| cl::sycl::detail::debug< pipe< value_type > > . . . . .  | 363 |
| cl::sycl::detail::pipe< value_type > . . . . .   | 29  |

|  |     |
|--|-----|
| cl::sycl::detail::debug< queue > . . . . .   | 363 |
| cl::sycl::queue . . . . .  | 232 |
| cl::sycl::detail::debug< static_pipe< T, Capacity > > . . . . .                                | 363 |
| cl::sycl::static_pipe< T, Capacity > . . . . .   | 29  |
| cl::sycl::detail::debug< task > . . . . .  | 363 |
| cl::sycl::detail::task . . . . .   | 543 |
| cl::sycl::detail::device . . . . .   | 232 |
| cl::sycl::detail::host_device . . . . .  | 493 |
| cl::sycl::detail::opencl_device . . . . .  | 510 |
| cl::sycl::device_selector . . . . .  | 232 |
| cl::sycl::device_type_selector . . . . .   | 232 |
| cl::sycl::device_type_name_selector< DeviceType > . . . . .                                    | 232 |
| cl::sycl::detail::display_vector< T > . . . . .  | 363 |
| cl::sycl::detail::display_vector< FinalType > . . . . .  | 363 |
| cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > . . . . .   | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 1 > . . . . .                             | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 2 > . . . . .                             | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 3 > . . . . .                             | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, Dims > . . . . .                          | 342 |
| cl::sycl::detail::display_vector< id< Dimensions > > . . . . .                                 | 363 |
| cl::sycl::detail::small_array< std::size_t, id< Dimensions >, Dims > . . . . .                 | 342 |
| cl::sycl::detail::display_vector< range< Dimensions > > . . . . .                              | 363 |
| cl::sycl::detail::small_array< std::size_t, range< Dimensions >, Dims > . . . . .              | 342 |
| cl::sycl::detail::display_vector< vec< DataType, NumElements > > . . . . .                     | 363 |
| cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > . . . . . | 342 |
| enable_shared_from_this . . . . .  |     |
| cl::sycl::detail::accessor< T, Dimensions, Mode, Target > . . . . .                            | 29  |
| cl::sycl::detail::buffer_base . . . . .  | 29  |
| cl::sycl::detail::buffer< T, Dimensions > . . . . .  | 29  |
| cl::sycl::detail::task . . . . .   | 543 |
| cl::sycl::error_handler . . . . .  | 368 |
| cl::sycl::trisycl::default_error_handler . . . . .   | 484 |
| euclidean_ring_operators . . . . .   |     |
| cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > . . . . .   | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 1 > . . . . .                             | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 2 > . . . . .                             | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 3 > . . . . .                             | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, Dims > . . . . .                          | 342 |
| cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > . . . . . | 342 |
| cl::sycl::detail::small_array< std::size_t, id< Dimensions >, Dims > . . . . .                 | 342 |
| cl::sycl::detail::small_array< std::size_t, range< Dimensions >, Dims > . . . . .              | 342 |
| cl::sycl::event . . . . .  | 485 |
| cl::sycl::exception . . . . .  | 368 |
| cl::sycl::async_exception . . . . .  | 368 |
| cl::sycl::cl_exception . . . . .   | 368 |
| cl::sycl::device_error . . . . .   | 368 |
| cl::sycl::compile_program_error . . . . .  | 368 |
| cl::sycl::feature_not_supported . . . . .  | 368 |
| cl::sycl::invalid_object_error . . . . .   | 368 |
| cl::sycl::link_program_error . . . . .   | 368 |
| cl::sycl::memory_allocation_error . . . . .  | 368 |
| cl::sycl::platform_error . . . . .   | 368 |
| cl::sycl::profiling_error . . . . .  | 368 |
| cl::sycl::runtime_error . . . . .  | 368 |
| cl::sycl::accessor_error . . . . .   | 368 |

|  |     |
|--|-----|
| cl::sycl::event_error . . . . .  | 368 |
| cl::sycl::invalid_parameter_error . . . . .  | 368 |
| cl::sycl::kernel_error . . . . .   | 368 |
| cl::sycl::nd_range_error . . . . .   | 368 |
| cl::sycl::non_cl_error . . . . .   | 368 |
| cl::sycl::pipe_error . . . . .   | 368 |
| cl::sycl::detail::expand_to_vector< V, Tuple, expansion > . . . . .  | 337 |
| cl::sycl::detail::expand_to_vector< V, Tuple, true > . . . . .   | 337 |
| false_type   |     |
| cl::sycl::is_wrapper< T > . . . . .  | 501 |
| cl::sycl::group< Dimensions > . . . . .  | 395 |
| handler_event . . . . .  | 486 |
| std::hash< cl::sycl::buffer< T, Dimensions, Allocator > > . . . . .  | 486 |
| std::hash< cl::sycl::context > . . . . .   | 487 |
| std::hash< cl::sycl::device > . . . . .  | 488 |
| std::hash< cl::sycl::kernel > . . . . .  | 489 |
| std::hash< cl::sycl::platform > . . . . .  | 490 |
| std::hash< cl::sycl::queue > . . . . .   | 491 |
| cl::sycl::image< Dimensions > . . . . .  | 29  |
| cl::sycl::item< Dimensions > . . . . .   | 395 |
| cl::sycl::nd_item< Dimensions > . . . . .  | 395 |
| cl::sycl::nd_range< Dimensions > . . . . .   | 395 |
| cl::sycl::detail::ocl_type< T, AS > . . . . .  | 203 |
| cl::sycl::detail::ocl_type< T, constant_address_space > . . . . .  | 203 |
| cl::sycl::detail::ocl_type< T, generic_address_space > . . . . .   | 203 |
| cl::sycl::detail::ocl_type< T, global_address_space > . . . . .  | 203 |
| cl::sycl::detail::ocl_type< T, local_address_space > . . . . .   | 203 |
| cl::sycl::detail::ocl_type< T, private_address_space > . . . . .   | 203 |
| cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id > . . . . .   | 395 |
| cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id > . . . . .   | 395 |
| cl::sycl::detail::parallel_OpenMP_for_iterate< level, Range, ParallelForFunctor, Id > . . . . .  | 395 |
| cl::sycl::info::param_traits< T, Param > . . . . .   | 529 |
| cl::sycl::pipe_reservation< PipeAccessor > . . . . .   | 29  |
| cl::sycl::detail::platform . . . . .   | 232 |
| cl::sycl::detail::host_platform . . . . .  | 232 |
| cl::sycl::detail::opencl_platform . . . . .  | 232 |
| cl::sycl::detail::reserve_id< T > . . . . .  | 29  |
| shiftable  |     |
| cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > . . . . .   | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 1 > . . . . .   | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 2 > . . . . .   | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, 3 > . . . . .   | 342 |
| cl::sycl::detail::small_array< BasicType, FinalType, Dims > . . . . .  | 342 |
| cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > . . . . .   | 342 |
| cl::sycl::detail::small_array< std::size_t, id< Dimensions >, Dims > . . . . .   | 342 |
| cl::sycl::detail::small_array< std::size_t, range< Dimensions >, Dims > . . . . .  | 342 |
| cl::sycl::detail::singleton< T > . . . . .   | 542 |
| cl::sycl::detail::singleton< host_context > . . . . .  | 542 |
| cl::sycl::detail::host_context . . . . .   | 232 |
| cl::sycl::detail::singleton< host_device > . . . . .   | 542 |
| cl::sycl::detail::host_device . . . . .  | 493 |
| cl::sycl::detail::singleton< host_platform > . . . . .   | 542 |
| cl::sycl::detail::host_platform . . . . .  | 232 |
| totally_ordered  |     |
| cl::sycl::detail::shared_ptr_implementation< Parent, Implementation > . . . . .  | 537 |
| cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target<br>>, detail::accessor< DataType, Dimensions, AccessMode, Target > > . . . . . | 537 |

|   |     |
|---|-----|
| cl::sycl::accessor< DataType, Dimensions, AccessMode, Target > . . . . .  | 29  |
| cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_↵<br>waiter< T, Dimensions, Allocator > > . . . . . | 537 |
| cl::sycl::buffer< T, Dimensions, Allocator > . . . . .  | 29  |
| cl::sycl::detail::shared_ptr_implementation< buffer_waiter< T, Dimensions, Allocator >, detail::↵<br>:buffer< T, Dimensions > > . . . . .           | 537 |
| cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator > . . . . .   | 29  |
| cl::sycl::detail::shared_ptr_implementation< context, detail::context > . . . . .   | 537 |
| cl::sycl::context . . . . .   | 232 |
| cl::sycl::detail::shared_ptr_implementation< device, detail::device > . . . . .   | 537 |
| cl::sycl::device . . . . .  | 232 |
| cl::sycl::detail::shared_ptr_implementation< kernel, detail::kernel > . . . . .   | 537 |
| cl::sycl::kernel . . . . .  | 232 |
| cl::sycl::detail::shared_ptr_implementation< pipe< T >, detail::pipe< T > > . . . . .   | 537 |
| cl::sycl::pipe< T > . . . . .   | 29  |
| cl::sycl::detail::shared_ptr_implementation< platform, detail::platform > . . . . .   | 537 |
| cl::sycl::platform . . . . .  | 232 |
| cl::sycl::detail::shared_ptr_implementation< queue, detail::queue > . . . . .   | 537 |
| cl::sycl::queue . . . . .   | 232 |
| cl::sycl::detail::shared_ptr_implementation< static_pipe< T, Capacity >, detail::pipe< T > > . . . . .  | 537 |
| cl::sycl::static_pipe< T, Capacity > . . . . .  | 29  |
| type  |     |
| cl::sycl::detail::address_space_object< T, AS > . . . . .   | 203 |
| vector  |     |
| cl::sycl::exception_list . . . . .  | 368 |
| bool . . . . .  | ??  |
| shared_ptr< cl::sycl::detail::pipe< DataType > > . . . . .  | ??  |
| 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::context > . . . . .   | ??  |
| 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:

|  |     |
|--|-----|
| <a href="#">cl::sycl::detail::cache&lt; Key, Value &gt;</a>  |     |
| A simple thread safe cache mechanism to cache std::shared_ptr of values indexed by keys  | 479 |
| <a href="#">cl::sycl::trisycl::default_error_handler</a>   | 484 |
| <a href="#">cl::sycl::event</a>  | 485 |
| <a href="#">handler_event</a>  |     |
| Handler event  | 486 |
| <a href="#">std::hash&lt; cl::sycl::buffer&lt; T, Dimensions, Allocator &gt; &gt;</a>  | 486 |
| <a href="#">std::hash&lt; cl::sycl::context &gt;</a>   | 487 |
| <a href="#">std::hash&lt; cl::sycl::device &gt;</a>  | 488 |
| <a href="#">std::hash&lt; cl::sycl::kernel &gt;</a>  | 489 |
| <a href="#">std::hash&lt; cl::sycl::platform &gt;</a>  | 490 |
| <a href="#">std::hash&lt; cl::sycl::queue &gt;</a>   | 491 |
| <a href="#">cl::sycl::detail::host_device</a>  |     |
| SYCL host device   | 493 |
| <a href="#">cl::sycl::detail::host_queue</a>   |     |
| Some implementation details about the SYCL queue   | 498 |
| <a href="#">cl::sycl::is_wrapper&lt; T &gt;</a>  | 501 |
| <a href="#">cl::sycl::detail::opencl_context</a>   |     |
| SYCL OpenCL context  | 502 |
| <a href="#">cl::sycl::detail::opencl_device</a>  |     |
| SYCL OpenCL device   | 510 |
| <a href="#">cl::sycl::detail::opencl_kernel</a>  |     |
| An abstraction of the OpenCL kernel  | 517 |
| <a href="#">cl::sycl::detail::opencl_queue</a>   |     |
| Some implementation details about the SYCL queue   | 523 |
| <a href="#">cl::sycl::info::param_traits&lt; T, Param &gt;</a>   |     |
| Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value)        | 529 |
| <a href="#">cl::sycl::detail::queue</a>  |     |
| Some implementation details about the SYCL queue   | 530 |
| <a href="#">cl::sycl::detail::shared_ptr_implementation&lt; Parent, Implementation &gt;</a>                                      |     |
| Provide an implementation as shared_ptr with total ordering and hashing to be used with algorithms and in (un)ordered containers | 537 |
| <a href="#">cl::sycl::detail::singleton&lt; T &gt;</a>   |     |
| Provide a singleton factory  | 542 |
| <a href="#">cl::sycl::detail::task</a>   |     |
| The abstraction to represent SYCL tasks executing inside command_group   | 543 |



## Chapter 7

# File Index

### 7.1 File List

Here is a list of all files with brief descriptions:

|  |     |
|--|-----|
| include/CL/sycl.hpp  | 557 |
| include/CL/sycl/access.hpp   | 559 |
| include/CL/sycl/accessor.hpp   | 561 |
| include/CL/sycl/address_space.hpp  |     |
| Implement OpenCL address spaces in SYCL with C++-style                                 | 588 |
| include/CL/sycl/allocator.hpp  | 592 |
| include/CL/sycl/buffer.hpp   | 601 |
| include/CL/sycl/buffer_allocator.hpp   | 617 |
| include/CL/sycl/context.hpp  | 626 |
| include/CL/sycl/device.hpp   | 673 |
| include/CL/sycl/device_selector.hpp  | 691 |
| include/CL/sycl/error_handler.hpp  | 697 |
| include/CL/sycl/event.hpp  | 699 |
| include/CL/sycl/exception.hpp  | 700 |
| include/CL/sycl/group.hpp  | 705 |
| include/CL/sycl/handler.hpp  | 709 |
| include/CL/sycl/handler_event.hpp  | 718 |
| include/CL/sycl/id.hpp   | 719 |
| include/CL/sycl/image.hpp  |     |
| OpenCL SYCL image class  | 722 |
| include/CL/sycl/item.hpp   | 736 |
| include/CL/sycl/kernel.hpp   | 742 |
| include/CL/sycl/math.hpp   |     |
| Implement a wrapper around OpenCL math operations Joan.Thibault AT ens-rennes POINT fr |     |
| This file is distributed under the University of Illinois Open Source License          | 749 |
| include/CL/sycl/nd_item.hpp  | 755 |
| include/CL/sycl/nd_range.hpp   | 759 |
| include/CL/sycl/opengl_types.hpp   |     |
| TriSYCL wrapper for OpenCL types   | 762 |
| include/CL/sycl/parallelism.hpp  |     |
| Implement parallel constructions to launch kernels                                     | 777 |
| include/CL/sycl/pipe.hpp   | 786 |
| include/CL/sycl/pipe_reservation.hpp   | 797 |
| include/CL/sycl/platform.hpp   | 733 |
| include/CL/sycl/queue.hpp  | 818 |

|  |     |
|--|-----|
| include/CL/sycl/ <a href="#">range.hpp</a>                                       | 824 |
| include/CL/sycl/ <a href="#">static_pipe.hpp</a>                                 | 827 |
| include/CL/sycl/ <a href="#">vec.hpp</a>   |     |
| Implement the small OpenCL vector class  | 829 |
| include/CL/sycl/accessor/detail/ <a href="#">local_accessor.hpp</a>              | 575 |
| include/CL/sycl/address_space/detail/ <a href="#">address_space.hpp</a>          |     |
| Implement OpenCL address spaces in SYCL with C++-style                           | 581 |
| include/CL/sycl/buffer/detail/ <a href="#">accessor.hpp</a>                      | 568 |
| include/CL/sycl/buffer/detail/ <a href="#">buffer.hpp</a>                        | 594 |
| include/CL/sycl/buffer/detail/ <a href="#">buffer_base.hpp</a>                   | 609 |
| include/CL/sycl/buffer/detail/ <a href="#">buffer_waiter.hpp</a>                 | 614 |
| include/CL/sycl/command_group/detail/ <a href="#">task.hpp</a>                   | 619 |
| include/CL/sycl/context/detail/ <a href="#">context.hpp</a>                      | 623 |
| include/CL/sycl/context/detail/ <a href="#">host_context.hpp</a>                 | 632 |
| include/CL/sycl/context/detail/ <a href="#">opencl_context.hpp</a>               | 635 |
| include/CL/sycl/detail/ <a href="#">array_tuple_helpers.hpp</a>                  |     |
| Some helpers to do array-tuple conversions                                       | 638 |
| include/CL/sycl/detail/ <a href="#">cache.hpp</a>                                | 642 |
| include/CL/sycl/detail/ <a href="#">container_element_aspect.hpp</a>             | 645 |
| include/CL/sycl/detail/ <a href="#">debug.hpp</a>                                | 646 |
| include/CL/sycl/detail/ <a href="#">default_classes.hpp</a>                      | 651 |
| include/CL/sycl/detail/ <a href="#">global_config.hpp</a>                        | 654 |
| include/CL/sycl/detail/ <a href="#">linear_id.hpp</a>                            | 657 |
| include/CL/sycl/detail/ <a href="#">shared_ptr_implementation.hpp</a>            | 659 |
| include/CL/sycl/detail/ <a href="#">singleton.hpp</a>                            | 662 |
| include/CL/sycl/detail/ <a href="#">small_array.hpp</a>                          | 663 |
| include/CL/sycl/detail/ <a href="#">unimplemented.hpp</a>                        | 669 |
| include/CL/sycl/device/detail/ <a href="#">device.hpp</a>                        | 671 |
| include/CL/sycl/device/detail/ <a href="#">device_tail.hpp</a>                   | 684 |
| include/CL/sycl/device/detail/ <a href="#">host_device.hpp</a>                   | 685 |
| include/CL/sycl/device/detail/ <a href="#">opencl_device.hpp</a>                 | 688 |
| include/CL/sycl/device_selector/detail/ <a href="#">device_selector_tail.hpp</a> | 693 |
| include/CL/sycl/info/ <a href="#">context.hpp</a>                                | 630 |
| include/CL/sycl/info/ <a href="#">device.hpp</a>                                 | 678 |
| include/CL/sycl/info/ <a href="#">param_traits.hpp</a>                           | 724 |
| include/CL/sycl/info/ <a href="#">platform.hpp</a>                               | 726 |
| include/CL/sycl/kernel/detail/ <a href="#">kernel.hpp</a>                        | 739 |
| include/CL/sycl/kernel/detail/ <a href="#">opencl_kernel.hpp</a>                 | 745 |
| include/CL/sycl/parallelism/detail/ <a href="#">parallelism.hpp</a>              |     |
| Implement the detail of the parallel constructions to launch kernels             | 770 |
| include/CL/sycl/pipe/detail/ <a href="#">pipe.hpp</a>                            | 779 |
| include/CL/sycl/pipe/detail/ <a href="#">pipe_accessor.hpp</a>                   | 789 |
| include/CL/sycl/pipe_reservation/detail/ <a href="#">pipe_reservation.hpp</a>    | 793 |
| include/CL/sycl/platform/detail/ <a href="#">host_platform.hpp</a>               | 800 |
| include/CL/sycl/platform/detail/ <a href="#">host_platform_tail.hpp</a>          | 804 |
| include/CL/sycl/platform/detail/ <a href="#">opencl_platform.hpp</a>             | 805 |
| include/CL/sycl/platform/detail/ <a href="#">opencl_platform_tail.hpp</a>        | 809 |
| include/CL/sycl/platform/detail/ <a href="#">platform.hpp</a>                    | 730 |
| include/CL/sycl/queue/detail/ <a href="#">host_queue.hpp</a>                     | 810 |
| include/CL/sycl/queue/detail/ <a href="#">opencl_queue.hpp</a>                   | 812 |
| include/CL/sycl/queue/detail/ <a href="#">queue.hpp</a>                          | 815 |

## 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::detail::accessor](#)< T, Dimensions, Mode, access::target::local >  
*The local accessor specialization abstracts the way local memory is allocated to a kernel to be shared between work-items of the same work-group. [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...](#)*
- 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...](#)*
- struct [cl::sycl::detail::buffer\\_base](#)  
*Factorize some template independent buffer aspects in a base class. [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...](#)*
- class [cl::sycl::buffer](#)< T, Dimensions, Allocator >  
*A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on. [More...](#)*
- struct [cl::sycl::image](#)< Dimensions >
- struct [cl::sycl::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 allocator objects give the programmer some control on how the memory is allocated inside SYCL.
- template<typename T >  
using `cl::sycl::image_allocator` = `std::allocator< T >`  
The allocator used for the `image` inside SYCL.
- template<typename T >  
using `cl::sycl::map_allocator` = `std::allocator< T >`  
The allocator used to map the memory at the same place.

## 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.
- 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)
- template<typename T , int Dimensions = 1, typename Allocator = `buffer_allocator<std::remove_const_t<T>>>`  
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::detail::accessor< T, Dimensions, Mode, access::target::local >`

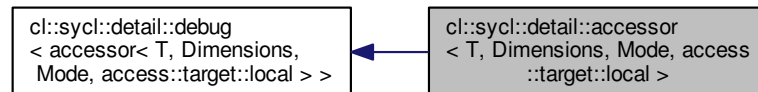
```
template<typename T, int Dimensions, access::mode Mode>
class cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >
```

The local accessor specialization abstracts the way local memory is allocated to a kernel to be shared between work-items of the same work-group.

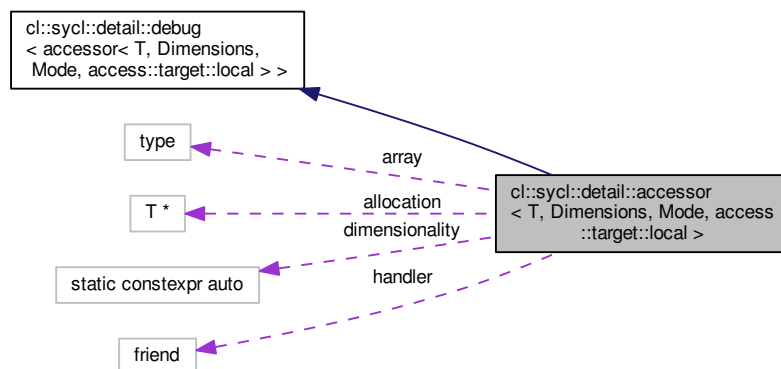
**Todo** Use the `access::mode`

Definition at line 54 of file `local_accessor.hpp`.

Inheritance diagram for `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >`:



Collaboration diagram for `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >`:



## Public Types

- using `value_type` = `T`
- using `element` = `T`
- using `reference` = `typename array_type::reference`
- using `const_reference` = `typename array_type::const_reference`
- using `iterator` = `typename array_type::iterator`

*Inherit the iterator types from the implementation.*

- using `const_iterator` = `typename array_type::const_iterator`
- using `reverse_iterator` = `typename array_type::reverse_iterator`
- using `const_reverse_iterator` = `typename array_type::const_reverse_iterator`

## Public Member Functions

- `accessor` (const `range`< Dimensions > &allocation\_size, `handler` &command\_group\_handler)  
*Construct a device accessor from an existing buffer.*
- `~accessor` ()
- `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 behind the accessor.*
- `auto get_size` () const  
*Returns the size of the underlying buffer storage in bytes.*
- `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.*
- `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_type` = `boost::multi_array_ref< T, Dimensions >`  
*The implementation is a multi\_array\_ref wrapper.*
- using `writable_array_type` = `typename std::remove_const< array_type >::type`

### Private Member Functions

- auto `allocate_accessor` (const `range< Dimensions > &r`)  
*Allocate uninitialized buffer memory.*
- void `deallocate_accessor` ()  
*Deallocate accessor memory.*

### Private Attributes

- `writable_array_type array`  
*The way the buffer is really accessed.*
- `T * allocation` = `nullptr`  
*The allocation on the host for the local accessor.*
- friend `handler`

#### 8.1.2.1.1 Member Typedef Documentation

##### 8.1.2.1.1.1 array\_type

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::array_type =
boost::multi_array_ref<T, Dimensions> [private]
```

The implementation is a multi\_array\_ref wrapper.

Definition at line 61 of file `local_accessor.hpp`.

##### 8.1.2.1.1.2 const\_iterator

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::const_iterator
= typename array_type::const_iterator
```

Definition at line 104 of file `local_accessor.hpp`.

#### 8.1.2.1.1.3 const\_reference

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::const_reference
= typename array_type::const_reference
```

Definition at line 97 of file [local\\_accessor.hpp](#).

#### 8.1.2.1.1.4 const\_reverse\_iterator

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::const_reverse↵
_iterator = typename array_type::const_reverse_iterator
```

Definition at line 107 of file [local\\_accessor.hpp](#).

#### 8.1.2.1.1.5 element

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::element = T
```

Definition at line 95 of file [local\\_accessor.hpp](#).

#### 8.1.2.1.1.6 iterator

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::iterator =
typename array_type::iterator
```

Inherit the iterator types from the implementation.

**Todo** Add iterators to accessors in the specification

Definition at line 103 of file [local\\_accessor.hpp](#).

#### 8.1.2.1.1.7 reference

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::reference =
typename array_type::reference
```

Definition at line 96 of file [local\\_accessor.hpp](#).

## 8.1.2.1.1.8 reverse\_iterator

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::reverse_↵
iterator = typename array_type::reverse_iterator
```

Definition at line 105 of file [local\\_accessor.hpp](#).

## 8.1.2.1.1.9 value\_type

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::value_type = T
```

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

Definition at line 94 of file [local\\_accessor.hpp](#).

## 8.1.2.1.1.10 writable\_array\_type

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::writable_↵
array_type = typename std::remove_const<array_type>::type [private]
```

Definition at line 66 of file [local\\_accessor.hpp](#).

## 8.1.2.1.2 Constructor &amp; Destructor Documentation

## 8.1.2.1.2.1 accessor()

```
template<typename T , int Dimensions, access::mode Mode>
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::accessor (
    const range< Dimensions > & allocation_size,
    handler & command_group_handler ) [inline]
```

Construct a device accessor from an existing buffer.

**Todo** fix the specification to rename target that shadows template param

Definition at line 115 of file [local\\_accessor.hpp](#).

```
00116                                     :
00117     array { allocate_accessor(allocation_size) } {}
```

#### 8.1.2.1.2.2 ~accessor()

```
template<typename T , int Dimensions, access::mode Mode>
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::~~accessor ( ) [inline]
```

Definition at line 121 of file [local\\_accessor.hpp](#).

```
00121         {
00122     deallocate_accessor();
00123     }
```

#### 8.1.2.1.3 Member Function Documentation

##### 8.1.2.1.3.1 allocate\_accessor()

```
template<typename T , int Dimensions, access::mode Mode>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::allocate_↵
accessor (
    const range< Dimensions > & r ) [inline], [private]
```

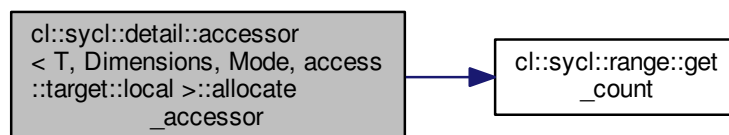
Allocate uninitialized buffer memory.

Definition at line 364 of file [local\\_accessor.hpp](#).

References [cl::sycl::range< Dimensions >::get\\_count\(\)](#).

```
00364                                     {
00365     auto count = r.get_count();
00366     // Allocate uninitialized memory
00367     allocation = std::allocator<value_type>{}.allocate(count);
00368     return boost::multi_array_ref<value_type, Dimensions> { allocation, r };
00369 }
```

Here is the call graph for this function:



8.1.2.1.3.2 `begin()`

```
template<typename T , int Dimensions, access::mode Mode>
iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::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

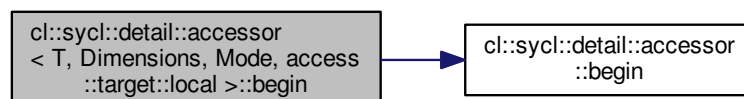
**Todo** Do we need this in `detail::accessor` too or only in `accessor`?

Definition at line 315 of file `local_accessor.hpp`.

References `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array`, and `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::begin()`.

```
00315         {
00316         return const_cast<writable_array_type &>(array).
00317         begin();
00317     }
```

Here is the call graph for this function:



#### 8.1.2.1.3.3 cbegin()

```
template<typename T , int Dimensions, access::mode Mode>
const_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >↵
::cbegin ( ) const [inline]
```

Definition at line 332 of file [local\\_accessor.hpp](#).

```
00332 { return array.begin(); }
```

#### 8.1.2.1.3.4 cend()

```
template<typename T , int Dimensions, access::mode Mode>
const_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::cend
( ) const [inline]
```

Definition at line 335 of file [local\\_accessor.hpp](#).

```
00335 { return array.end(); }
```

#### 8.1.2.1.3.5 crbegin()

```
template<typename T , int Dimensions, access::mode Mode>
const_reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local
>::crbegin ( ) const [inline]
```

Definition at line 356 of file [local\\_accessor.hpp](#).

```
00356 { return array.rbegin(); }
```

#### 8.1.2.1.3.6 crend()

```
template<typename T , int Dimensions, access::mode Mode>
const_reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local
>::crend ( ) const [inline]
```

Definition at line 359 of file [local\\_accessor.hpp](#).

```
00359 { return array.rend(); }
```

## 8.1.2.1.3.7 deallocate\_accessor()

```
template<typename T , int Dimensions, access::mode Mode>
void cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::deallocate_↵
accessor ( ) [inline], [private]
```

Deallocate accessor memory.

Definition at line 373 of file `local_accessor.hpp`.

```
00373         {
00374     std::allocator<value_type>{}.deallocate(allocation, array.num_elements());
00375 }
```

## 8.1.2.1.3.8 end()

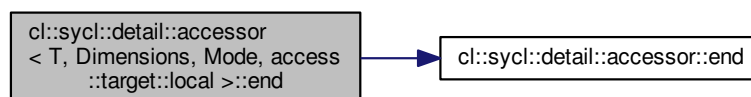
```
template<typename T , int Dimensions, access::mode Mode>
iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::end ( )
const [inline]
```

Definition at line 321 of file `local_accessor.hpp`.

References `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array`, and `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::end()`.

```
00321         {
00322     return const_cast<writable_array_type &>(array).end();
00323 }
```

Here is the call graph for this function:



## 8.1.2.1.3.9 get\_count()

```
template<typename T , int Dimensions, access::mode Mode>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::get_count ( )
const [inline]
```

Returns the total number of elements behind the accessor.

Equal to `get_range()[0] * ... * get_range()[Dimensions-1]`.

**Todo** Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

Definition at line 156 of file `local_accessor.hpp`.

```
00156         {
00157     return array.num_elements();
00158 }
```

#### 8.1.2.1.3.10 get\_range()

```
template<typename T , int Dimensions, access::mode Mode>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::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** Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

Definition at line 134 of file `local_accessor.hpp`.

```
00134         {
00135     /* Interpret the shape which is a pointer to the first element as an
00136        array of Dimensions elements so that the range<Dimensions>
00137        constructor is happy with this collection
00138
00139        \todo Add also a constructor in range<> to accept a const
00140        std::size_t *?
00141    */
00142     return range<Dimensions> {
00143         *(const std::size_t (*) [Dimensions]) (array.shape())
00144     };
00145 }
```

#### 8.1.2.1.3.11 get\_size()

```
template<typename T , int Dimensions, access::mode Mode>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::get_size ( )
const [inline]
```

Returns the size of the underlying buffer storage in bytes.

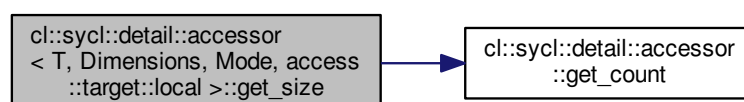
**Todo** Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

Definition at line 167 of file `local_accessor.hpp`.

References `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_count()`.

```
00167     {
00168     return get_count () * sizeof (value_type);
00169 }
```

Here is the call graph for this function:





#### 8.1.2.1.3.12 is\_read\_access()

```
template<typename T , int Dimensions, access::mode Mode>
constexpr bool cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::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 267 of file `local_accessor.hpp`.

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

```
00267                                     {
00268     return Mode == access::mode::read
00269        || Mode == access::mode::read_write
00270        || Mode == access::mode::discard_read_write;
00271 }
```

#### 8.1.2.1.3.13 is\_write\_access()

```
template<typename T , int Dimensions, access::mode Mode>
constexpr bool cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::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 282 of file `local_accessor.hpp`.

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

```
00282                                     {
00283     return Mode == access::mode::write
00284        || Mode == access::mode::read_write
00285        || Mode == access::mode::discard_write
00286        || Mode == access::mode::discard_read_write;
00287 }
```

**8.1.2.1.3.14 operator\*()** [1/2]

```
template<typename T , int Dimensions, access::mode Mode>
reference cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::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 239 of file [local\\_accessor.hpp](#).

```
00239         {
00240     return *array.data();
00241     }
```

**8.1.2.1.3.15 operator\*()** [2/2]

```
template<typename T , int Dimensions, access::mode Mode>
reference cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::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 254 of file [local\\_accessor.hpp](#).

```
00254         {
00255     return *array.data();
00256     }
```

**8.1.2.1.3.16 operator[]()** [1/8]

```
template<typename T , int Dimensions, access::mode Mode>
reference cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::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 177 of file [local\\_accessor.hpp](#).

```
00177         {
00178     return array[index];
00179     }
```

**8.1.2.1.3.17** `operator[]()` [2/8]

```
template<typename T , int Dimensions, access::mode Mode>
reference cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::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 187 of file `local_accessor.hpp`.

```
00187                                     {
00188     return array[index];
00189 }
```

**8.1.2.1.3.18** `operator[]()` [3/8]

```
template<typename T , int Dimensions, access::mode Mode>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[] (
    id< dimensionality > index ) [inline]
```

To use the accessor with `[id<>]`.

Definition at line 193 of file `local_accessor.hpp`.

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

```
00193                                     {
00194     return array(index);
00195 }
```

**8.1.2.1.3.19** `operator[]()` [4/8]

```
template<typename T , int Dimensions, access::mode Mode>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[] (
    id< dimensionality > index ) const [inline]
```

To use the accessor with `[id<>]`.

Definition at line 199 of file `local_accessor.hpp`.

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

```
00199                                     {
00200     return array(index);
00201 }
```

**8.1.2.1.3.20** `operator[]()` [5/8]

```
template<typename T , int Dimensions, access::mode Mode>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[] (
    item< dimensionality > index ) [inline]
```

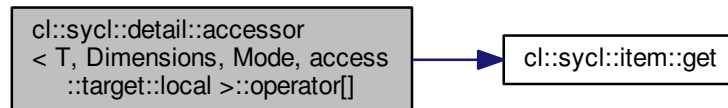
To use an accessor with [item<>].

Definition at line 205 of file `local_accessor.hpp`.

References `cl::sycl::item< Dimensions >::get()`.

```
00205                                     {
00206     return (*this)[index.get()];
00207 }
```

Here is the call graph for this function:

**8.1.2.1.3.21** `operator[]()` [6/8]

```
template<typename T , int Dimensions, access::mode Mode>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[] (
    item< dimensionality > index ) const [inline]
```

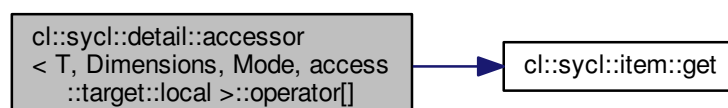
To use an accessor with [item<>].

Definition at line 211 of file `local_accessor.hpp`.

References `cl::sycl::item< Dimensions >::get()`.

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

Here is the call graph for this function:



8.1.2.1.3.22 `operator[]()` [7/8]

```
template<typename T , int Dimensions, access::mode Mode>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[] (
    nd_item< dimensionality > index ) [inline]
```

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

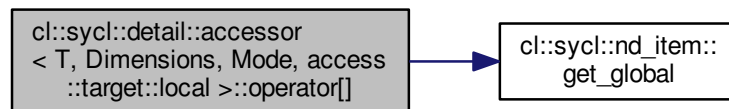
**Todo** Add in the specification because used by HPC-GPU slide 22

Definition at line 220 of file `local_accessor.hpp`.

References `cl::sycl::nd_item< Dimensions >::get_global()`.

```
00220                                     {
00221     return (*this)[index.get_global()];
00222 }
```

Here is the call graph for this function:

8.1.2.1.3.23 `operator[]()` [8/8]

```
template<typename T , int Dimensions, access::mode Mode>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[] (
    nd_item< dimensionality > index ) const [inline]
```

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

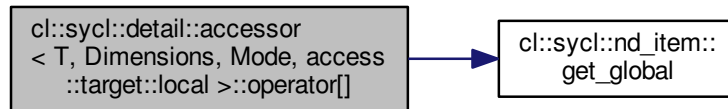
**Todo** Add in the specification because used by HPC-GPU slide 22

Definition at line 228 of file `local_accessor.hpp`.

References `cl::sycl::nd_item< Dimensions >::get_global()`.

```
00228                                     {
00229     return (*this)[index.get_global()];
00230 }
```

Here is the call graph for this function:



#### 8.1.2.1.3.24 rbegin()

```

template<typename T , int Dimensions, access::mode Mode>
reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >↵
::rbegin ( ) const [inline]
  
```

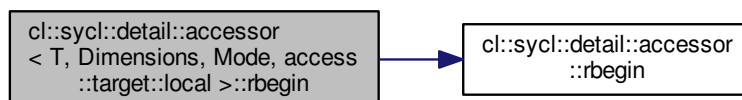
Definition at line 339 of file [local\\_accessor.hpp](#).

References [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array](#), and [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::rbegin\(\)](#).

```

00339         {
00340     return const_cast<writable_array_type &>(array) .
    rbegin();
00341 }
  
```

Here is the call graph for this function:



8.1.2.1.3.25 `rend()`

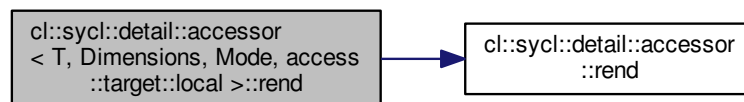
```
template<typename T , int Dimensions, access::mode Mode>
reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >↔
::rend ( ) const [inline]
```

Definition at line 345 of file [local\\_accessor.hpp](#).

References [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array](#), and [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::rend\(\)](#).

```
00345         {
00346     return const_cast<writable_array_type &>(array).rend();
00347 }
```

Here is the call graph for this function:



## 8.1.2.1.4 Member Data Documentation

8.1.2.1.4.1 `allocation`

```
template<typename T , int Dimensions, access::mode Mode>
T* cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::allocation =
nullptr [mutable], [private]
```

The allocation on the host for the local accessor.

Note that this is uninitialized memory, as stated in SYCL specification.

Definition at line 82 of file [local\\_accessor.hpp](#).

8.1.2.1.4.2 `array`

```
template<typename T , int Dimensions, access::mode Mode>
writable_array_type cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >↔
::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 75 of file [local\\_accessor.hpp](#).

#### 8.1.2.1.4.3 dimensionality

```
template<typename T , int Dimensions, access::mode Mode>
constexpr auto cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >↔
::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 90 of file [local\\_accessor.hpp](#).

#### 8.1.2.1.4.4 handler

```
template<typename T , int Dimensions, access::mode Mode>
friend cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::handler
[private]
```

Definition at line 379 of file [local\\_accessor.hpp](#).

### 8.1.2.2 class cl::sycl::accessor

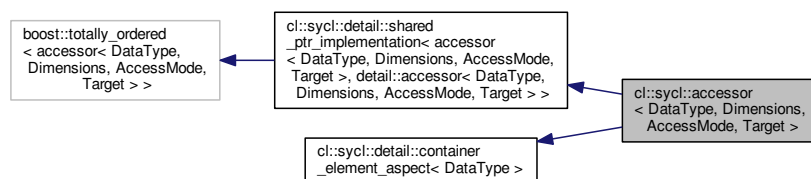
```
template<typename DataType, int Dimensions, access::mode AccessMode, access::target Target = access::target::global_↔
buffer>
class cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >
```

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

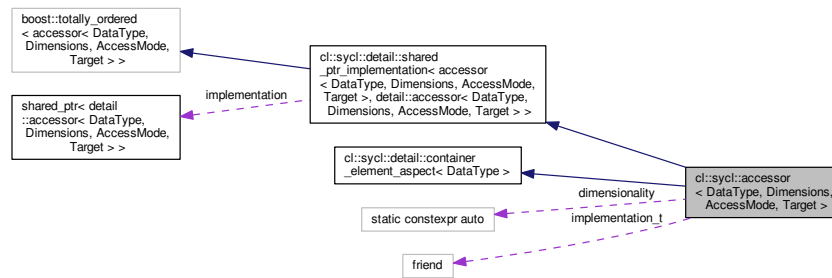
Definition at line 47 of file [accessor.hpp](#).

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





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



## Public 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.*
- `template<typename Allocator >`  
`accessor (buffer< DataType, Dimensions, Allocator > &target_buffer, handler &command_group_handler, const range< Dimensions > &offset, const 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 (const range< Dimensions > &allocation_size, handler &command_group_handler)`  
*Construct an accessor of dimension Dimensions with elements of type DataType using the passed range to specify the size in each dimension.*
- `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 behind the accessor.*
- `auto get_size () const`  
*Returns the size of the underlying buffer storage in bytes.*
- `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.*
- auto `get_pointer` () const  
*Get the pointer to the start of the data.*
- `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` = typename `detail::accessor` < `DataType`, `Dimensions`, `AccessMode`, `Target` >
- using `implementation_t` = typename `accessor::shared_ptr_implementation`

#### Private Attributes

- friend `implementation_t`

#### Additional Inherited Members

##### 8.1.2.2.1 Member Typedef Documentation

###### 8.1.2.2.1.1 `accessor_detail`

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

Definition at line 68 of file `accessor.hpp`.

## 8.1.2.2.1.2 implementation\_t

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
using cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::implementation_t =
typename accessor::shared_ptr_implementation [private]
```

Definition at line 71 of file [accessor.hpp](#).

## 8.1.2.2.2 Constructor &amp; Destructor Documentation

## 8.1.2.2.2.1 accessor() [1/4]

```
template<typename DataType , int 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 97 of file [accessor.hpp](#).

References [cl::sycl::access::constant\\_buffer](#), [cl::sycl::access::global\\_buffer](#), [cl::sycl::detail::shared\\_ptr\\_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation](#), and [cl::sycl::detail::shared\\_ptr\\_implementation< buffer< T, Dimensions, Allocator >, detail::buffer\\_waiter< T, Dimensions, Allocator > >::implementation](#).

```
00098                                     : implementation_t {
00099     new detail::accessor<DataType, Dimensions, AccessMode, Target> {
00100         target_buffer.implementation->implementation, command_group_handler }
00101     } {
00102     static_assert(Target == access::target::global_buffer
00103         || Target == access::target::constant_buffer,
00104         "access target should be global_buffer or constant_buffer "
00105         "when a handler is used");
00106     // Now the implementation is created, register it
00107     implementation->register_accessor();
00108 }
```

**8.1.2.2.2 accessor()** [2/4]

```
template<typename DataType , int 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.

Constructor only available for host\_buffer target.

access\_target defines the form of access being obtained.

Definition at line 118 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](#).

```
00119     : implementation_t {
00120     new detail::accessor<DataType, Dimensions, AccessMode, Target> {
00121         target_buffer.implementation->implementation }
00122     } {
00123     static_assert(Target == access::target::host_buffer,
00124         "without a handler, access target should be host_buffer");
00125     }
```

**8.1.2.2.3 accessor()** [3/4]

```
template<typename DataType , int 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,
    const range< Dimensions > & offset,
    const 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, and constant\_buffer (see Table "Buffer accessor constructors"). access\_target defines the form of access being obtained.

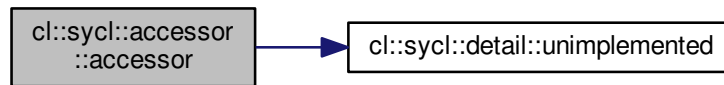
This accessor is recommended for discard-write and discard read write access modes, when the unaffected parts of the processing should be retained.

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

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

```
00149                                     {
00150     detail::unimplemented();
00151     }
```

Here is the call graph for this function:



#### 8.1.2.2.4 accessor() [4/4]

```

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

Construct an accessor of dimension 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 162 of file [accessor.hpp](#).

References [cl::sycl::access::local](#).

```

00164     : implementation_t { new detail::accessor<DataType,
00165                          Dimensions,
00166                          AccessMode,
00167                          access::target::local> {
00168          allocation_size, command_group_handler
00169      }
00170 }
00171 {
00172     static_assert(Target == access::target::local,
00173         "This accessor constructor requires "
00174         "access target be local");
00175 }
  
```

#### 8.1.2.2.3 Member Function Documentation

#### 8.1.2.3.1 begin()

```
template<typename DataType , int 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 343 of file [accessor.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation](#).

```
00343                                     {
00344     return implementation->begin();
00345 }
```

#### 8.1.2.3.2 cbegin()

```
template<typename DataType , int 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 360 of file [accessor.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation](#).

```
00360                                     {
00361     return implementation->cbegin();
00362 }
```

#### 8.1.2.2.3.3 cend()

```
template<typename DataType , int 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 365 of file [accessor.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation](#).

```
00365                                     {
00366     return implementation->cend();
00367 }
```

#### 8.1.2.2.3.4 crbegin()

```
template<typename DataType , int 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 386 of file [accessor.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation](#).

```
00386                                     {
00387     return implementation->rbegin();
00388 }
```

#### 8.1.2.2.3.5 crend()

```
template<typename DataType , int 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 391 of file [accessor.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation](#).

```
00391                                     {
00392     return implementation->rend();
00393 }
```

#### 8.1.2.3.6 end()

```
template<typename DataType , int 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 349 of file [accessor.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation](#).

```
00349                                     {
00350     return implementation->end();
00351 }
```

#### 8.1.2.3.7 get\_count()

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
auto cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::get_count ( ) const
[inline]
```

Returns the total number of elements behind the accessor.

Equal to [get\\_range\(\)\[0\]](#) \* ... \* [get\\_range\(\)\[Dimensions-1\]](#).

**Todo** Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

Definition at line 206 of file [accessor.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation](#).

```
00206                                     {
00207     return implementation->get_count();
00208 }
```

#### 8.1.2.3.8 get\_pointer()

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
auto cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::get_pointer ( ) const
[inline]
```

Get the pointer to the start of the data.

**Todo** Should it be named `data()` instead?

Definition at line 315 of file [accessor.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation](#).

```
00315                                     {
00316     return implementation->get_pointer();
00317 }
```



8.1.2.2.3.9 `get_range()`

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
auto cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::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** Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

Definition at line 186 of file `accessor.hpp`.

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

```
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 implementation->get_range();
00195     }
```

8.1.2.2.3.10 `get_size()`

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
auto cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::get_size ( ) const [inline]
```

Returns the size of the underlying buffer storage in bytes.

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

**Todo** Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

Definition at line 219 of file `accessor.hpp`.

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

```
00219         {
00220         return implementation->get_size();
00221     }
```

**8.1.2.3.11** `operator*()` [1/2]

```
template<typename DataType , int 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 291 of file [accessor.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation](#).

```
00291                                     {
00292     return **implementation;
00293 }
```

**8.1.2.3.12** `operator*()` [2/2]

```
template<typename DataType , int 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 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;
00308 }
```

**8.1.2.3.13 operator[]()** [1/8]

```
template<typename DataType , int 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 229 of file `accessor.hpp`.

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

```
00229                                     {
00230     return (*implementation)[index];
00231 }
```

**8.1.2.3.14 operator[]()** [2/8]

```
template<typename DataType , int 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 239 of file `accessor.hpp`.

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

```
00239                                     {
00240     return (*implementation)[index];
00241 }
```

**8.1.2.3.15 operator[]()** [3/8]

```
template<typename DataType , int 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 245 of file `accessor.hpp`.

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

```
00245                                     {
00246     return (*implementation)[index];
00247 }
```

**8.1.2.3.16** `operator[]()` [4/8]

```
template<typename DataType , int 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 251 of file [accessor.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation](#).

```
00251                                     {
00252     return (*implementation)[index];
00253 }
```

**8.1.2.3.17** `operator[]()` [5/8]

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

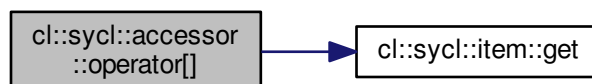
To use an accessor with `[item<>]`.

Definition at line 257 of file [accessor.hpp](#).

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

```
00257                                     {
00258     return (*this)[index.get()];
00259 }
```

Here is the call graph for this function:



**8.1.2.2.3.18** `operator[]()` [6/8]

```
template<typename DataType , int 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]
```

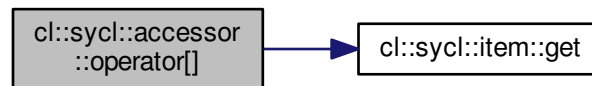
To use an accessor with `[item<>]`.

Definition at line 263 of file [accessor.hpp](#).

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

```
00263                                     {
00264     return (*this)[index.get()];
00265 }
```

Here is the call graph for this function:

**8.1.2.2.3.19** `operator[]()` [7/8]

```
template<typename DataType , int 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<>]`.

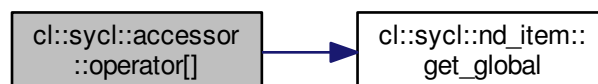
**Todo** Add in the specification because used by HPC-GPU slide 22

Definition at line 272 of file [accessor.hpp](#).

References [cl::sycl::nd\\_item< Dimensions >::get\\_global\(\)](#).

```
00272                                     {
00273     return (*this)[index.get_global()];
00274 }
```

Here is the call graph for this function:



## 8.1.2.2.3.20 operator[]() [8/8]

```
template<typename DataType , int 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<>].

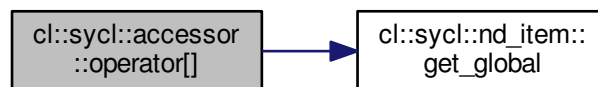
**Todo** Add in the specification because used by HPC-GPU slide 22

Definition at line 280 of file [accessor.hpp](#).

References [cl::sycl::nd\\_item< Dimensions >::get\\_global\(\)](#).

```
00280                                     {
00281     return (*this)[index.get_global()];
00282 }
```

Here is the call graph for this function:



## 8.1.2.2.3.21 rbegin()

```
template<typename DataType , int 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 370 of file [accessor.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation](#).

```
00370                                     {
00371     return implementation->rbegin();
00372 };
```

8.1.2.2.3.22 `rend()`

```
template<typename DataType , int 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 375 of file [accessor.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< \[accessor\]\(#\)< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target >>::implementation](#).

```
00375                                     {
00376     return implementation->rend();
00377 }
```

## 8.1.2.2.4 Member Data Documentation

8.1.2.2.4.1 `dimensionality`

```
template<typename DataType , int 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 61 of file [accessor.hpp](#).

8.1.2.2.4.2 `implementation_t`

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
friend cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::implementation_t [private]
```

Definition at line 74 of file [accessor.hpp](#).

### 8.1.2.3 class `cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >`

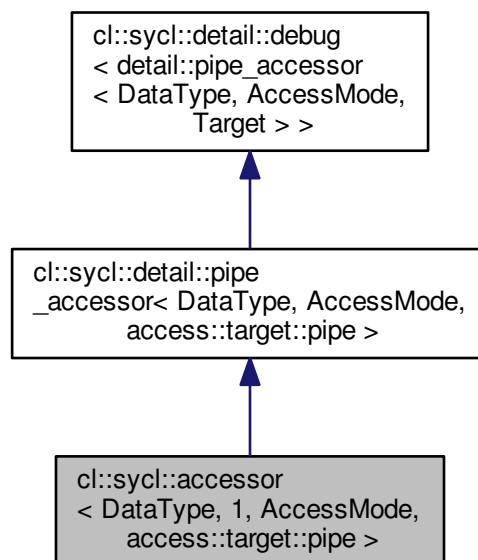
```
template<typename DataType, access::mode AccessMode>
class cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >
```

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

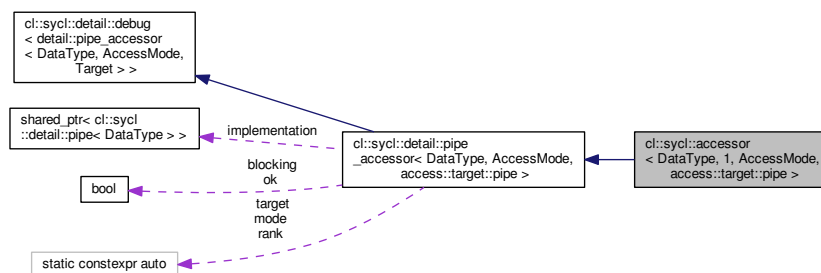
A specialization for an non-blocking pipe

Definition at line 405 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

## 8.1.2.3.1 Member Typedef Documentation

8.1.2.3.1.1 `accessor_detail`

```
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 410 of file `accessor.hpp`.

## 8.1.2.3.2 Constructor &amp; Destructor Documentation

8.1.2.3.2.1 `accessor()`

```
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 419 of file `accessor.hpp`.

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

```
00420      : accessor_detail { p.implementation, command_group_handler } { }
```

## 8.1.2.3.3 Member Function Documentation

#### 8.1.2.3.3.1 get\_pipe\_detail()

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

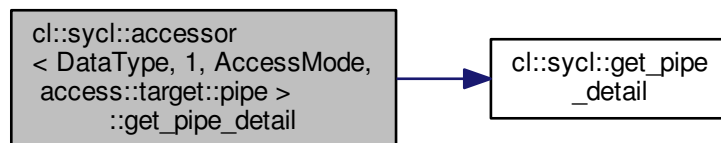
Get the underlying pipe implementation.

Definition at line 429 of file [accessor.hpp](#).

References [cl::sycl::get\\_pipe\\_detail\(\)](#).

```
00429         {
00430     return accessor_detail::get_pipe_detail();
00431     }
```

Here is the call graph for this function:



#### 8.1.2.3.3.2 reserve()

```
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 423 of file [accessor.hpp](#).

```
00423         {
00424     return accessor_detail::reserve(size);
00425     }
```

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

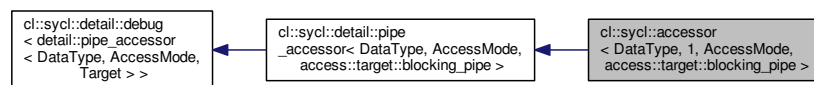
```
template<typename DataType, access::mode AccessMode>
class cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >
```

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

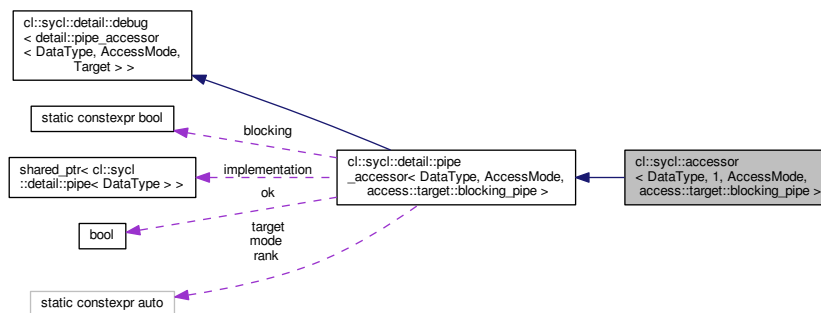
A specialization for a blocking pipe

Definition at line 443 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

### 8.1.2.4.1 Member Typedef Documentation

#### 8.1.2.4.1.1 accessor\_detail

```
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 448 of file [accessor.hpp](#).

### 8.1.2.4.2 Constructor & Destructor Documentation

#### 8.1.2.4.2.1 accessor()

```
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 457 of file [accessor.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< pipe< T >, detail::pipe< T > >::implementation](#).

```
00458      : accessor_detail { p.implementation, command_group_handler } { }
```

### 8.1.2.4.3 Member Function Documentation

8.1.2.4.3.1 `get_pipe_detail()`

```
template<typename DataType , access::mode AccessMode>
auto& cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >::get_pipe←
_detail ( ) [inline]
```

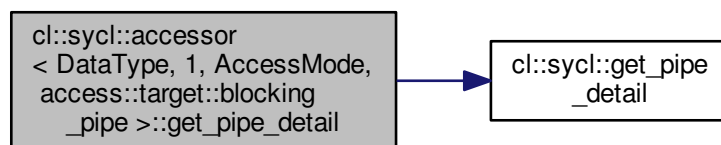
Get the underlying pipe implementation.

Definition at line 468 of file [accessor.hpp](#).

References [cl::sycl::get\\_pipe\\_detail\(\)](#).

```
00468         {
00469     return accessor_detail::get_pipe_detail();
00470     }
```

Here is the call graph for this function:

8.1.2.4.3.2 `reserve()`

```
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 462 of file [accessor.hpp](#).

```
00462                                     {
00463     return accessor_detail::reserve(size);
00464     }
```

### 8.1.2.5 class `cl::sycl::detail::accessor`

```
template<typename T, int 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.

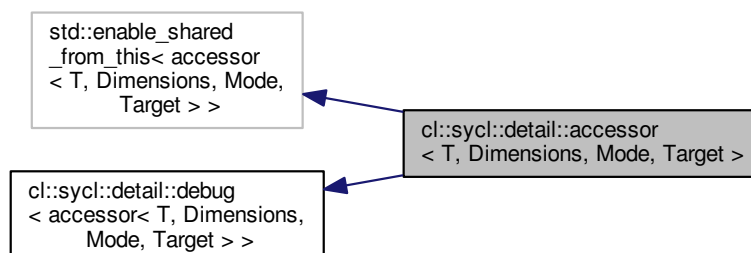
This implementation relies on `boost::multi_array` to provide this nice syntax and behaviour.

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).

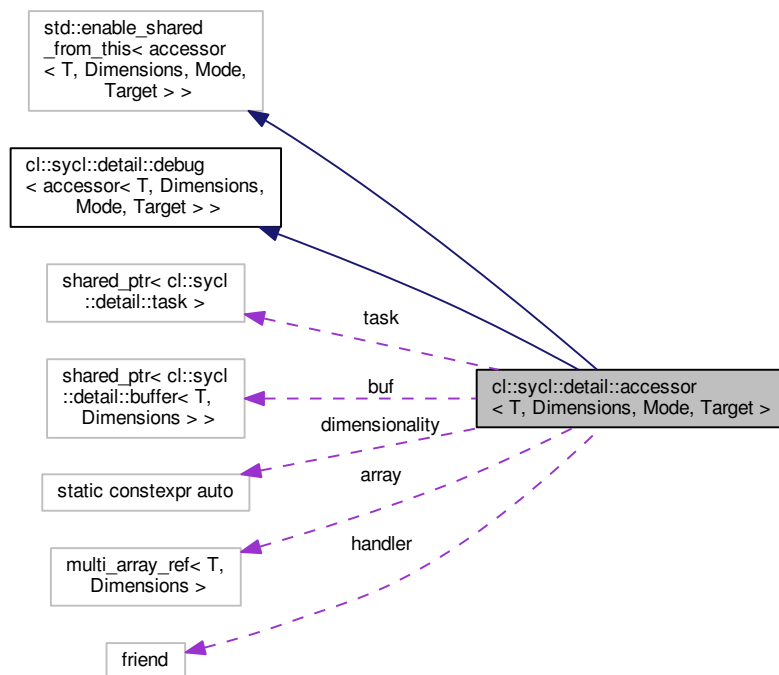
**Todo** Use the `access::mode`

Definition at line 39 of file `local_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.*
- void `register_accessor` ()  
*Register the accessor once a `std::shared_ptr` is created on it.*
- 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 behind the accessor.*
- auto `get_size` () const  
*Returns the size of the underlying buffer storage in bytes.*
- `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.*
- auto `get_pointer` ()  
*Return the pointer to the data.*
- `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 it if required, updates the state of the data in the buffer across contexts.*
- void `copy_back_cl_buffer` ()  
*Does nothing.*

### 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.*
- friend `handler`

#### 8.1.2.5.1 Member Typedef Documentation

##### 8.1.2.5.1.1 array\_view\_type

```
template<typename T , int 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 73 of file `accessor.hpp`.

##### 8.1.2.5.1.2 const\_iterator

```
template<typename T , int 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 111 of file `accessor.hpp`.

##### 8.1.2.5.1.3 const\_reference

```
template<typename T , int 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 104 of file `accessor.hpp`.

#### 8.1.2.5.1.4 `const_reverse_iterator`

```
template<typename T , int 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 114 of file [accessor.hpp](#).

#### 8.1.2.5.1.5 `element`

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

Definition at line 102 of file [accessor.hpp](#).

#### 8.1.2.5.1.6 `iterator`

```
template<typename T , int 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 110 of file [accessor.hpp](#).

#### 8.1.2.5.1.7 `reference`

```
template<typename T , int 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 103 of file [accessor.hpp](#).

#### 8.1.2.5.1.8 `reverse_iterator`

```
template<typename T , int 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 112 of file [accessor.hpp](#).

## 8.1.2.5.1.9 value\_type

```
template<typename T , int 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 101 of file [accessor.hpp](#).

## 8.1.2.5.1.10 writable\_array\_view\_type

```
template<typename T , int 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 77 of file [accessor.hpp](#).

## 8.1.2.5.2 Constructor &amp; Destructor Documentation

## 8.1.2.5.2.1 accessor() [1/2]

```
template<typename T , int 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.

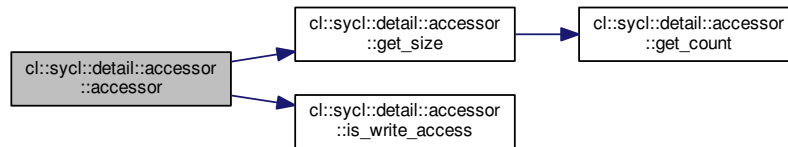
**Todo** fix the specification to rename target that shadows template parm

Definition at line 122 of file [accessor.hpp](#).

References [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get\\_size\(\)](#), [cl::sycl::access::host\\_buffer](#), [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is\\_write\\_access\(\)](#), and [TRISYCL\\_DUMP\\_T](#).

```
00122
00123     buf { target_buffer }, array { target_buffer->access } {
00124         target_buffer->template track_access_mode<Mode>();
00125         TRISYCL_DUMP_T("Create a host accessor write = " <<
is_write_access());
00126         static_assert(Target == access::target::host_buffer,
00127             "without a handler, access target should be host_buffer");
00128         /* The host needs to wait for all the producers of the buffer to
00129             have finished */
00130         buf->wait();
00131
00132 #ifdef TRISYCL_OPENCL
00133     /* For the host context, we are obligated to update the buffer state
00134         during the accessors creation, otherwise we have no way of knowing
00135         if a buffer was modified on the host. This is only true because
00136         host accessors are blocking
00137         */
00138     cl::sycl::context ctx;
00139     buf->update_buffer_state(ctx, Mode, get_size(), array.data());
00140 #endif
00141 }
```

Here is the call graph for this function:



#### 8.1.2.5.2.2 accessor() [2/2]

```

template<typename T , int 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

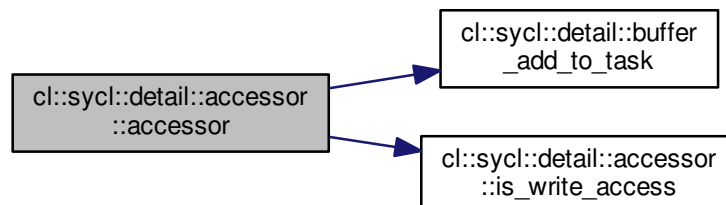
Definition at line 149 of file [accessor.hpp](#).

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](#).

```

00150                                     :
00151     buf { target_buffer }, array { target_buffer->access } {
00152     target_buffer->template track_access_mode<Mode>();
00153     TRISYCL_DUMP_T("Create a kernel accessor write = " <<
is_write_access());
00154     static_assert(Target == access::target::global_buffer
00155         || Target == access::target::constant_buffer,
00156         "access target should be global_buffer or constant_buffer "
00157         "when a handler is used");
00158     // Register the buffer to the task dependencies
00159     task = buffer_add_to_task(buf, &command_group_handler,
is_write_access());
00160     }
  
```

Here is the call graph for this function:



## 8.1.2.5.3 Member Function Documentation

## 8.1.2.5.3.1 begin()

```
template<typename T , int 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`?

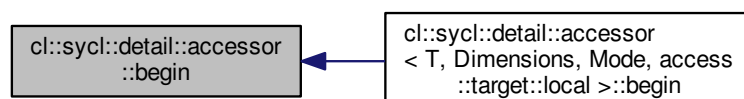
Definition at line 400 of file `accessor.hpp`.

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

Referenced by `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::begin()`.

```
00400     {
00401     return const_cast<writable_array_view_type &>(array) .
    begin();
00402 }
```

Here is the caller graph for this function:



#### 8.1.2.5.3.2 cbegin()

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

Definition at line 417 of file [accessor.hpp](#).

```
00417 { return array.begin(); }
```

#### 8.1.2.5.3.3 cend()

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

Definition at line 420 of file [accessor.hpp](#).

```
00420 { return array.end(); }
```

#### 8.1.2.5.3.4 copy\_back\_cl\_buffer()

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

Does nothing.

Definition at line 473 of file [accessor.hpp](#).

```
00473 {
00474     /* The copy back is handled by the host accessor and the buffer destructor.
00475        We don't need to systematically transfer the data after the
00476        kernel execution
00477
00478        \todo Figure out what to do with this function
00479     */
00480 }
```

8.1.2.5.3.5 `copy_in_cl_buffer()`

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

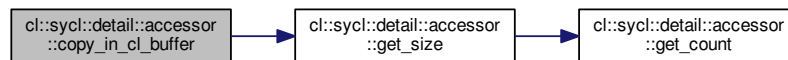
Lazily associate a CL buffer to the SYCL buffer and copy data in it if required, updates the state of the data in the buffer across contexts.

Definition at line 463 of file [accessor.hpp](#).

References [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get\\_size\(\)](#).

```
00463     {
00464     /* Create the OpenCL buffer and copy in it the data from the host if
00465        the buffer doesn't already exists or if the data is not up to date
00466        */
00467     auto ctx = task->get_queue()->get_context();
00468     buf->update_buffer_state(ctx, Mode, get_size(), array.data());
00469 }
```

Here is the call graph for this function:

8.1.2.5.3.6 `crbegin()`

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

Definition at line 441 of file [accessor.hpp](#).

```
00441 { return array.rbegin(); }
```

8.1.2.5.3.7 `crend()`

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

Definition at line 444 of file [accessor.hpp](#).

```
00444 { return array.rend(); }
```

#### 8.1.2.5.3.8 end()

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

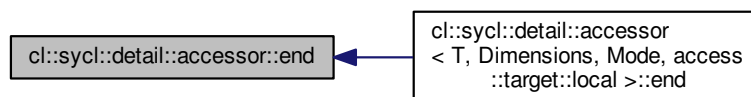
Definition at line 406 of file [accessor.hpp](#).

References [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array](#).

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

```
00406         {
00407         return const_cast<writable_array_view_type >>(array) .
00408         end();
00408     }
```

Here is the caller graph for this function:



#### 8.1.2.5.3.9 get\_buffer()

```
template<typename T , int 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 329 of file [accessor.hpp](#).

References [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::buf](#).

```
00329         {
00330         return *buf;
00331     }
```



8.1.2.5.3.10 `get_cl_buffer()`

```
template<typename T , int 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 453 of file `accessor.hpp`.

```
00453     {
00454         // This throws if not set
00455         auto ctx = task->get_queue()->get_context();
00456         return buf->get_cl_buffer(ctx);
00457     }
```

8.1.2.5.3.11 `get_count()`

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_count ( ) const [inline]
```

Returns the total number of elements behind the accessor.

Equal to `get_range()[0] * ... * get_range()[Dimensions-1]`.

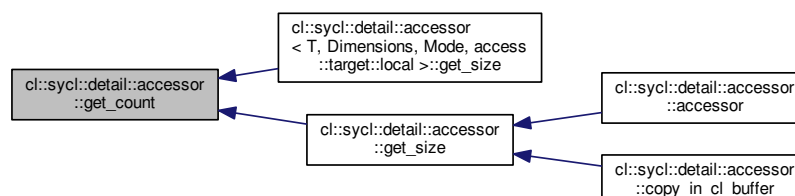
**Todo** Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

Definition at line 225 of file `accessor.hpp`.

Referenced by `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::get_size()`, and `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_size()`.

```
00225     {
00226         return array.num_elements();
00227     }
```

Here is the caller graph for this function:



#### 8.1.2.5.3.12 get\_pointer()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_pointer ( ) [inline]
```

Return the pointer to the data.

**Todo** Implement the various pointer address spaces

Definition at line 370 of file [accessor.hpp](#).

```
00370         {
00371         return array.data();
00372     }
```

#### 8.1.2.5.3.13 get\_range()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::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** Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

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

```
00203         {
00204         /* Interpret the shape which is a pointer to the first element as an
00205         array of Dimensions elements so that the range<Dimensions>
00206         constructor is happy with this collection
00207
00208         \todo Add also a constructor in range<> to accept a const
00209         std::size_t *?
00210         */
00211         return range<Dimensions> {
00212             *(const std::size_t (*)[Dimensions]) (array.shape())
00213         };
00214     }
```

8.1.2.5.3.14 `get_size()`

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

Returns the size of the underlying buffer storage in bytes.

**Todo** Move on [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15564](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564) and [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=14404](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404)

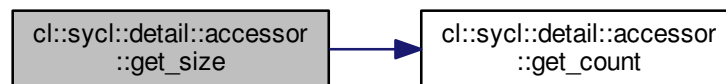
Definition at line 236 of file `accessor.hpp`.

References `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_count()`.

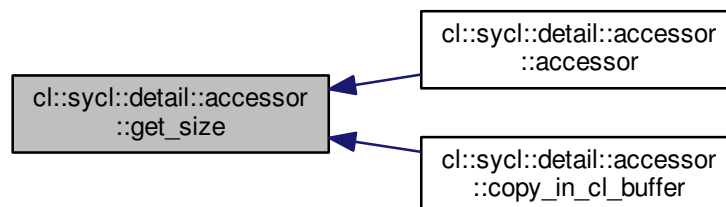
Referenced by `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor()`, and `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_in_cl_buffer()`.

```
00236         {
00237     return get_count()*sizeof(value_type);
00238     }
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.1.2.5.3.15 is\_read\_access()

```
template<typename T , int 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 342 of file `accessor.hpp`.

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

```
00342
00343     return Mode == access::mode::read      {
00344         || Mode == access::mode::read_write
00345         || Mode == access::mode::discard_read_write;
00346     }
```

#### 8.1.2.5.3.16 is\_write\_access()

```
template<typename T , int 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 357 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()`.

```
00357
00358     return Mode == access::mode::write      {
00359         || Mode == access::mode::read_write
00360         || Mode == access::mode::discard_write
00361         || Mode == access::mode::discard_read_write;
00362     }
```

Here is the caller graph for this function:



**8.1.2.5.3.17** `operator*()` [1/2]

```
template<typename T , int 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 308 of file [accessor.hpp](#).

```
00308         {
00309     return *array.data();
00310 }
```

**8.1.2.5.3.18** `operator*()` [2/2]

```
template<typename T , int 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 323 of file [accessor.hpp](#).

```
00323         {
00324     return *array.data();
00325 }
```

**8.1.2.5.3.19** `operator[]()` [1/8]

```
template<typename T , int 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 246 of file [accessor.hpp](#).

```
00246         {
00247     return array[index];
00248 }
```

**8.1.2.5.3.20** `operator[]()` [2/8]

```
template<typename T , int 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 256 of file [accessor.hpp](#).

```
00256                                     {
00257     return array[index];
00258 }
```

**8.1.2.5.3.21** `operator[]()` [3/8]

```
template<typename T , int 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 262 of file [accessor.hpp](#).

References [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array](#).

```
00262                                     {
00263     return array(index);
00264 }
```

**8.1.2.5.3.22** `operator[]()` [4/8]

```
template<typename T , int 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 268 of file [accessor.hpp](#).

References [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array](#).

```
00268                                     {
00269     return array(index);
00270 }
```

**8.1.2.5.3.23** `operator[]()` [5/8]

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

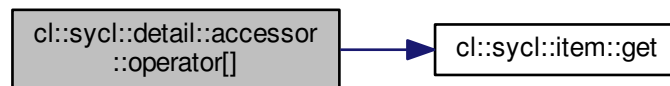
To use an accessor with `[item<>]`.

Definition at line 274 of file [accessor.hpp](#).

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

```
00274                                     {
00275     return (*this)[index.get()];
00276 }
```

Here is the call graph for this function:

**8.1.2.5.3.24** `operator[]()` [6/8]

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

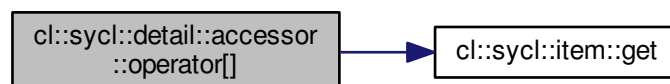
To use an accessor with `[item<>]`.

Definition at line 280 of file [accessor.hpp](#).

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

```
00280                                     {
00281     return (*this)[index.get()];
00282 }
```

Here is the call graph for this function:



#### 8.1.2.5.3.25 `operator[]()` [7/8]

```
template<typename T , int 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<>]`.

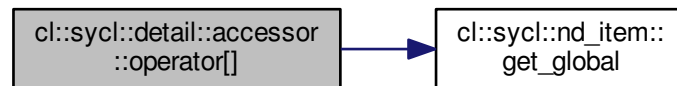
**Todo** Add in the specification because used by HPC-GPU slide 22

Definition at line 289 of file [accessor.hpp](#).

References [cl::sycl::nd\\_item< Dimensions >::get\\_global\(\)](#).

```
00289                                     {
00290     return (*this)[index.get_global()];
00291 }
```

Here is the call graph for this function:



#### 8.1.2.5.3.26 `operator[]()` [8/8]

```
template<typename T , int 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<>]`.

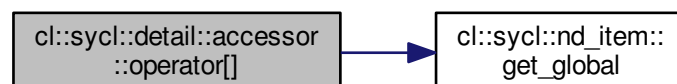
**Todo** Add in the specification because used by HPC-GPU slide 22

Definition at line 297 of file [accessor.hpp](#).

References [cl::sycl::nd\\_item< Dimensions >::get\\_global\(\)](#).

```
00297                                     {
00298     return (*this)[index.get_global()];
00299 }
```

Here is the call graph for this function:





## 8.1.2.5.3.27 rbegin()

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

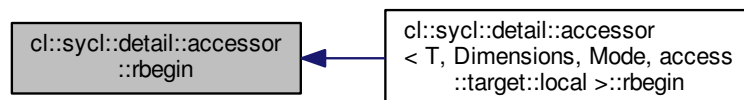
Definition at line 424 of file [accessor.hpp](#).

References [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array](#).

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

```
00424         {
00425     return const_cast<writable_array_view_type &>(array) .
    rbegin();
00426 }
```

Here is the caller graph for this function:



## 8.1.2.5.3.28 register\_accessor()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
void cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::register_accessor ( ) [inline]
```

Register the accessor once a `std::shared_ptr` is created on it.

This is to be called from outside once the object is created. It has been tried directly inside the constructor, but calling `shared_from_this()` from the constructor dead-lock with `libstdc++6`

**Todo** Double-check with the C++ committee on this issue.

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

```
00172         {
00173     #ifdef TRISYCL_OPENCL
00174     if (!task->get_queue()->is_host()) {
00175     // To keep alive this accessor in the following lambdas
00176     auto acc = this->shared_from_this();
00177     /* Before running the kernel, make sure the cl_mem behind this
00178     accessor is up-to-date on the device if needed and pass it to
00179     the kernel */
00180     task->add_prelude([=] {
00181     acc->copy_in_cl_buffer();
00182     });
00183     // After running the kernel, deal with some copy-back if needed
00184     task->add_postlude([=] {
00185     /* Even if this function does nothing, it is required to
00186     have the capture of acc to keep the accessor alive across
00187     the kernel execution up to the execution postlude */
00188     acc->copy_back_cl_buffer();
00189     });
00190     }
00191     #endif
00192 }
```

#### 8.1.2.5.3.29 `rend()`

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

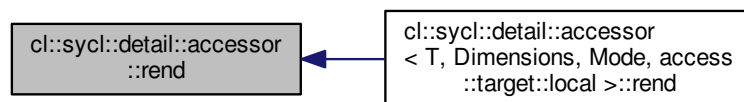
Definition at line 430 of file [accessor.hpp](#).

References [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array](#).

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

```
00430                                     {
00431     return const_cast<writable_array_view_type &>(array) .
    rend();
00432 }
```

Here is the caller graph for this function:



#### 8.1.2.5.4 Member Data Documentation

##### 8.1.2.5.4.1 `array`

```
template<typename T , int 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 86 of file [accessor.hpp](#).

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

## 8.1.2.5.4.2 buf

```
template<typename T , int 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 70 of file [accessor.hpp](#).

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

## 8.1.2.5.4.3 dimensionality

```
template<typename T , int 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 97 of file [accessor.hpp](#).

## 8.1.2.5.4.4 handler

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

Definition at line 449 of file [accessor.hpp](#).

## 8.1.2.5.4.5 task

```
template<typename T , int 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 89 of file [accessor.hpp](#).

### 8.1.2.6 class `cl::sycl::detail::buffer`

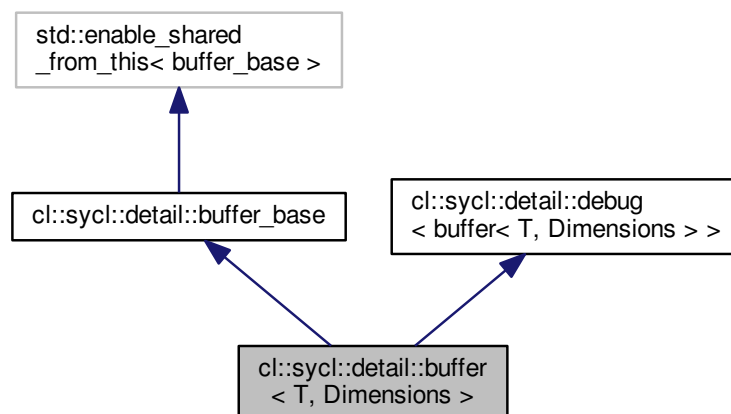
```
template<typename T, int Dimensions = 1>
class cl::sycl::detail::buffer< T, Dimensions >
```

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 32 of file [accessor.hpp](#).

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



```
graph TD
    subgraph Nodes
        direction TB
        N1["std::enable_shared_from_this<buffer_base>"]
        N2["atomic<size_t>"]
        N3["condition_variable"]
        N4["optional<std::promise<void>>"]
        N5["unordered_set<cl::sycl::context>"]
        N6["mutex"]
        N7["unordered_map<cl::sycl::context, boost::compute::buffer>"]
        N8["weak_ptr<cl::sycl::detail::task>"]
        N9["cl::sycl::detail::buffer_base"]
        N10["cl::sycl::detail::debug<buffer<T, Dimensions>>"]
        N11["allocator<non_const_value_type>"]
        N12["optional<std::function<void(void)>>"]
        N13["multi_array_ref<value_type, Dimensions>"]
        N14["bool"]
        N15["non_const_value_type*"]
        N16["shared_ptr_class<T>"]
        N17["cl::sycl::detail::buffer<T, Dimensions>"]
    end

    N1 --> N9
    N2 -.-> N9
    N3 -.-> N9
    N4 -.-> N9
    N5 -.-> N9
    N6 -.-> N9
    N7 -.-> N9
    N8 -.-> N9
    N9 --> N10
    N10 --> N17
    N11 -.-> N17
    N12 -.-> N17
    N13 -.-> N17
    N14 -.-> N17
    N15 -.-> N17
    N16 -.-> N17
    N17 --> N1
```

- using `element` = T
- using `value_type` = T
- using `non_const_value_type` = `std::remove_const_t< value_type >`

- `buffer` (const `range`< Dimensions > &r)  
*Create a new read-write buffer of size.*
- `buffer` (T \*host\_data, const `range`< Dimensions > &r)  
*Create a new read-write buffer from.*
- `template<typename Dependent = T, typename = std::enable_if_t<!std::is_const<Dependent>::value>>`  
`buffer` (const T \*host\_data, const `range`< Dimensions > &r)  
*Create a new read-only buffer from.*
- `buffer` (`shared_ptr_class`< T > &host\_data, const `range`< Dimensions > &r)  
*Create a new buffer with associated memory, using the data in host\_data.*
- `template<typename Iterator >`  
`buffer` (Iterator start\_iterator, Iterator end\_iterator)  
*Create a new allocated 1D buffer from the given elements.*
- `~buffer` ()  
*Create a new sub-buffer without allocation to have separate accessors later.*
- `void mark_as_written` ()  
*Enforce the buffer to be considered as being modified.*
- `template<access::mode Mode, access::target Target = access::target::host_buffer>`  
`void track_access_mode` ()  
*This method is to be called whenever an accessor is created.*

- 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.*
- auto [get\\_size](#) () const  
*Returns the size of the buffer storage in bytes.*
- void [set\\_final\\_data](#) (std::weak\_ptr< T > &&final\_data)  
*Set the weak pointer as destination for write-back on buffer destruction.*
- void [set\\_final\\_data](#) (std::shared\_ptr< T > &&final\_data)  
*Provide destination for write-back on buffer destruction as a shared pointer.*
- void [set\\_final\\_data](#) (std::nullptr\_t)  
*Disable write-back on buffer destruction as an iterator.*
- template<typename Iterator >  
void [set\\_final\\_data](#) (Iterator final\_data)  
*Provide destination for write-back on buffer destruction as an iterator.*
- 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 Member Functions

- auto [allocate\\_buffer](#) (const [range](#)< Dimensions > &r)  
*Allocate uninitialized buffer memory.*
- void [deallocate\\_buffer](#) ()  
*Deallocate buffer memory if required.*
- template<typename BaseType = T, typename DataType >  
void [call\\_update\\_buffer\\_state](#) (cl::sycl::context ctx, [access::mode](#) mode, size\_t size, DataType \*data, std::enable\_if\_t<!std::is\_const< BaseType >::value > \* = 0)  
*Function pair to work around the fact that T might be a const type.*
- template<typename BaseType = T, typename DataType >  
void [call\\_update\\_buffer\\_state](#) (cl::sycl::context ctx, [access::mode](#) mode, size\_t size, DataType \*data, std::enable\_if\_t< std::is\_const< BaseType >::value > \* = 0)  
*Version of `call_update_buffer_state` that does nothing.*

### Private Attributes

- std::allocator< [non\\_const\\_value\\_type](#) > [alloc](#)  
*The allocator to be used when some memory is needed.*
- boost::multi\_array\_ref< [value\\_type](#), 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.*
- [non\\_const\\_value\\_type](#) \* [allocation](#) = nullptr  
*If some allocation is requested on the host for the buffer memory, this is where the memory is attached to.*
- boost::optional< std::function< void(void)> > [final\\_write\\_back](#)
- [shared\\_ptr\\_class](#)< T > [input\\_shared\\_pointer](#)
- bool [data\\_host](#) = false
- bool [copy\\_if\\_modified](#) = false
- bool [modified](#) = false

## Friends

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

## Additional Inherited Members

## 8.1.2.6.1 Member Typedef Documentation

8.1.2.6.1.1 `element`

```
template<typename T, int Dimensions = 1>
using cl::sycl::detail::buffer< T, Dimensions >::element = T
```

Definition at line [49](#) of file [buffer.hpp](#).

8.1.2.6.1.2 `non_const_value_type`

```
template<typename T, int Dimensions = 1>
using cl::sycl::detail::buffer< T, Dimensions >::non\_const\_value\_type = std::remove_const_t<value\_type>
```

Definition at line [53](#) of file [buffer.hpp](#).

8.1.2.6.1.3 `value_type`

```
template<typename T, int Dimensions = 1>
using cl::sycl::detail::buffer< T, Dimensions >::value\_type = T
```

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

## 8.1.2.6.2 Constructor &amp; Destructor Documentation

8.1.2.6.2.1 `buffer()` [1/5]

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

Create a new read-write buffer of size.

## Parameters

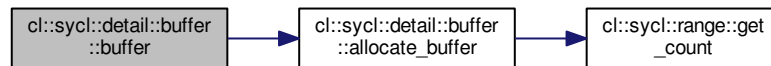
|          |  |
|----------|--|
| <i>r</i> |  |
|----------|--|

Definition at line 107 of file [buffer.hpp](#).

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

```
00107 : access { allocate_buffer(r) } {}
```

Here is the call graph for this function:



#### 8.1.2.6.2.2 `buffer()` [2/5]

```
template<typename T, int 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 112 of file [buffer.hpp](#).

```
00112                                     :
00113     access { host_data, r },
00114     data_host { true }
00115     {}
```

#### 8.1.2.6.2.3 `buffer()` [3/5]

```
template<typename T, int Dimensions = 1>
template<typename Dependent = T, typename = std::enable_if_t<!std::is_const<Dependent>>::value>>
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 |

If the buffer is non const, use a copy-on-write mechanism with internal writable memory.

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

Only enable this constructor if the value type is not constant, because if it is constant, the buffer is constant too.

Definition at line 132 of file `buffer.hpp`.

```

00132                                     :
00133     /* The buffer is read-only, even if the internal multidimensional
00134        wrapper is not. If a write accessor is requested, there should
00135        be a copy on write. So this pointer should not be written and
00136        this const_cast should be acceptable. */
00137     access { const_cast<T *>(host_data), r },
00138     data_host { true },
00139     /* Set copy_if_modified to true, so that if an accessor with write
00140        access is created, data are copied before to be modified. */
00141     copy_if_modified { true }
00142 {}

```

8.1.2.6.2.4 `buffer()` [4/5]

```

template<typename T, int 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 153 of file `buffer.hpp`.

```

00153                                     :
00154     access { host_data.get(), r },
00155     input_shared_pointer { host_data },
00156     data_host { true }
00157 {}

```

#### 8.1.2.6.2.5 `buffer()` [5/5]

```
template<typename T, int Dimensions = 1>
template<typename Iterator >
cl::sycl::detail::buffer< T, Dimensions >::buffer (
    Iterator start_iterator,
    Iterator end_iterator ) [inline]
```

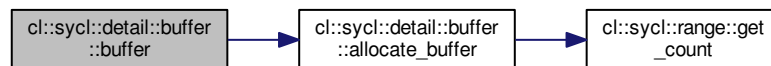
Create a new allocated 1D buffer from the given elements.

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

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

```
00162                                     :
00163     access { allocate_buffer(std::distance(start_iterator, end_iterator)) }
00164     {
00165         /* Then assign allocation since this is the only multi_array
00166            method with this iterator interface */
00167         access.assign(start_iterator, end_iterator);
00168     }
```

Here is the call graph for this function:



#### 8.1.2.6.2.6 `~buffer()`

```
template<typename T, int 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 191 of file [buffer.hpp](#).

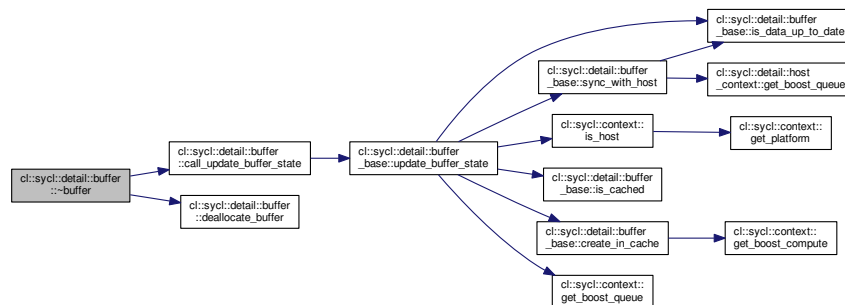
References [cl::sycl::detail::buffer< T, Dimensions >::call\\_update\\_buffer\\_state\(\)](#), [cl::sycl::detail::buffer< T, Dimensions >::deallocate\\_buffer\(\)](#), and [cl::sycl::access::read](#).

```

00191         {
00192     #ifdef TRISYCL_OPENCL
00193         /* We ensure that the host has the most up-to-date version of the data
00194            before the buffer is destroyed. This is necessary because we do not
00195            systematically transfer the data back from a device with
00196            \c copy_back_cl_buffer any more.
00197            \todo Optimize for the case the buffer is not based on host memory
00198         */
00199         cl::sycl::context ctx;
00200         auto size = access.num_elements() * sizeof(value_type);
00201         call_update_buffer_state(ctx, access::mode::read, size,
00202             access.data());
00203     #endif
00204         if (modified && final_write_back)
00205             (*final_write_back)();
00206         // Allocate explicitly allocated memory if required
00207         deallocate_buffer();
00208     }

```

Here is the call graph for this function:



### 8.1.2.6.3 Member Function Documentation

#### 8.1.2.6.3.1 allocate\_buffer()

```

template<typename T, int Dimensions = 1>
auto cl::sycl::detail::buffer< T, Dimensions >::allocate_buffer (
    const range< Dimensions > & r ) [inline], [private]

```

Allocate uninitialized buffer memory.

Definition at line 343 of file [buffer.hpp](#).

References [cl::sycl::detail::buffer< T, Dimensions >::allocation](#), and [cl::sycl::range< Dimensions >::get\\_count\(\)](#).

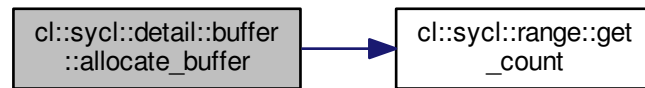
Referenced by [cl::sycl::detail::buffer< T, Dimensions >::buffer\(\)](#), and [cl::sycl::detail::buffer< T, Dimensions >::track\\_access\\_mode\(\)](#).

```

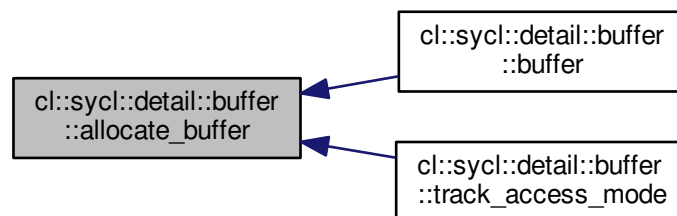
00343         {
00344             auto count = r.get_count();
00345             // Allocate uninitialized memory
00346             allocation = alloc.allocate(count);
00347             return boost::multi_array_ref<value_type, Dimensions> { allocation, r };
00348         }

```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.1.2.6.3.2 `call_update_buffer_state()` [1/2]

```

template<typename T, int Dimensions = 1>
template<typename BaseType = T, typename DataType >
void cl::sycl::detail::buffer< T, Dimensions >::call_update_buffer_state (
    cl::sycl::context ctx,
    access::mode mode,
    size_t size,
    DataType * data,
    std::enable_if_t<!std::is_const< BaseType > ::value > * = 0 ) [inline], [private]
  
```

Function pair to work around the fact that T might be a `const` type.

We call `update_buffer_state` only if T is not `const`, we have to use `enable_if` otherwise the compiler will try to cast `const void*` to `void*` if we create a buffer with a `const` type

**Todo** Use `if constexpr` when it is available with C++17

Definition at line 366 of file `buffer.hpp`.

References `cl::sycl::detail::buffer_base::update_buffer_state()`.

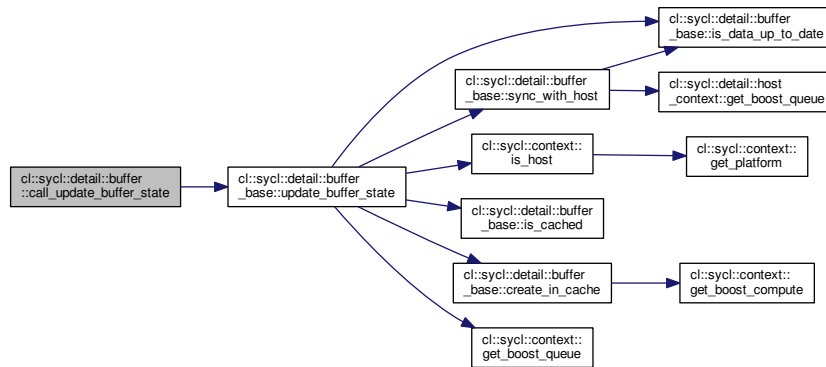
Referenced by `cl::sycl::detail::buffer< T, Dimensions >::~~buffer()`.

```

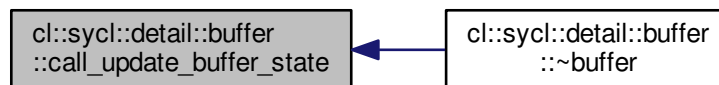
00369     {
00370         update_buffer_state(ctx, mode, size, data);
00371     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.1.2.6.3.3 call\_update\_buffer\_state() [2/2]

```

template<typename T, int Dimensions = 1>
template<typename BaseType = T, typename DataType >
void cl::sycl::detail::buffer< T, Dimensions >::call_update_buffer_state (
    cl::sycl::context ctx,
    access::mode mode,
    size_t size,
    DataType * data,
    std::enable_if_t< std::is_const< BaseType > ::value > * = 0 ) [inline], [private]

```

Version of `call_update_buffer_state` that does nothing.

It is called if the type of the data in the buffer is `const`

Definition at line 378 of file [buffer.hpp](#).

```

00381     { }

```

#### 8.1.2.6.3.4 deallocate\_buffer()

```
template<typename T, int Dimensions = 1>
void cl::sycl::detail::buffer< T, Dimensions >::deallocate_buffer ( ) [inline], [private]
```

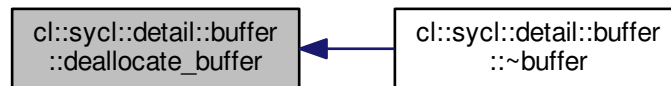
Deallocate buffer memory if required.

Definition at line 352 of file [buffer.hpp](#).

Referenced by [cl::sycl::detail::buffer< T, Dimensions >::~~buffer\(\)](#).

```
00352         {
00353     if (allocation)
00354         alloc.deallocate(allocation, access.num_elements());
00355     }
```

Here is the caller graph for this function:



#### 8.1.2.6.3.5 get\_count()

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

Returns the total number of elements in the buffer.

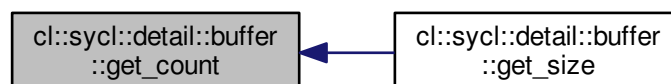
Equal to [get\\_range\(\)\[0\]](#) \* ... \* [get\\_range\(\)\[Dimensions-1\]](#).

Definition at line 280 of file [buffer.hpp](#).

Referenced by [cl::sycl::detail::buffer< T, Dimensions >::get\\_size\(\)](#).

```
00280     {
00281     return access.num_elements();
00282     }
```

Here is the caller graph for this function:



8.1.2.6.3.6 `get_destructor_future()`

```
template<typename T, int Dimensions = 1>
boost::optional<std::future<void> > cl::sycl::detail::buffer< T, Dimensions >::get_destructor←
_future ( ) [inline]
```

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

**Todo** Make the function private again

Definition at line 393 of file [buffer.hpp](#).

References [cl::sycl::detail::buffer\\_base::notify\\_buffer\\_destructor](#).

```
00393                                     {
00394     /* If there is only 1 shared_ptr user of the buffer, this is the
00395        caller of this function, the \c buffer_waiter, so there is no
00396        need to get a \ future otherwise there will be a dead-lock if
00397        there is only 1 thread waiting for itself.
00398
00399        Since \c use_count() is applied to a \c shared_ptr just created
00400        for this purpose, it actually increase locally the count by 1,
00401        so check for 1 + 1 use count instead...
00402    */
00403    // If the buffer's destruction triggers a write-back, wait
00404    if ((shared_from_this().use_count() > 2) &&
00405        modified && (final_write_back || data_host)) {
00406        // Create a promise to wait for
00407        notify_buffer_destructor = std::promise<void> {};
00408        // And return the future to wait for it
00409        return notify_buffer_destructor->get_future();
00410    }
00411    return boost::none;
00412 }
```

8.1.2.6.3.7 `get_range()`

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

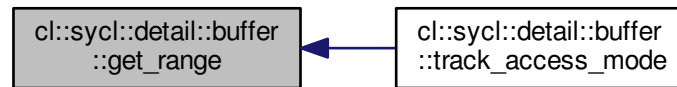
Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.

Definition at line 262 of file [buffer.hpp](#).

Referenced by [cl::sycl::detail::buffer< T, Dimensions >::track\\_access\\_mode\(\)](#).

```
00262     {
00263     /* Interpret the shape which is a pointer to the first element as an
00264        array of Dimensions elements so that the range<Dimensions>
00265        constructor is happy with this collection
00266
00267        \todo Add also a constructor in range<> to accept a const
00268        std::size_t *?
00269    */
00270    return range<Dimensions> {
00271        *(const std::size_t (*)[Dimensions]) (access.shape())
00272    };
00273 }
```

Here is the caller graph for this function:



#### 8.1.2.6.3.8 `get_size()`

```
template<typename T, int Dimensions = 1>
auto cl::sycl::detail::buffer< T, Dimensions >::get_size ( ) const [inline]
```

Returns the size of the buffer storage in bytes.

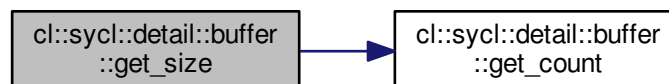
**Todo** rename to something else. In <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf> it is named `bytes()` for example

Definition at line 291 of file `buffer.hpp`.

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

```
00291         {
00292     return get_count () * sizeof (value_type);
00293     }
```

Here is the call graph for this function:





**8.1.2.6.3.9 mark\_as\_written()**

```
template<typename T, int Dimensions = 1>
void cl::sycl::detail::buffer< T, Dimensions >::mark_as_written ( ) [inline]
```

Enforce the buffer to be considered as being modified.

Same as creating an accessor with write access.

Definition at line 214 of file [buffer.hpp](#).

References [cl::sycl::access::host\\_buffer](#).

```
00214                                     {
00215     modified = true;
00216 }
```

**8.1.2.6.3.10 set\_final\_data()** [1/4]

```
template<typename T, int Dimensions = 1>
void cl::sycl::detail::buffer< T, Dimensions >::set_final_data (
    std::weak_ptr< T > && final_data ) [inline]
```

Set the weak pointer as destination for write-back on buffer destruction.

Definition at line 299 of file [buffer.hpp](#).

```
00299                                     {
00300     final_write_back = [=] {
00301         if (auto sptr = final_data.lock()) {
00302             std::copy_n(access.data(), access.num_elements(), sptr.get());
00303         }
00304     };
00305 }
```

**8.1.2.6.3.11 set\_final\_data()** [2/4]

```
template<typename T, int Dimensions = 1>
void cl::sycl::detail::buffer< T, Dimensions >::set_final_data (
    std::shared_ptr< T > && final_data ) [inline]
```

Provide destination for write-back on buffer destruction as a shared pointer.

Definition at line 311 of file [buffer.hpp](#).

```
00311                                     {
00312     final_write_back = [=] {
00313         std::copy_n(access.data(), access.num_elements(), final_data.get());
00314     };
00315 }
```

**8.1.2.6.3.12 set\_final\_data()** [3/4]

```
template<typename T, int Dimensions = 1>
void cl::sycl::detail::buffer< T, Dimensions >::set_final_data (
    std::nullptr_t ) [inline]
```

Disable write-back on buffer destruction as an iterator.

Definition at line 320 of file [buffer.hpp](#).

```
00320
00321     final_write_back = boost::none;
00322 }
```

**8.1.2.6.3.13 set\_final\_data()** [4/4]

```
template<typename T, int Dimensions = 1>
template<typename Iterator >
void cl::sycl::detail::buffer< T, Dimensions >::set_final_data (
    Iterator final_data ) [inline]
```

Provide destination for write-back on buffer destruction as an iterator.

Definition at line 329 of file [buffer.hpp](#).

```
00329
00330 /* using type_ = typename iterator_value_type<Iterator>::value_type;
00331     static_assert(std::is_same<type_, T>::value, "buffer type mismatch");
00332     static_assert(!std::is_const<type_>::value,
00333         "const iterator is not allowed");*/
00334     final_write_back = [=] {
00335         std::copy_n(access.data(), access.num_elements(), final_data);
00336     };
00337 }
```

**8.1.2.6.3.14 track\_access\_mode()**

```
template<typename T, int Dimensions = 1>
template<access::mode Mode, access::target Target = access::target::host_buffer>
void cl::sycl::detail::buffer< T, Dimensions >::track_access_mode ( ) [inline]
```

This method is to be called whenever an accessor is created.

Its current purpose is to track if an accessor with write access is created and acting accordingly.

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

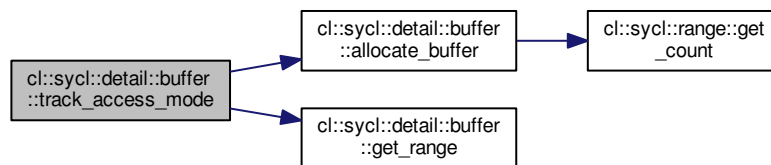
References [cl::sycl::detail::buffer< T, Dimensions >::access](#), [cl::sycl::detail::buffer< T, Dimensions >::allocate\\_buffer\(\)](#), [cl::sycl::access::atomic](#), [cl::sycl::access::discard\\_read\\_write](#), [cl::sycl::access::discard\\_write](#), [cl::sycl::detail::buffer< T, Dimensions >::get\\_range\(\)](#), [cl::sycl::access::read\\_write](#), and [cl::sycl::access::write](#).

```

00226         {
00227     // test if write access is required
00228     if (    Mode == access::mode::write
00229         || Mode == access::mode::read_write
00230         || Mode == access::mode::discard_write
00231         || Mode == access::mode::discard_read_write
00232         || Mode == access::mode::atomic
00233     ) {
00234         modified = true;
00235     if (copy_if_modified) {
00236         // Implement the allocate & copy-on-write optimization
00237         copy_if_modified = false;
00238         data_host = false;
00239         // Since \c allocate_buffer() changes \c access, keep a copy first
00240         auto current_access = access;
00241         /* The range is actually computed from \c access itself, so
00242            save it */
00243         auto current_range = get_range();
00244         allocate_buffer(current_range);
00245         /* Then move everything to the new place
00246
00247            \todo Use std::uninitialized_move instead, when we switch
00248            to full C++17
00249         */
00250         std::copy(current_access.begin(),
00251                 current_access.end(),
00252                 access.begin());
00253     }
00254 }
00255 }

```

Here is the call graph for this function:



#### 8.1.2.6.4 Friends And Related Function Documentation

##### 8.1.2.6.4.1 detail::accessor

```

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

```

Definition at line 63 of file [buffer.hpp](#).

#### 8.1.2.6.5 Member Data Documentation

#### 8.1.2.6.5.1 access

```
template<typename T, int Dimensions = 1>
boost::multi_array_ref<value_type, 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 76 of file [buffer.hpp](#).

Referenced by [cl::sycl::detail::buffer< T, Dimensions >::track\\_access\\_mode\(\)](#).

#### 8.1.2.6.5.2 alloc

```
template<typename T, int Dimensions = 1>
std::allocator<non_const_value_type> cl::sycl::detail::buffer< T, Dimensions >::alloc [private]
```

The allocator to be used when some memory is needed.

**Todo** Implement user-provided allocator

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

#### 8.1.2.6.5.3 allocation

```
template<typename T, int Dimensions = 1>
non_const_value_type* cl::sycl::detail::buffer< T, Dimensions >::allocation = nullptr [private]
```

If some allocation is requested on the host for the buffer memory, this is where the memory is attached to.

Note that this is uninitialized memory, as stated in SYCL specification.

Definition at line 84 of file [buffer.hpp](#).

Referenced by [cl::sycl::detail::buffer< T, Dimensions >::allocate\\_buffer\(\)](#).

#### 8.1.2.6.5.4 copy\_if\_modified

```
template<typename T, int Dimensions = 1>
bool cl::sycl::detail::buffer< T, Dimensions >::copy_if_modified = false [private]
```

Definition at line 99 of file [buffer.hpp](#).

## 8.1.2.6.5.5 data\_host

```
template<typename T, int Dimensions = 1>
bool cl::sycl::detail::buffer< T, Dimensions >::data_host = false [private]
```

Definition at line 96 of file [buffer.hpp](#).

## 8.1.2.6.5.6 final\_write\_back

```
template<typename T, int Dimensions = 1>
boost::optional<std::function<void(void)> > cl::sycl::detail::buffer< T, Dimensions >↵
::final_write_back [private]
```

Definition at line 89 of file [buffer.hpp](#).

## 8.1.2.6.5.7 input\_shared\_pointer

```
template<typename T, int Dimensions = 1>
shared_ptr_class<T> cl::sycl::detail::buffer< T, Dimensions >::input_shared_pointer [private]
```

Definition at line 92 of file [buffer.hpp](#).

## 8.1.2.6.5.8 modified

```
template<typename T, int Dimensions = 1>
bool cl::sycl::detail::buffer< T, Dimensions >::modified = false [private]
```

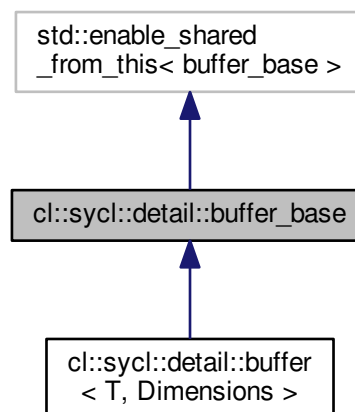
Definition at line 102 of file [buffer.hpp](#).

## 8.1.2.7 struct cl::sycl::detail::buffer\_base

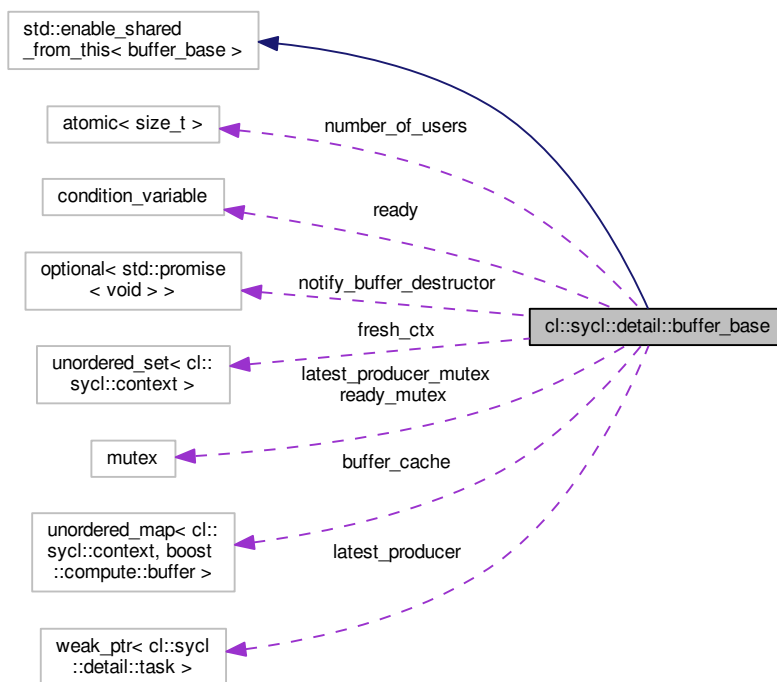
Factorize some template independent buffer aspects in a base class.

Definition at line 49 of file [buffer\\_base.hpp](#).

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



Collaboration diagram for `cl::sycl::detail::buffer_base`:



## Public Member Functions

- `buffer_base()`  
Create a buffer base and marks the host context as the context that holds the most recent version of the data.
- `~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.
- `bool is_data_up_to_date(const cl::sycl::context &ctx)`  
Check if the data of this buffer is up-to-date in a certain context.
- `bool is_cached(const cl::sycl::context &ctx)`  
Check if the buffer is already cached for a certain context.
- `void create_in_cache(const cl::sycl::context &ctx, size_t size, cl_mem_flags flags, void *data)`

Create a `boost::compute::buffer` for this `cl::sycl::buffer` in the cache and associate it with a given context.

- void `sync_with_host` (std::size\_t size, void \*data)  
Transfer the most up-to-date version of the data to the host if the host version is not already up-to-date.
- void `update_buffer_state` (const `cl::sycl::context` &target\_ctx, `access::mode` mode, std::size\_t size, void \*data)  
When a transfer is requested this function is called, it will update the state of the buffer according to the context in which the accessor is created and the access mode.
- boost::compute::buffer `get_cl_buffer` (const `cl::sycl::context` &context)  
Returns the `cl_buffer` for a given context.

## Public Attributes

- 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.
- std::unordered\_set< `cl::sycl::context` > `fresh_ctx`  
To track contexts in which the data is up-to-date.
- std::unordered\_map< `cl::sycl::context`, boost::compute::buffer > `buffer_cache`  
Buffer-side cache that keeps the `boost::compute::buffer` (and the underlying `cl_buffer`) so that if the buffer already exists inside the same context it is not recreated.

### 8.1.2.7.1 Constructor & Destructor Documentation

#### 8.1.2.7.1.1 `buffer_base()`

```
cl::sycl::detail::buffer_base::buffer_base ( ) [inline]
```

Create a buffer base and marks the host context as the context that holds the most recent version of the data.

**Todo** Use lazy allocation for the context tracking set

Definition at line 86 of file `buffer_base.hpp`.

```
00086         : number_of_users { 0 },
00087         fresh_ctx { cl::sycl::context {} } {}
```

#### 8.1.2.7.1.2 ~buffer\_base()

```
cl::sycl::detail::buffer_base::~~buffer_base ( ) [inline]
```

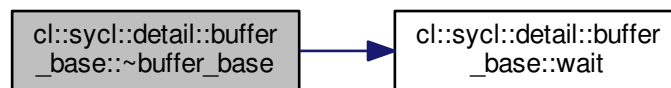
The destructor wait for not being used anymore.

Definition at line 91 of file [buffer\\_base.hpp](#).

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

```
00091         {
00092             wait();
00093             // If there is the last SYCL user buffer waiting, notify it
00094             if (notify_buffer_destructor)
00095                 notify_buffer_destructor->set_value();
00096         }
```

Here is the call graph for this function:



#### 8.1.2.7.2 Member Function Documentation

##### 8.1.2.7.2.1 add\_to\_task()

```
std::shared_ptr<detail::task> cl::sycl::detail::buffer_base::add_to_task (
    handler * command_group_handler,
    bool is_write_mode ) [inline]
```

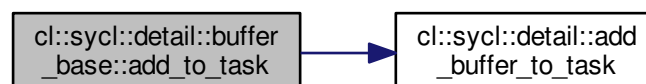
Add a buffer to the task running the command group.

Definition at line 148 of file [buffer\\_base.hpp](#).

References [cl::sycl::detail::add\\_buffer\\_to\\_task\(\)](#).

```
00148                                     {
00149         return add_buffer_to_task(command_group_handler,
00150                                   shared_from_this(),
00151                                   is_write_mode);
00152     }
```

Here is the call graph for this function:





8.1.2.7.2.2 `create_in_cache()`

```
void cl::sycl::detail::buffer_base::create_in_cache (
    const cl::sycl::context & ctx,
    size_t size,
    cl_mem_flags flags,
    void * data ) [inline]
```

Create a `boost::compute::buffer` for this `cl::sycl::buffer` in the cache and associate it with a given context.

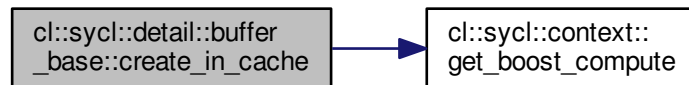
Definition at line 171 of file `buffer_base.hpp`.

References `cl::sycl::context::get_boost_compute()`.

Referenced by `update_buffer_state()`.

```
00172                                     {
00173     buffer_cache[ctx] = boost::compute::buffer
00174     { ctx.get_boost_compute(),
00175       size,
00176       flags,
00177       data
00178     };
00179 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

8.1.2.7.2.3 `get_cl_buffer()`

```
boost::compute::buffer cl::sycl::detail::buffer_base::get_cl_buffer (
    const cl::sycl::context & context ) [inline]
```

Returns the `cl_buffer` for a given context.

Definition at line 307 of file `buffer_base.hpp`.

```
00307                                     {
00308     return buffer_cache[context];
00309 }
```

#### 8.1.2.7.2.4 get\_latest\_producer()

```
std::shared_ptr<detail::task> cl::sycl::detail::buffer_base::get_latest_producer ( ) [inline]
```

Return the latest producer for the buffer.

Definition at line 125 of file [buffer\\_base.hpp](#).

```
00125                                     {
00126     std::lock_guard<std::mutex> lg { latest_producer_mutex };
00127     // Return the valid shared_ptr to the task, if any
00128     return latest_producer.lock();
00129 }
```

#### 8.1.2.7.2.5 is\_cached()

```
bool cl::sycl::detail::buffer_base::is_cached (
    const cl::sycl::context & ctx ) [inline]
```

Check if the buffer is already cached for a certain context.

Definition at line 163 of file [buffer\\_base.hpp](#).

Referenced by [update\\_buffer\\_state\(\)](#).

```
00163                                     {
00164     return buffer_cache.count (ctx);
00165 }
```

Here is the caller graph for this function:



#### 8.1.2.7.2.6 is\_data\_up\_to\_date()

```
bool cl::sycl::detail::buffer_base::is_data_up_to_date (
    const cl::sycl::context & ctx ) [inline]
```

Check if the data of this buffer is up-to-date in a certain context.

Definition at line 157 of file [buffer\\_base.hpp](#).

Referenced by [sync\\_with\\_host\(\)](#), and [update\\_buffer\\_state\(\)](#).

```
00157                                     {
00158     return fresh_ctx.count (ctx);
00159 }
```

Here is the caller graph for this function:



8.1.2.7.2.7 `release()`

```
void cl::sycl::detail::buffer_base::release ( ) [inline]
```

A task has released the buffer.

Definition at line 117 of file `buffer_base.hpp`.

```
00117         {
00118         if (--number_of_users == 0)
00119             // Notify the host consumers or the buffer destructor that it is ready
00120             ready.notify_all();
00121         }
```

8.1.2.7.2.8 `set_latest_producer()`

```
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 136 of file `buffer_base.hpp`.

```
00136         {
00137         std::lock_guard<std::mutex> lg { latest_producer_mutex };
00138         using std::swap;
00139
00140         swap(newer_latest_producer, latest_producer);
00141         // Return the valid shared_ptr to the previous producing task, if any
00142         return newer_latest_producer.lock();
00143     }
```

8.1.2.7.2.9 `sync_with_host()`

```
void cl::sycl::detail::buffer_base::sync_with_host (
    std::size_t size,
    void * data ) [inline]
```

Transfer the most up-to-date version of the data to the host if the host version is not already up-to-date.

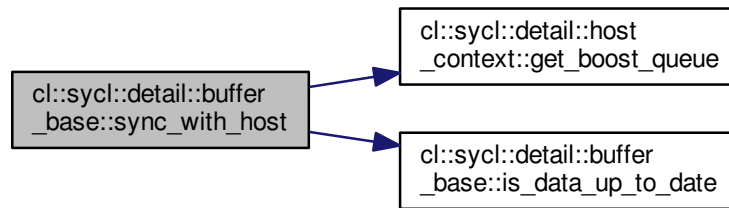
Definition at line 185 of file `buffer_base.hpp`.

References `cl::sycl::detail::host_context::get_boost_queue()`, and `is_data_up_to_date()`.

Referenced by `update_buffer_state()`.

```
00185         {
00186         cl::sycl::context host_context;
00187         if (!is_data_up_to_date(host_context) && !fresh_ctx.empty()) {
00188             /* We know that the context(s) in \c fresh_ctx hold the most recent
00189              version of the buffer
00190              */
00191             auto fresh_context = *(fresh_ctx.begin());
00192             auto fresh_q = fresh_context.get_boost_queue();
00193             fresh_q.enqueue_read_buffer(buffer_cache[fresh_context], 0, size, data);
00194             fresh_ctx.insert(host_context);
00195         }
00196     }
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.1.2.7.2.10 update\_buffer\_state()

```

void cl::sycl::detail::buffer_base::update_buffer_state (
    const cl::sycl::context & target_ctx,
    access::mode mode,
    std::size_t size,
    void * data ) [inline]
  
```

When a transfer is requested this function is called, it will update the state of the buffer according to the context in which the accessor is created and the access mode.

Definition at line 203 of file [buffer\\_base.hpp](#).

References [cl::sycl::access::atomic](#), [create\\_in\\_cache\(\)](#), [cl::sycl::access::discard\\_read\\_write](#), [cl::sycl::access::discard\\_write](#), [cl::sycl::context::get\\_boost\\_queue\(\)](#), [is\\_cached\(\)](#), [is\\_data\\_up\\_to\\_date\(\)](#), [cl::sycl::context::is\\_host\(\)](#), [cl::sycl::access::read](#), [cl::sycl::access::read\\_write](#), [sync\\_with\\_host\(\)](#), and [cl::sycl::access::write](#).

Referenced by [cl::sycl::detail::buffer< T, Dimensions >::call\\_update\\_buffer\\_state\(\)](#).

```

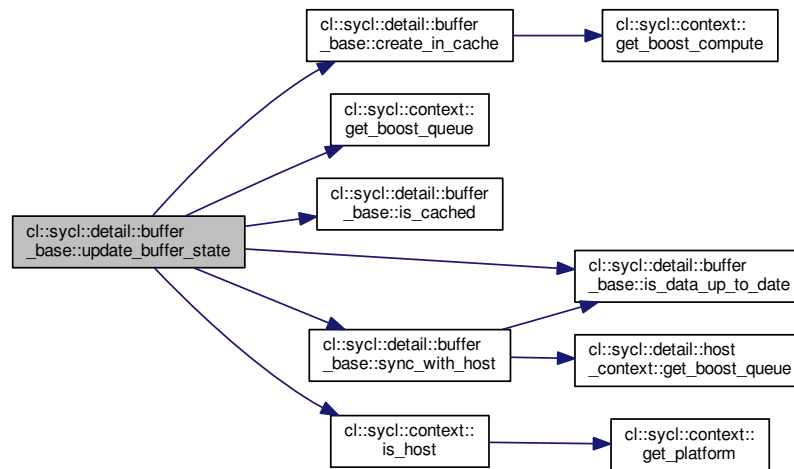
00204                                     {
00205     /* The \c cl_buffer we put in the cache might get accessed again in the
00206        future, this means that we have to always to create it in read/write
00207        mode to be able to write to it if it is accessed through a
00208        write accessor in the future
00209     */
00210     auto constexpr flag = CL_MEM_READ_WRITE;
00211
00212     /* The buffer is accessed in read mode, we want to transfer the data only if
00213        necessary. We start a transfer if the data on the target context is not
00214        up to date and then update the fresh context set.
00215     */
00216     if (mode == access::mode::read) {
  
```

```

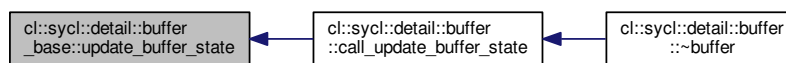
00217
00218     if (is_data_up_to_date(target_ctx))
00219         // If read mode and the data is up-to-date there is nothing to do
00220         return;
00221
00222     // The data is not up-to-date, we need a transfer
00223     // We also want to be sure that the host holds the most recent data
00224     sync_with_host(size, data);
00225
00226     if (!target_ctx.is_host()) {
00227         // If the target context is a device context
00228         if (!is_cached(target_ctx)) {
00229             /* If not cached, we create the buffer and copy the data
00230              * at the same time
00231              */
00232             create_in_cache(target_ctx, size,
00233                             (flag | CL_MEM_COPY_HOST_PTR), data);
00234             fresh_ctx.insert(target_ctx);
00235             return;
00236         }
00237
00238         /* Else we transfer the data to the existing buffer associated
00239          * with the target context buffer
00240          */
00241         auto q = target_ctx.get_boost_queue();
00242         q.enqueue_write_buffer(buffer_cache[target_ctx], 0, size, data);
00243         fresh_ctx.insert(target_ctx);
00244     }
00245     return;
00246 }
00247
00248 /* The buffer might be written to, this means that we have to consider
00249    every version of the data obsolete except in the target context
00250
00251    We go through the same process as in read mode but in addition
00252    we empty the fresh context set and just add the target context
00253
00254    If the data is up to date on the target we just have to update
00255    the context set and nothing else
00256 */
00257 if (!is_data_up_to_date(target_ctx)) {
00258
00259     if ( mode == access::mode::read_write
00260         || mode == access::mode::write
00261         || mode == access::mode::atomic) {
00262         // If the data is not up-to-date in the target context
00263         // We want to host to be up-to-date
00264         sync_with_host(size, data);
00265
00266         if (!target_ctx.is_host()) {
00267             // If the target context is a device context
00268             if (!is_cached(target_ctx)) {
00269                 create_in_cache(target_ctx, size,
00270                                 (flag | CL_MEM_COPY_HOST_PTR), data);
00271             }
00272             else {
00273                 // We update the buffer associated with the target context
00274                 auto q = target_ctx.get_boost_queue();
00275                 q.enqueue_write_buffer(buffer_cache[target_ctx], 0, size, data);
00276             }
00277         }
00278     }
00279
00280     /* When in discard mode we don't need to transfer any data, we just create
00281        the \c cl_buffer if it doesn't exist in the cache
00282     */
00283     if ( mode == access::mode::discard_write
00284         || mode == access::mode::discard_read_write) {
00285         /* We only need to create the buffer if it doesn't exist
00286          * but without copying any data because of the discard mode
00287          */
00288         if (!target_ctx.is_host() && !is_cached(target_ctx)) {
00289             // If the context doesn't exist we create it.
00290             /* We don't want to transfer any data so we don't
00291              * add \c CL_MEM_COPY_HOST_PTR
00292              */
00293             create_in_cache(target_ctx, size, flag, 0);
00294         }
00295     }
00296 }
00297
00298 /* Here we are sure that we are in some kind of write mode,
00299    we indicate that all contexts except the target context
00300    are not up-to-date anymore
00301 */
00302 fresh_ctx.clear();
00303 fresh_ctx.insert(target_ctx);
00304 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.1.2.7.2.11 use()

```
void cl::sycl::detail::buffer_base::use ( ) [inline]
```

Mark this buffer in use by a task.

Definition at line 110 of file [buffer\\_base.hpp](#).

References [number\\_of\\_users](#).

```
00110     {
00111         // Increment the use count
00112         ++number_of_users;
00113     }
```

## 8.1.2.7.2.12 wait()

```
void cl::sycl::detail::buffer_base::wait ( ) [inline]
```

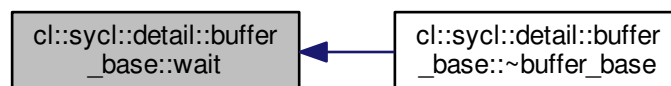
Wait for this buffer to be ready, which is no longer in use.

Definition at line 100 of file [buffer\\_base.hpp](#).

Referenced by [~buffer\\_base\(\)](#).

```
00100     {
00101     std::unique_lock<std::mutex> ul { ready_mutex };
00102     ready.wait(ul, [&] {
00103         // When there is no producer for this buffer, we are ready to use it
00104         return number_of_users == 0;
00105     });
00106 }
```

Here is the caller graph for this function:



## 8.1.2.7.3 Member Data Documentation

## 8.1.2.7.3.1 buffer\_cache

```
std::unordered_map<cl::sycl::context, boost::compute::buffer> cl::sycl::detail::buffer_base←
::buffer_cache
```

Buffer-side cache that keeps the `boost::compute::buffer` (and the underlying `cl_buffer`) so that if the buffer already exists inside the same context it is not recreated.

Definition at line 79 of file [buffer\\_base.hpp](#).

## 8.1.2.7.3.2 fresh\_ctx

```
std::unordered_set<cl::sycl::context> cl::sycl::detail::buffer_base::fresh_ctx
```

To track contexts in which the data is up-to-date.

Definition at line 72 of file [buffer\\_base.hpp](#).

#### 8.1.2.7.3.3 latest\_producer

```
std::weak_ptr<detail::task> cl::sycl::detail::buffer_base::latest_producer
```

Track the latest task to produce this buffer.

Definition at line 55 of file [buffer\\_base.hpp](#).

#### 8.1.2.7.3.4 latest\_producer\_mutex

```
std::mutex cl::sycl::detail::buffer_base::latest_producer_mutex
```

To protect the access to latest\_producer.

Definition at line 57 of file [buffer\\_base.hpp](#).

#### 8.1.2.7.3.5 notify\_buffer\_destructor

```
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 69 of file [buffer\\_base.hpp](#).

Referenced by [cl::sycl::detail::buffer< T, Dimensions >::get\\_destructor\\_future\(\)](#).

#### 8.1.2.7.3.6 number\_of\_users

```
std::atomic<size_t> cl::sycl::detail::buffer_base::number_of_users
```

Definition at line 52 of file [buffer\\_base.hpp](#).

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

#### 8.1.2.7.3.7 ready

```
std::condition_variable cl::sycl::detail::buffer_base::ready
```

To signal when this buffer ready.

Definition at line 60 of file [buffer\\_base.hpp](#).



## 8.1.2.7.3.8 ready\_mutex

```
std::mutex cl::sycl::detail::buffer_base::ready_mutex
```

To protect the access to the condition variable.

Definition at line 62 of file [buffer\\_base.hpp](#).

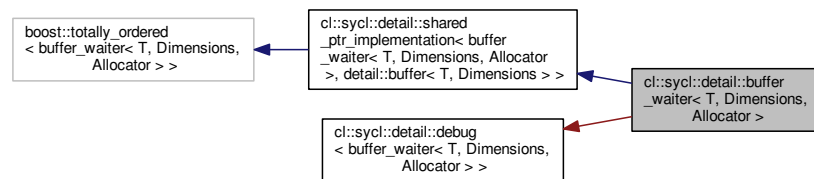
## 8.1.2.8 class cl::sycl::detail::buffer\_waiter

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

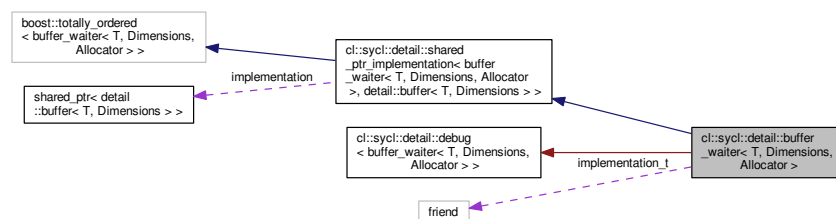
A helper class to wait for the final buffer destruction if the conditions for blocking are met.

Definition at line 33 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](#) = typename [buffer\\_waiter::shared\\_ptr\\_implementation](#)

## Private Attributes

- friend [implementation\\_t](#)

## Additional Inherited Members

### 8.1.2.8.1 Member Typedef Documentation

#### 8.1.2.8.1.1 implementation\_t

```
template<typename T , int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↵
const_t<T>>>>
using cl::sycl::detail::buffer\_waiter< T, Dimensions, Allocator >::implementation\_t = typename
buffer\_waiter::shared\_ptr\_implementation [private]
```

Definition at line 41 of file [buffer\\_waiter.hpp](#).

### 8.1.2.8.2 Constructor & Destructor Documentation

#### 8.1.2.8.2.1 buffer\_waiter()

```
template<typename T , int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↵
const_t<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 52 of file [buffer\\_waiter.hpp](#).

```
00052 : implementation\_t { b } {}
```

8.1.2.8.2.2 `~buffer_waiter()`

```
template<typename T , int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_
const_t<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 58 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`.

```
00058         {
00059         /* Get a future from the implementation if we have to wait for its
00060            destruction */
00061         auto f = implementation->get_destructor_future();
00062         if (f) {
00063             /* No longer carry for the implementation buffer which is free to
00064                live its life up to its destruction */
00065             implementation.reset();
00066             TRISYCL_DUMP_T("~buffer_waiter() is waiting");
00067             // Then wait for its end in some other thread
00068             f->wait();
00069             TRISYCL_DUMP_T("~buffer_waiter() is done");
00070         }
00071     }
```

## 8.1.2.8.3 Member Data Documentation

8.1.2.8.3.1 `implementation_t`

```
template<typename T , int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_
const_t<T>>>
friend cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >::implementation_t [private]
```

Definition at line 44 of file `buffer_waiter.hpp`.

8.1.2.9 `class cl::sycl::buffer`

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_const_t<T>>>
class cl::sycl::buffer< T, Dimensions, Allocator >
```

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** 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

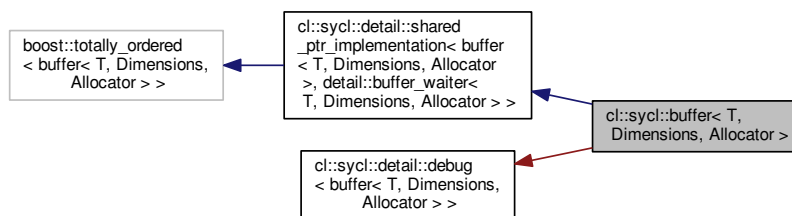
**Todo** Update the specification to have a non-const allocator for const buffer? Or do we rely on `rebind_alloc<T>`. But does this work with `astate-full` allocator?

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

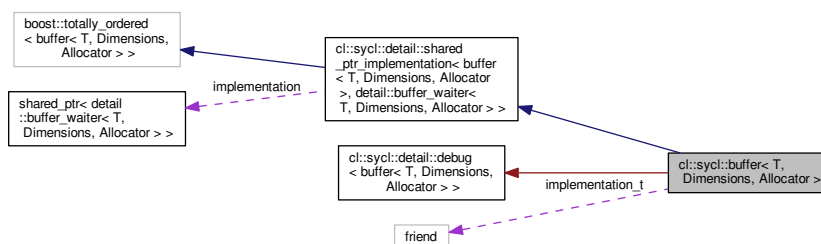
**Todo** Add constructors from `array_ref`

Definition at line 29 of file [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.*
- `template<typename Dependent = T, typename = std::enable_if_t<!std::is_const<Dependent>::value>>`  
`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 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.*
- `void mark_as_written ()`  
*Force the buffer to behave like if we had created an accessor in write 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 constexpr is_read_only () const`  
*Ask for read-only status of the buffer.*
- `void set_final_data (shared_ptr_class< T > finalData)`  
*Set destination of buffer data on destruction.*
- `void set_final_data (weak_ptr_class< T > finalData)`  
*Set destination of buffer data on destruction.*
- `void set_final_data (std::nullptr_t)`  
*Disable write-back on buffer destruction.*
- `bool is_cached (cl::sycl::context &ctx)`

*Check if the buffer is already cached in a certain context.*

- `bool is_data_up_to_date (cl::sycl::context &ctx)`

*Check if the data stored in the buffer is up-to-date in a certain context.*

- `template<typename Iterator >  
void set_final_data (Iterator &&finalData)`

*Set destination of buffer data on destruction.*

### Private Types

- using `implementation_t` = typename `buffer::shared_ptr_implementation`

### Private Attributes

- friend `implementation_t`

## Additional Inherited Members

### 8.1.2.9.1 Member Typedef Documentation

#### 8.1.2.9.1.1 allocator\_type

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↵  
const_t<T>>>  
using cl::sycl::buffer< T, Dimensions, Allocator >::allocator_type = Allocator
```

Definition at line 75 of file `buffer.hpp`.

#### 8.1.2.9.1.2 const\_reference

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↵  
const_t<T>>>  
using cl::sycl::buffer< T, Dimensions, Allocator >::const_reference = const value_type&
```

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

#### 8.1.2.9.1.3 implementation\_t

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↵  
const_t<T>>>  
using cl::sycl::buffer< T, Dimensions, Allocator >::implementation_t = typename buffer::shared_↵  
_ptr_implementation [private]
```

Definition at line 80 of file `buffer.hpp`.

## 8.1.2.9.1.4 reference

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<T>>>>
using cl::sycl::buffer< T, Dimensions, Allocator >::reference = value_type&
```

Definition at line 73 of file [buffer.hpp](#).

## 8.1.2.9.1.5 value\_type

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<T>>>>
using cl::sycl::buffer< T, Dimensions, Allocator >::value_type = T
```

The STL-like types.

Definition at line 72 of file [buffer.hpp](#).

## 8.1.2.9.2 Constructor &amp; Destructor Documentation

## 8.1.2.9.2.1 buffer() [1/9]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<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.

## 8.1.2.9.2.2 buffer() [2/9]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<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 113 of file [buffer.hpp](#).

```
00113                                     {}
00114     : implementation\_t { detail::waiter<T, Dimensions, Allocator>{
00115         new detail::buffer<T, Dimensions> { r }} }
00116     {}
```

#### 8.1.2.9.2.3 [buffer\(\)](#) [3/9]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer\_allocator<std::remove\_const\_t<T>>>
template<typename Dependent = T, typename = std::enable\_if\_t<!std::is\_const<Dependent>>
::value>>
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 <a href="#">cl::sycl::buffer_allocator</a> <T> by default |

The host address is

```
const T*
```

, so the host memory is 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.

Only enable this constructor if it is not the same as the one with

```
const T *host_data
```

, which is when T is already a constant type.

**Todo** Actually this is redundant.

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

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



```

00148             {}
00149     : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
00150                                     { host_data, r }) }
00151     {}

```

Here is the call graph for this function:



#### 8.1.2.9.2.4 buffer() [4/9]

```

template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_<
const_t<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 <code>cl::sycl::buffer_allocator&lt;T&gt;</code> 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 170 of file `buffer.hpp`.

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

```

00172             {}
00173     : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
00174                                     { host_data, r }) }
00175     {}

```

Here is the call graph for this function:



#### 8.1.2.9.2.5 `buffer()` [5/9]

```

template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<T>>>>
cl::sycl::buffer< T, Dimensions, Allocator >::buffer (
    shared_ptr_class< T > & host_data,
    const range< Dimensions > & buffer_range,
    cl::sycl::mutex_class & m,
    Allocator allocator = {} ) [inline]
  
```

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

##### Parameters

|         |                  |  |
|---------|------------------|--|
| in, out | <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 <code>cl::sycl::buffer_allocator&lt;T&gt;</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 199 of file `buffer.hpp`.

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

```

00202                                     {} ) {
00203     detail::unimplemented();
00204 }
  
```

Here is the call graph for this function:



#### 8.1.2.9.2.6 `buffer()` [6/9]

```

template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<T>>>>
cl::sycl::buffer< T, Dimensions, Allocator >::buffer (
    shared_ptr_class< T > host_data,
    const range< Dimensions > & buffer_range,
    Allocator allocator = {} ) [inline]
  
```

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

##### Parameters

|         |                  |  |
|---------|------------------|--|
| in, out | <i>host_data</i> | points to the storage and values used by the buffer  |
| in      | <i>r</i>         | defines the size   |
| in, out | <i>m</i>         | is the mutex used to protect the data access   |
| in      | <i>allocator</i> | is to be used by the SYCL runtime, of type <code>cl::sycl::buffer_allocator&lt;T&gt;</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 227 of file `buffer.hpp`.

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

```

00229         {}
00230     : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
00231         { host_data, buffer_range }) }
00232     {}
  
```

Here is the call graph for this function:



**8.1.2.9.2.7 buffer()** [7/9]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<T>>>>
template<typename InputIterator , typename ValueType = typename std::iterator_traits<InputIterator>::value_type>
cl::sycl::buffer< T, Dimensions, Allocator >::buffer (
    InputIterator start_iterator,
    InputIterator end_iterator,
    Allocator allocator = {} ) [inline]
```

Create a new allocated 1D buffer initialized from the given elements ranging from first up to one before last.

The data is copied to an intermediate memory position by the runtime. Data is written back to the same iterator set if the iterator is not a const iterator.

**Parameters**

|         |                       |   |
|---------|-----------------------|---|
| in, out | <i>start_iterator</i> | points to the first element to copy   |
| in      | <i>end_iterator</i>   | points to just after the last element to copy   |
| in      | <i>allocator</i>      | is to be used by the SYCL runtime, of type <a href="#">cl::sycl::buffer_allocator&lt;T&gt;</a> 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 272 of file [buffer.hpp](#).

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

```
00274         {} ) :
00275     implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
00276         { start_iterator, end_iterator }) }
00277     {}
```

Here is the call graph for this function:



#### 8.1.2.9.2.8 `buffer()` [8/9]

```

template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<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 294 of file `buffer.hpp`.

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

```

00297                                     {} ) { detail::unimplemented(); }
  
```

Here is the call graph for this function:



#### 8.1.2.9.2.9 `buffer()` [9/9]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<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 321 of file `buffer.hpp`.

References `cl::sycl::access::global_buffer`, and `cl::sycl::detail::unimplemented()`.

```
00323         {},
00324         Allocator allocator = {}) { detail::unimplemented(); }
```

Here is the call graph for this function:



#### 8.1.2.9.3 Member Function Documentation

##### 8.1.2.9.3.1 `get_access()` [1/2]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<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 347 of file [buffer.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](#).

```

00347         {
00348     static_assert(Target == access::target::global_buffer
00349         || Target == access::target::constant_buffer,
00350         "get_access(handler) can only deal with access::global_buffer"
00351         " or access::constant_buffer (for host_buffer accessor)"
00352         " do not use a command group handler");
00353     implementation->implementation->template track_access_mode<Mode, Target>();
00354     return { *this, command_group_handler };
00355 }
```

8.1.2.9.3.2 `get_access()` [2/2]

```

template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_
const_t<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 377 of file [buffer.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](#).

```

00377         {
00378             static_assert(Target == access::target::host_buffer,
00379                 "get_access() without a command group handler is only"
00380                 " for host_buffer accessor");
00381             implementation->implementation->template track_access_mode<Mode, Target>();
00382             return { *this };
00383         }

```

#### 8.1.2.9.3.3 get\_count()

```

template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<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 407 of file `buffer.hpp`.

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

```

00407         {
00408             return implementation->implementation->get_count();
00409         }

```

#### 8.1.2.9.3.4 get\_range()

```

template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<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 394 of file `buffer.hpp`.

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

```

00394         {
00395             /* Interpret the shape which is a pointer to the first element as an
00396              * array of Dimensions elements so that the range<Dimensions>
00397              * constructor is happy with this collection
00398              */
00399             return implementation->implementation->get_range();
00400         }

```



8.1.2.9.3.5 `get_size()`

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<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 420 of file `buffer.hpp`.

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

```
00420         {
00421     return implementation->implementation->get_size();
00422     }
```

8.1.2.9.3.6 `is_cached()`

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<T>>>>
bool cl::sycl::buffer< T, Dimensions, Allocator >::is_cached (
    cl::sycl::context & ctx ) [inline]
```

Check if the buffer is already cached in a certain context.

Definition at line 501 of file `buffer.hpp`.

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

```
00501         {
00502     return implementation->implementation->is_cached(ctx);
00503     }
```

8.1.2.9.3.7 `is_data_up_to_date()`

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<T>>>>
bool cl::sycl::buffer< T, Dimensions, Allocator >::is_data_up_to_date (
    cl::sycl::context & ctx ) [inline]
```

Check if the data stored in the buffer is up-to-date in a certain context.

Definition at line 508 of file `buffer.hpp`.

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

```
00508         {
00509     return implementation->implementation->is_data_up_to_date(ctx);
00510     }
```

#### 8.1.2.9.3.8 is\_read\_only()

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↵
const_t<T>>>
bool constexpr 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 447 of file [buffer.hpp](#).

```
00447                                     {
00448     return std::is_const<T>::value;
00449 }
```

#### 8.1.2.9.3.9 mark\_as\_written()

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↵
const_t<T>>>
void cl::sycl::buffer< T, Dimensions, Allocator >::mark_as_written ( ) [inline]
```

Force the buffer to behave like if we had created an accessor in write mode.

Definition at line 361 of file [buffer.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](#).

```
00361                                     {
00362     return implementation->implementation->mark_as_written();
00363 }
```

#### 8.1.2.9.3.10 set\_final\_data() [1/4]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↵
const_t<T>>>
void cl::sycl::buffer< T, Dimensions, Allocator >::set_final_data (
    shared_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

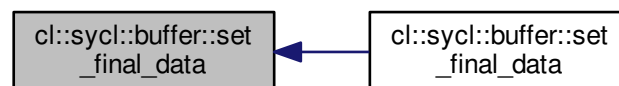
Definition at line 479 of file `buffer.hpp`.

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

Referenced by `cl::sycl::buffer< T, Dimensions, Allocator >::set_final_data()`.

```
00479                                     {
00480     implementation->implementation->set_final_data(std::move(finalData));
00481 }
```

Here is the caller graph for this function:



#### 8.1.2.9.3.11 `set_final_data()` [2/4]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_cv<
const_t<T>>>>
void cl::sycl::buffer< T, Dimensions, Allocator >::set_final_data (
    weak_ptr_class< T > finalData ) [inline]
```

Set destination of buffer data on destruction.

Definition at line 486 of file `buffer.hpp`.

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

```
00486                                     {
00487     implementation->implementation->set_final_data(std::move(finalData));
00488 }
```

**8.1.2.9.3.12 set\_final\_data()** [3/4]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↵
const_t<T>>>
void cl::sycl::buffer< T, Dimensions, Allocator >::set_final_data (
    std::nullptr_t ) [inline]
```

Disable write-back on buffer destruction.

Definition at line 493 of file [buffer.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< buffer< T, Dimensions, Allocator >, detail::buffer\\_waiter< T, Dimensions, Allocator > >::implementation](#).

```
00493                                     {
00494     implementation->implementation->set_final_data(nullptr);
00495 }
```

**8.1.2.9.3.13 set\_final\_data()** [4/4]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↵
const_t<T>>>
template<typename Iterator >
void cl::sycl::buffer< T, Dimensions, Allocator >::set_final_data (
    Iterator && finalData ) [inline]
```

Set destination of buffer data on destruction.

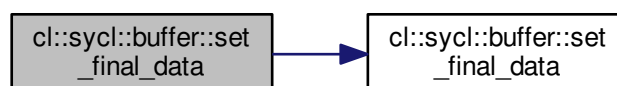
WARNING: the user has to ensure that the object referred to by the iterator will be alive after buffer destruction, otherwise the behaviour is undefined.

Definition at line 520 of file [buffer.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< buffer< T, Dimensions, Allocator >, detail::buffer\\_waiter< T, Dimensions, Allocator > >::implementation](#), and [cl::sycl::buffer< T, Dimensions, Allocator >::set\\_final\\_data\(\)](#).

```
00520                                     {
00521     implementation->implementation->
00522     set\_final\_data(std::forward<Iterator>(finalData));
00523 }
```

Here is the call graph for this function:



## 8.1.2.9.3.14 use\_count()

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↵
const_t<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 437 of file [buffer.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< buffer< T, Dimensions, Allocator >, detail::buffer\\_waiter< T, Dimensions, Allocator > >::implementation](#).

```
00437         {
00438         // Rely on the shared_ptr<> use_count()
00439         return implementation.use_count();
00440     }
```

## 8.1.2.9.4 Member Data Documentation

## 8.1.2.9.4.1 implementation\_t

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↵
const_t<T>>>
friend cl::sycl::buffer< T, Dimensions, Allocator >::implementation_t [private]
```

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

## 8.1.2.10 struct cl::sycl::image

```
template<int Dimensions>
struct cl::sycl::image< Dimensions >
```

**Todo** implement image

Definition at line 23 of file [image.hpp](#).

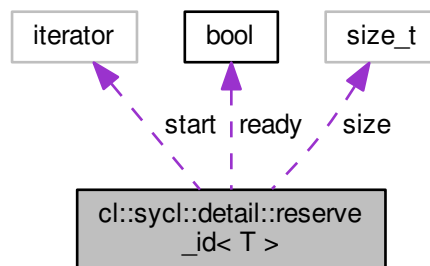
### 8.1.2.11 struct cl::sycl::detail::reserve\_id

```
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

### 8.1.2.11.1 Constructor & Destructor Documentation

#### 8.1.2.11.1.1 reserve\_id()

```
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 } {}
```

## 8.1.2.11.2 Member Data Documentation

## 8.1.2.11.2.1 ready

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

Definition at line 49 of file [pipe.hpp](#).

## 8.1.2.11.2.2 size

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

## 8.1.2.11.2.3 start

```
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](#).

8.1.2.12 class `cl::sycl::detail::pipe`

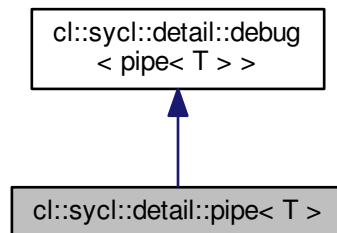
```
template<typename T>
class cl::sycl::detail::pipe< T >
```

Implement a pipe object.

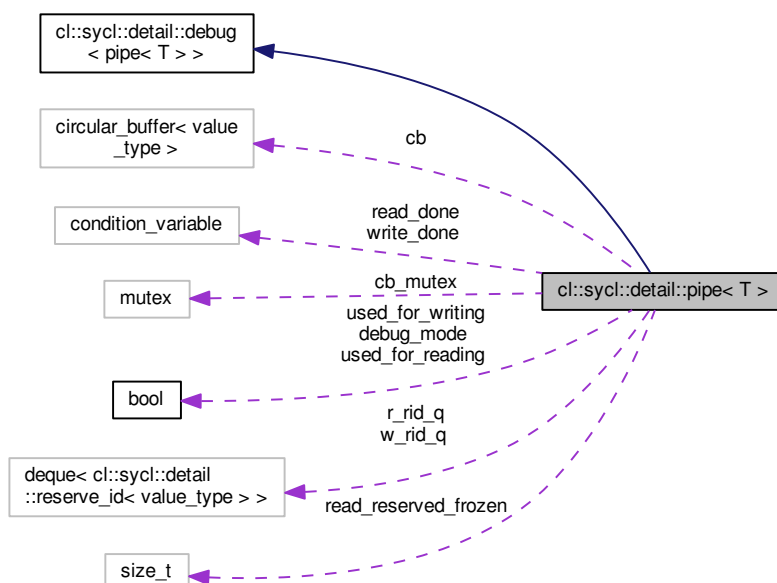
Use some mutable members so that the pipe object can be changed even when the accessors are captured in a lambda.

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.*

#### 8.1.2.12.1 Member Typedef Documentation

##### 8.1.2.12.1.1 implementation\_t

```
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`.

##### 8.1.2.12.1.2 rid\_iterator

```
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`.

##### 8.1.2.12.1.3 value\_type

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

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

#### 8.1.2.12.2 Constructor & Destructor Documentation

## 8.1.2.12.2.1 pipe()

```
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 126 of file [pipe.hpp](#).

```
00126 : cb { capacity }, read_reserved_frozen { 0 } { }
```

## 8.1.2.12.3 Member Function Documentation

## 8.1.2.12.3.1 capacity()

```
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 131 of file [pipe.hpp](#).

```
00131         {
00132     // No lock required since it is fixed and set at construction time
00133     return cb.capacity();
00134 }
```

## 8.1.2.12.3.2 empty()

```
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 166 of file [pipe.hpp](#).

```
00166         {
00167     TRISYCL_DUMP_T("empty() cb.size() = " << cb.size()
00168     << " size() = " << size());
00169     // It is empty when the size is zero, taking into account reservations
00170     return size() == 0;
00171 }
```

#### 8.1.2.12.3.3 empty\_with\_lock()

```
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 197 of file `pipe.hpp`.

```
00197         {
00198     std::lock_guard<std::mutex> lg { cb_mutex };
00199     return empty();
00200 }
```

#### 8.1.2.12.3.4 full()

```
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 182 of file `pipe.hpp`.

```
00182         {
00183     return cb.full();
00184 }
```

#### 8.1.2.12.3.5 full\_with\_lock()

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

Definition at line 204 of file `pipe.hpp`.

```
00204         {
00205     std::lock_guard<std::mutex> lg { cb_mutex };
00206     return full();
00207 }
```

## 8.1.2.12.3.6 move\_read\_reservation\_forward()

```
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 425 of file [pipe.hpp](#).

Referenced by [cl::sycl::detail::pipe\\_reservation< PipeAccessor >::commit\(\)](#).

```
00425                                     {
00426     // Lock the pipe to avoid nuisance
00427     std::lock_guard<std::mutex> lg { cb_mutex };
00428
00429     for (;;) {
00430         if (r_rid_q.empty())
00431             // No pending reservation, so nothing to do
00432             break;
00433         if (!r_rid_q.front().ready)
00434             /* If the first reservation is not ready to be released, stop
00435              because it is blocking all the following in the queue
00436              anyway */
00437             break;
00438         // Remove the reservation to be released from the queue
00439         r_rid_q.pop_front();
00440         std::size_t n_to_pop;
00441         if (r_rid_q.empty())
00442             // If it was the last one, remove all the reservation
00443             n_to_pop = read_reserved_frozen;
00444         else
00445             // Else remove everything up to the next reservation
00446             n_to_pop = r_rid_q.front().start - cb.begin();
00447         // No longer take into account these reserved slots
00448         read_reserved_frozen -= n_to_pop;
00449         // Release the elements from the FIFO
00450         while (n_to_pop--)
00451             cb.pop_front();
00452         // Notify the clients waiting for some room to write in the pipe
00453         read_done.notify_all();
00454         /* ...and process the next reservation to see if it is ready to
00455          be released too */
00456     }
00457 }
```

Here is the caller graph for this function:



## 8.1.2.12.3.7 move\_write\_reservation\_forward()

```
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 463 of file [pipe.hpp](#).

Referenced by [cl::sycl::detail::pipe\\_reservation< PipeAccessor >::commit\(\)](#).

```

00463                                     {
00464     // Lock the pipe to avoid nuisance
00465     std::lock_guard<std::mutex> lg { cb_mutex };
00466
00467     for (;;) {
00468         if (w_rid_q.empty())
00469             // No pending reservation, so nothing to do
00470             break;
00471         // Get the first reservation
00472         const auto &rid = w_rid_q.front();
00473         if (!rid.ready)
00474             /* If the reservation is not ready to be released, stop
00475              because it is blocking all the following in the queue
00476              anyway */
00477             break;
00478         // Remove the reservation to be released from the queue
00479         w_rid_q.pop_front();
00480         // Notify the clients waiting to read something from the pipe
00481         write_done.notify_all();
00482         /* ...and process the next reservation to see if it is ready to
00483          be released too */
00484     }
00485 }

```

Here is the caller graph for this function:



#### 8.1.2.12.3.8 read()

```

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 258 of file [pipe.hpp](#).

```

00258                                     {
00259     // Lock the pipe to avoid being disturbed

```

```

00260     std::unique_lock<std::mutex> ul { cb_mutex };
00261     TRISYCL_DUMP_T("Read pipe empty = " << empty());
00262
00263     if (blocking)
00264         /* If in blocking mode, wait for the not empty condition, that
00265            may be changed when a write is done */
00266         write_done.wait(ul, [&] { return !empty(); });
00267     else if (empty())
00268         return false;
00269
00270     TRISYCL_DUMP_T("Read pipe front = " << cb.front()
00271                   << " back = " << cb.back()
00272                   << " reserved_for_reading() = " << reserved_for_reading());
00273     if (read_reserved_frozen)
00274         /** If there is a pending reservation, read the next element to
00275            be read and update the number of reserved elements */
00276         value = cb.begin()[read_reserved_frozen++];
00277     else {
00278         /* There is no pending read reservation, so pop the read value
00279            from the pipe */
00280         value = cb.front();
00281         cb.pop_front();
00282     }
00283
00284     TRISYCL_DUMP_T("Read pipe value = " << value);
00285     // Notify the clients waiting for some room to write in the pipe
00286     read_done.notify_all();
00287     return true;
00288 }

```

#### 8.1.2.12.3.9 reserve\_read()

```

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 335 of file [pipe.hpp](#).

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

```

00337     {
00338         // Lock the pipe to avoid being disturbed
00339         std::unique_lock<std::mutex> ul { cb_mutex };
00340
00341         TRISYCL_DUMP_T("Before read reservation cb.size() = " << cb.size()
00342                       << " size() = " << size());
00343         if (s == 0)
00344             // Empty reservation requested, so nothing to do
00345             return false;
00346

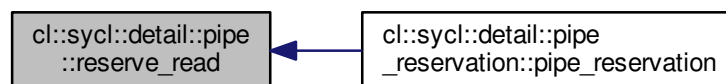
```

```

00347     if (blocking)
00348         /* If in blocking mode, wait for enough elements to read in the
00349            pipe for the reservation. This condition can change when a
00350            write is done */
00351         write_done.wait(ul, [&] { return s <= size(); });
00352     else if (s > size())
00353         // Not enough elements to read in the pipe for the reservation
00354         return false;
00355
00356     // Compute the location of the first element of the reservation
00357     auto first = cb.begin() + read_reserved_frozen;
00358     // Increment the number of frozen elements
00359     read_reserved_frozen += s;
00360     /* Add a description of the reservation at the end of the
00361        reservation queue */
00362     r_rid_q.emplace_back(first, s);
00363     // Return the iterator to the last reservation descriptor
00364     rid = r_rid_q.end() - 1;
00365     TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
00366                   << " size() = " << size());
00367     return true;
00368 }

```

Here is the caller graph for this function:



#### 8.1.2.12.3.10 reserve\_write()

```

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 383 of file [pipe.hpp](#).

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

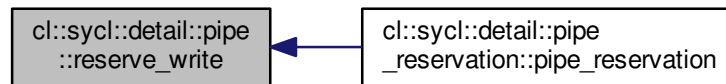


```

00385                                     {
00386     // Lock the pipe to avoid being disturbed
00387     std::unique_lock<std::mutex> ul { cb_mutex };
00388
00389     TRISYCL_DUMP_T("Before write reservation cb.size() = " << cb.size()
00390                   << " size() = " << size());
00391     if (s == 0)
00392         // Empty reservation requested, so nothing to do
00393         return false;
00394
00395     if (blocking)
00396         /* If in blocking mode, wait for enough room in the pipe, that
00397            may be changed when a read is done. Do not use a difference
00398            here because it is only about unsigned values */
00399         read_done.wait(ul, [&] { return cb.size() + s <= capacity(); });
00400     else if (cb.size() + s > capacity())
00401         // Not enough room in the pipe for the reservation
00402         return false;
00403
00404     /* If there is enough room in the pipe, just create default values
00405        in it to do the reservation */
00406     for (std::size_t i = 0; i != s; ++i)
00407         cb.push_back();
00408     /* Compute the location of the first element a posteriori since it
00409        may not exist a priori if cb was empty before */
00410     auto first = cb.end() - s;
00411     /* Add a description of the reservation at the end of the
00412        reservation queue */
00413     w_rid_q.emplace_back(first, s);
00414     // Return the iterator to the last reservation descriptor
00415     rid = w_rid_q.end() - 1;
00416     TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
00417                   << " size() = " << size());
00418     return true;
00419 }

```

Here is the caller graph for this function:



#### 8.1.2.12.3.11 reserved\_for\_reading()

```

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 299 of file [pipe.hpp](#).

```

00299                                     {
00300     return read_reserved_frozen;
00301 }

```

#### 8.1.2.12.3.12 reserved\_for\_writing()

```
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 312 of file [pipe.hpp](#).

```
00312                                     {
00313     if (w_rid_q.empty())
00314         // No on-going reservation
00315         return 0;
00316     else
00317         /* The reserved size is from the first element of the first
00318            on-going reservation up to the end of the pipe content */
00319         return cb.end() - w_rid_q.front().start;
00320 }
```

#### 8.1.2.12.3.13 size()

```
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.

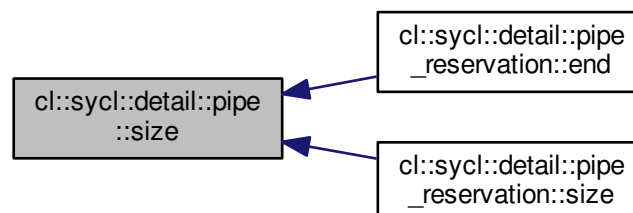
Note that on some devices it may be costly to implement (for example on FPGA).

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

Referenced by [cl::sycl::detail::pipe\\_reservation< PipeAccessor >::end\(\)](#), and [cl::sycl::detail::pipe\\_reservation< PipeAccessor >::size\(\)](#).

```
00146                                     {
00147     TRISYCL_DUMP_T("size() cb.size() = " << cb.size()
00148                   << " cb.end() = " << (void *)&cb.end()
00149                   << " reserved_for_reading() = " << reserved_for_reading()
00150                   << " reserved_for_writing() = " << reserved_for_writing());
00151     /* The actual number of available elements depends from the
00152        elements blocked by some reservations.
00153        This prevents a consumer to read into reserved area. */
00154     return cb.size() - reserved_for_reading() -
00155            reserved_for_writing();
00155 }
```

Here is the caller graph for this function:



8.1.2.12.3.14 `size_with_lock()`

```
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 190 of file `pipe.hpp`.

```
00190
00191     std::lock_guard<std::mutex> lg { cb_mutex };
00192     return size();
00193 }
```

8.1.2.12.3.15 `write()`

```
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 221 of file `pipe.hpp`.

```
00221
00222     // Lock the pipe to avoid being disturbed
00223     std::unique_lock<std::mutex> ul { cb_mutex };
00224     TRISYCL_DUMP_T("Write pipe full = " << full()
00225                   << " value = " << value);
00226
00227     if (blocking)
00228         /* If in blocking mode, wait for the not full condition, that
00229            may be changed when a read is done */
00230         read_done.wait(ul, [&] { return !full(); });
00231     else if (full())
00232         return false;
00233
00234     cb.push_back(value);
00235     TRISYCL_DUMP_T("Write pipe front = " << cb.front()
00236                   << " back = " << cb.back()
00237                   << " cb.begin() = " << (void *)&cb.begin()
00238                   << " cb.size() = " << cb.size()
00239                   << " cb.end() = " << (void *)&cb.end()
00240                   << " reserved_for_reading() = " << reserved_for_reading()
00241                   << " reserved_for_writing() = " << reserved_for_writing());
00242     // Notify the clients waiting to read something from the pipe
00243     write_done.notify_all();
00244     return true;
00245 }
```

#### 8.1.2.12.4 Member Data Documentation

##### 8.1.2.12.4.1 `cb`

```
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](#).

##### 8.1.2.12.4.2 `cb_mutex`

```
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](#).

##### 8.1.2.12.4.3 `debug_mode`

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

To control the debug mode, disabled by default.

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

##### 8.1.2.12.4.4 `r_rid_q`

```
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 103 of file [pipe.hpp](#).

#### 8.1.2.12.4.5 read\_done

```
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 109 of file [pipe.hpp](#).

#### 8.1.2.12.4.6 read\_reserved\_frozen

```
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 106 of file [pipe.hpp](#).

#### 8.1.2.12.4.7 used\_for\_reading

```
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 120 of file [pipe.hpp](#).

#### 8.1.2.12.4.8 used\_for\_writing

```
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 123 of file [pipe.hpp](#).

#### 8.1.2.12.4.9 w\_rid\_q

```
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](#).

#### 8.1.2.12.4.10 write\_done

```
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 112 of file [pipe.hpp](#).

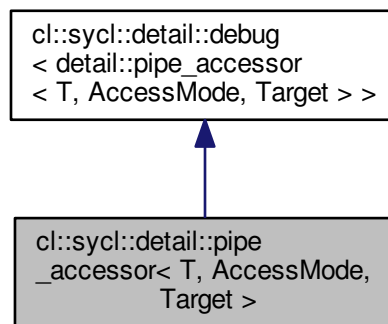
#### 8.1.2.13 class cl::sycl::detail::pipe\_accessor

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

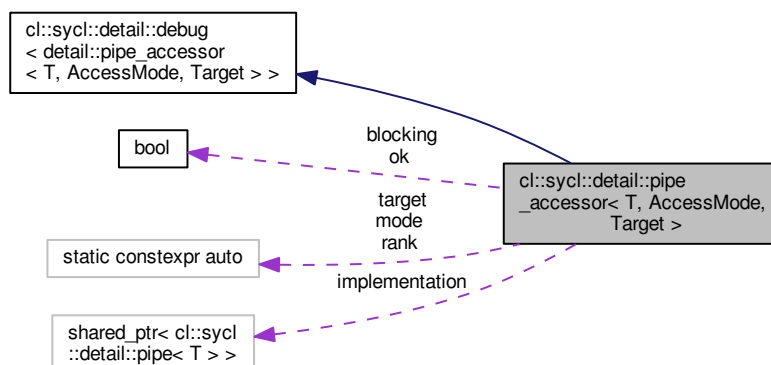
The accessor abstracts the way pipe data are accessed inside a kernel.

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.*

### 8.1.2.13.1 Member Typedef Documentation

#### 8.1.2.13.1.1 `const_reference`

```
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](#).

#### 8.1.2.13.1.2 `reference`

```
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](#).

#### 8.1.2.13.1.3 `value_type`

```
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](#).

### 8.1.2.13.2 Constructor & Destructor Documentation

#### 8.1.2.13.2.1 `pipe_accessor()` [1/2]

```
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     }
```

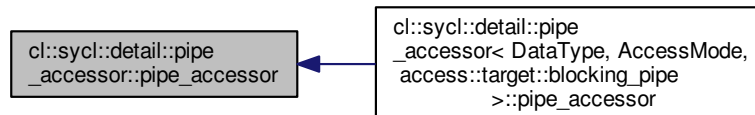


8.1.2.13.2.2 `pipe_accessor()` [2/2]

```
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:

8.1.2.13.2.3 `~pipe_accessor()`

```
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 }
```

## 8.1.2.13.3 Member Function Documentation

8.1.2.13.3.1 `capacity()`

```
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 }
```

#### 8.1.2.13.3.2 empty()

```
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     }
```

#### 8.1.2.13.3.3 full()

```
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     }
```

#### 8.1.2.13.3.4 get\_pipe\_detail()

```
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     }
```

#### 8.1.2.13.3.5 operator bool()

```
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 }
```

#### 8.1.2.13.3.6 operator<<()

```
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 }
```

### 8.1.2.13.3.7 operator>>()

```
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 }
```

### 8.1.2.13.3.8 read() [1/2]

```
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 }
```

**8.1.2.13.3.9 read()** [2/2]

```
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

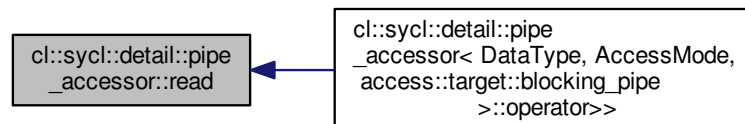
This function is const so it can work when the accessor is passed by copy in the [=] kernel lambda, which is not mutable by default

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:

**8.1.2.13.3.10 reserve()**

```
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 }
```

#### 8.1.2.13.3.11 set\_debug()

```
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     }
```

#### 8.1.2.13.3.12 size()

```
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     }
```

#### 8.1.2.13.3.13 write()

```
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

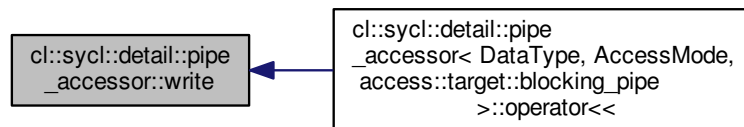
Definition at line 180 of file [pipe\\_accessor.hpp](#).

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:



#### 8.1.2.13.4 Member Data Documentation

##### 8.1.2.13.4.1 blocking

```

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](#).

##### 8.1.2.13.4.2 implementation

```

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\(\)](#).

#### 8.1.2.13.4.3 mode

```
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](#).

#### 8.1.2.13.4.4 ok

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

#### 8.1.2.13.4.5 rank

```
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](#).

#### 8.1.2.13.4.6 target

```
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](#).



8.1.2.14 class `cl::sycl::pipe`

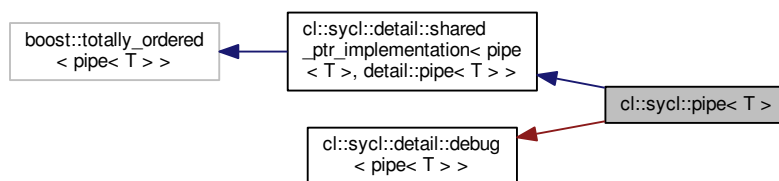
```
template<typename T>
class cl::sycl::pipe< T >
```

A SYCL pipe.

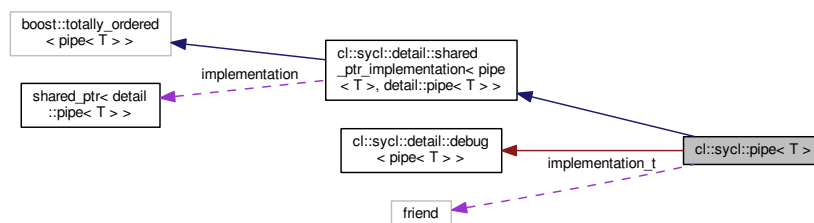
Implement a FIFO-style object that can be used through accessors to send some objects `T` from the input to the output

Definition at line 31 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](#) = typename [pipe::shared\\_ptr\\_implementation](#)

### Private Attributes

- friend [implementation\\_t](#)

## Additional Inherited Members

### 8.1.2.14.1 Member Typedef Documentation

#### 8.1.2.14.1.1 [implementation\\_t](#)

```
template<typename T>
using cl::sycl::pipe< T >::implementation\_t = typename pipe::shared\_ptr\_implementation [private]
```

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

#### 8.1.2.14.1.2 [value\\_type](#)

```
template<typename T>
using cl::sycl::pipe< T >::value\_type = T
```

The STL-like types.

Definition at line 53 of file [pipe.hpp](#).

### 8.1.2.14.2 Constructor & Destructor Documentation

#### 8.1.2.14.2.1 [pipe\(\)](#)

```
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 57 of file [pipe.hpp](#).

References [cl::sycl::access::pipe](#).

```
00058      : implementation\_t { new detail::pipe<T> { capacity } } { }
```

## 8.1.2.14.3 Member Function Documentation

## 8.1.2.14.3.1 capacity()

```
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 83 of file [pipe.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< pipe< T >, detail::pipe< T > >::implementation](#).

```
00083         {
00084     return implementation->capacity();
00085     }
```

## 8.1.2.14.3.2 get\_access()

```
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 73 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](#).

```
00073         {
00074     static_assert(Target == access::target::pipe
00075         || Target == access::target::blocking_pipe,
00076         "get_access(handler) with pipes can only deal with "
00077         "access::pipe or access::blocking_pipe");
00078     return { implementation, command_group_handler };
00079     }
```

## 8.1.2.14.4 Member Data Documentation

#### 8.1.2.14.4.1 implementation\_t

```
template<typename T>
friend cl::sycl::pipe< T >::implementation_t [private]
```

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

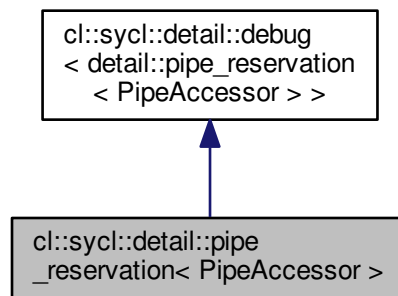
#### 8.1.2.15 class `cl::sycl::detail::pipe_reservation`

```
template<typename PipeAccessor>
class cl::sycl::detail::pipe\_reservation< PipeAccessor >
```

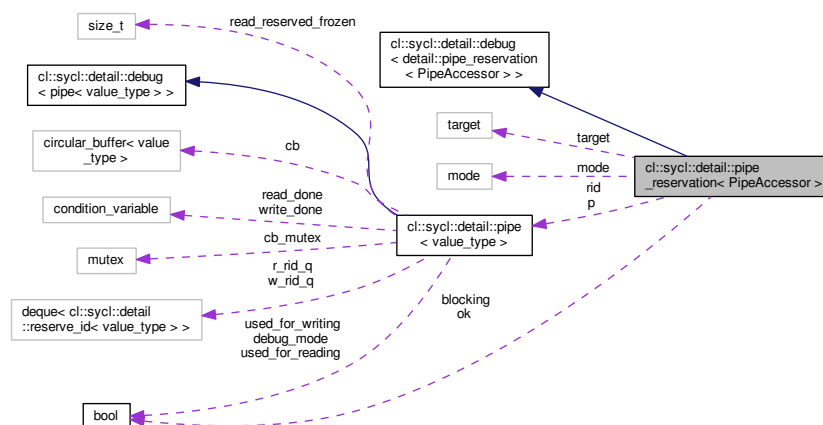
The implementation of the pipe reservation station.

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](#)

### 8.1.2.15.1 Member Typedef Documentation

#### 8.1.2.15.1.1 accessor\_type

```
template<typename PipeAccessor>
using cl::sycl::detail::pipe\_reservation< PipeAccessor >::accessor_type = PipeAccessor [private]
```

Definition at line 35 of file [pipe\\_reservation.hpp](#).

#### 8.1.2.15.1.2 const\_iterator

```
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](#).

#### 8.1.2.15.1.3 iterator

```
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](#).

#### 8.1.2.15.1.4 reference

```
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](#).

#### 8.1.2.15.1.5 value\_type

```
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](#).

## 8.1.2.15.2 Constructor &amp; Destructor Documentation

## 8.1.2.15.2.1 pipe\_reservation() [1/4]

```
template<typename PipeAccessor>
cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation (
    detail::pipe< value_type > & p,
    std::size_t s ) [inline]
```

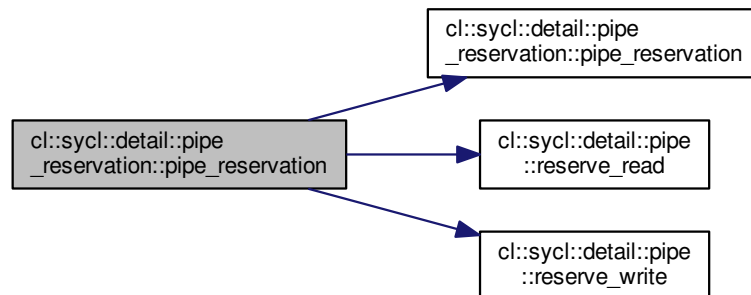
Create a pipe reservation station that reserves the pipe itself.

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:



## 8.1.2.15.2.2 pipe\_reservation() [2/4]

```
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.

#### 8.1.2.15.2.3 pipe\_reservation() [3/4]

```
template<typename PipeAccessor>
cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation (
    pipe_reservation< PipeAccessor > && orig ) [inline]
```

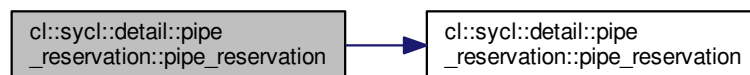
Only a move constructor is required to move it into the shared\_ptr.

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:



#### 8.1.2.15.2.4 pipe\_reservation() [4/4]

```
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:





## 8.1.2.15.2.5 ~pipe\_reservation()

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

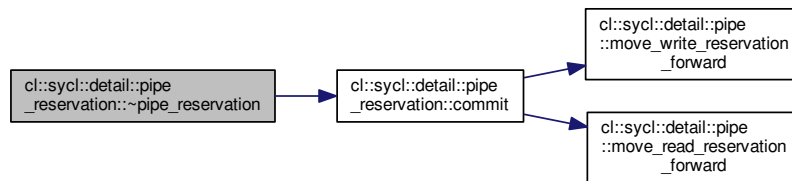
An implicit commit is made in the destructor.

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:



## 8.1.2.15.3 Member Function Documentation

## 8.1.2.15.3.1 assume\_validity()

```
template<typename PipeAccessor>
void cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity ( ) [inline]
```

Test that the reservation is in a usable state.

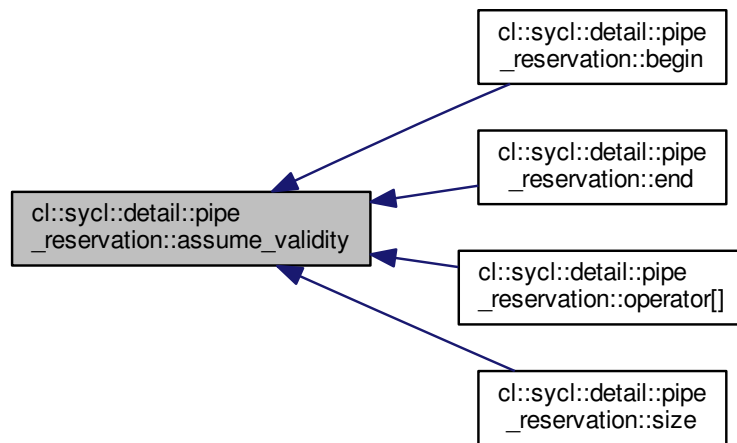
**Todo** Throw exception instead

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:



#### 8.1.2.15.3.2 begin()

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

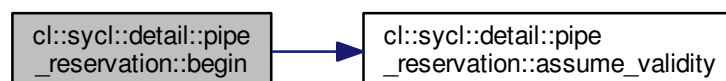
Start of the reservation area.

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:



## 8.1.2.15.3.3 commit()

```
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

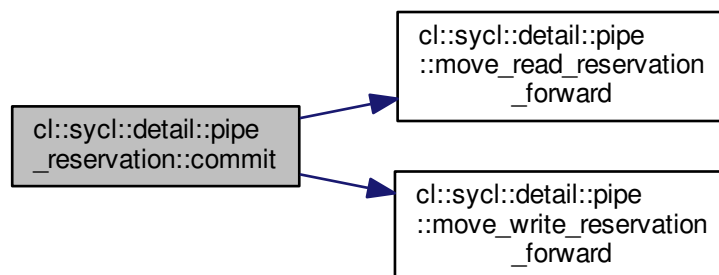
Definition at line 170 of file `pipe_reservation.hpp`.

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`.

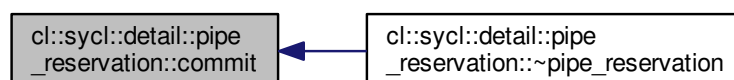
Referenced by `cl::sycl::detail::pipe_reservation< PipeAccessor >::~~pipe_reservation()`.

```
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:



#### 8.1.2.15.3.4 end()

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

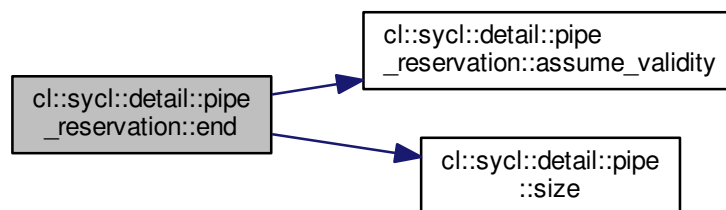
Past the end of the reservation area.

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:



#### 8.1.2.15.3.5 operator bool()

```
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     }
```

## 8.1.2.15.3.6 operator[]()

```
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:



## 8.1.2.15.3.7 size()

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

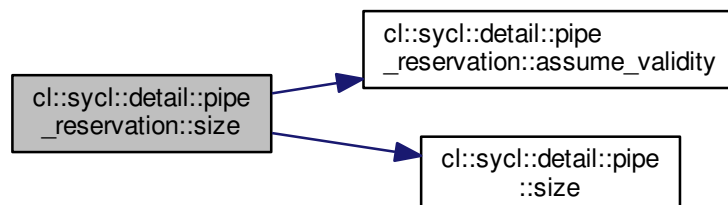
Get the number of elements in the reservation station.

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:



#### 8.1.2.15.4 Member Data Documentation

##### 8.1.2.15.4.1 blocking

```
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](#).

##### 8.1.2.15.4.2 mode

```
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](#).

##### 8.1.2.15.4.3 ok

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

##### 8.1.2.15.4.4 p

```
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](#).

## 8.1.2.15.4.5 rid

```
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](#).

## 8.1.2.15.4.6 target

```
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](#).

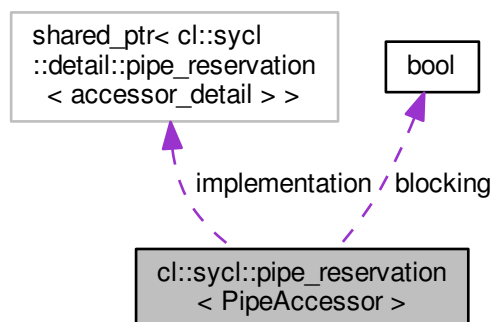
## 8.1.2.16 struct cl::sycl::pipe\_reservation

```
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](#)

#### 8.1.2.16.1 Member Typedef Documentation

##### 8.1.2.16.1.1 [accessor\\_detail](#)

```
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](#).

##### 8.1.2.16.1.2 [accessor\\_type](#)

```
template<typename PipeAccessor >
using cl::sycl::pipe\_reservation< PipeAccessor >::accessor\_type = PipeAccessor
```

Definition at line [31](#) of file [pipe\\_reservation.hpp](#).

##### 8.1.2.16.1.3 [const\\_iterator](#)

```
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](#).

##### 8.1.2.16.1.4 [const\\_pointer](#)

```
template<typename PipeAccessor >
using cl::sycl::pipe\_reservation< PipeAccessor >::const\_pointer = const value\_type*
```

Definition at line [40](#) of file [pipe\\_reservation.hpp](#).

#### 8.1.2.16.1.5 const\_reference

```
template<typename PipeAccessor >  
using cl::sycl::pipe_reservation< PipeAccessor >::const_reference = const value_type&
```

Definition at line 38 of file [pipe\\_reservation.hpp](#).

#### 8.1.2.16.1.6 const\_reverse\_iterator

```
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](#).

#### 8.1.2.16.1.7 difference\_type

```
template<typename PipeAccessor >  
using cl::sycl::pipe_reservation< PipeAccessor >::difference_type = ptrdiff_t
```

Definition at line 42 of file [pipe\\_reservation.hpp](#).

#### 8.1.2.16.1.8 iterator

```
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](#).

#### 8.1.2.16.1.9 pointer

```
template<typename PipeAccessor >  
using cl::sycl::pipe_reservation< PipeAccessor >::pointer = value_type*
```

Definition at line 39 of file [pipe\\_reservation.hpp](#).

#### 8.1.2.16.1.10 reference

```
template<typename PipeAccessor >  
using cl::sycl::pipe_reservation< PipeAccessor >::reference = value_type&
```

Definition at line 37 of file [pipe\\_reservation.hpp](#).

**8.1.2.16.1.11 reverse\_iterator**

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::reverse_iterator = std::reverse_iterator<iterator>
```

Definition at line 47 of file [pipe\\_reservation.hpp](#).

**8.1.2.16.1.12 size\_type**

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::size_type = std::size_t
```

Definition at line 41 of file [pipe\\_reservation.hpp](#).

**8.1.2.16.1.13 value\_type**

```
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](#).

**8.1.2.16.2 Constructor & Destructor Documentation****8.1.2.16.2.1 pipe\_reservation() [1/3]**

```
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.

## 8.1.2.16.2.2 pipe\_reservation() [2/3]

```
template<typename PipeAccessor >
cl::sycl::pipe_reservation< PipeAccessor >::pipe_reservation (
    accessor_type & accessor,
    std::size_t s ) [inline]
```

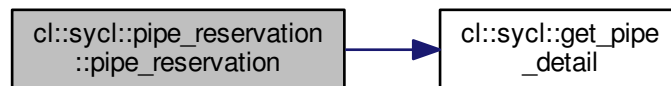
Create a [pipe\\_reservation](#) for an accessor and a number of elements.

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:



## 8.1.2.16.2.3 pipe\_reservation() [3/3]

```
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 {}
```

## 8.1.2.16.3 Member Function Documentation

8.1.2.16.3.1 `begin()`

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

Get an iterator on the first element of the reservation station.

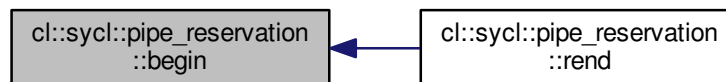
Definition at line 119 of file `pipe_reservation.hpp`.

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:

8.1.2.16.3.2 `cbegin()`

```
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:



#### 8.1.2.16.3.3 cend()

```
template<typename PipeAccessor >
const_iterator cl::sycl::pipe_reservation< PipeAccessor >::cend ( ) const [inline]
```

Build a constant iterator past the end of the reservation station.

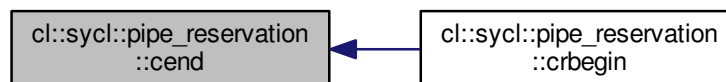
Definition at line 137 of file [pipe\\_reservation.hpp](#).

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:



#### 8.1.2.16.3.4 commit()

```
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     }
```

8.1.2.16.3.5 `crbegin()`

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

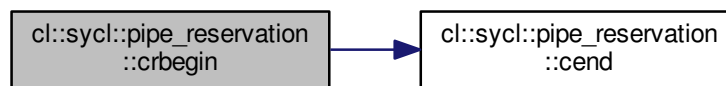
Get a constant reverse iterator on the last element of the reservation station.

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:

8.1.2.16.3.6 `crend()`

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

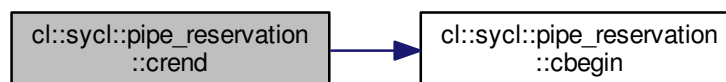
Get a constant reverse iterator on the first element past the end of the reservation station.

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:



#### 8.1.2.16.3.7 end()

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

Get an iterator past the end of the reservation station.

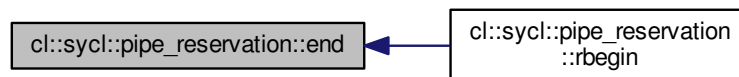
Definition at line 125 of file [pipe\\_reservation.hpp](#).

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:



#### 8.1.2.16.3.8 operator bool()

```
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 }
```



8.1.2.16.3.9 `operator[]()`

```
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 }
```

8.1.2.16.3.10 `rbegin()`

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

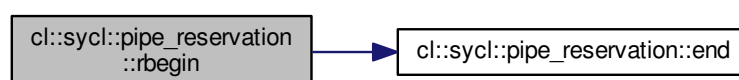
Get a reverse iterator on the last element of the reservation station.

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:



#### 8.1.2.16.3.11 `rend()`

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

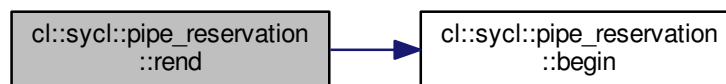
Get a reverse iterator on the first element past the end of the reservation station.

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:



#### 8.1.2.16.3.12 `size()`

```
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 }
```

#### 8.1.2.16.4 Member Data Documentation

## 8.1.2.16.4.1 blocking

```
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](#).

## 8.1.2.16.4.2 implementation

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

## 8.1.2.17 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.

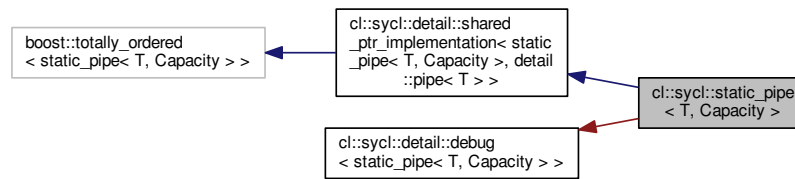
Compared to a normal pipe, a [static\\_pipe](#) takes a constexpr size and is expected to be declared in a compile-unit static context so the compiler can generate everything at compile time.

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.

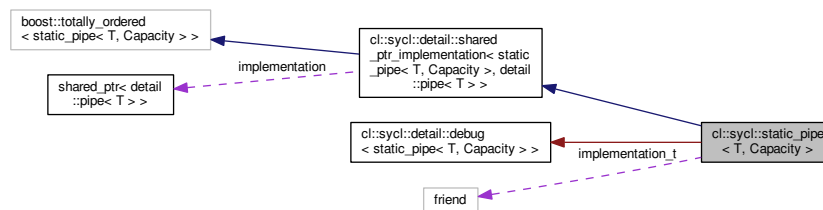
It is not directly mapped to the OpenCL program-scoped pipe because in SYCL there is not this concept of separated program. But the SYCL device compiler is expected to generate some OpenCL program(s) with program-scoped pipes when a SYCL static-scoped pipe is used. These details are implementation defined.

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` = `typename static_pipe::shared_ptr_implementation`

## Private Attributes

- friend `implementation_t`

## Additional Inherited Members

### 8.1.2.17.1 Member Typedef Documentation

#### 8.1.2.17.1.1 implementation\_t

```
template<typename T , std::size_t Capacity>
using cl::sycl::static_pipe< T, Capacity >::implementation_t = typename static_pipe::shared_ptr_implementation [private]
```

Definition at line 58 of file [static\\_pipe.hpp](#).

#### 8.1.2.17.1.2 value\_type

```
template<typename T , std::size_t Capacity>
using cl::sycl::static_pipe< T, Capacity >::value_type = T
```

The STL-like types.

Definition at line 69 of file [static\\_pipe.hpp](#).

### 8.1.2.17.2 Constructor & Destructor Documentation

#### 8.1.2.17.2.1 static\_pipe()

```
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 73 of file [static\\_pipe.hpp](#).

References [cl::sycl::access::pipe](#).

```
00074      : implementation_t { new detail::pipe<T> { Capacity } } { }
```

### 8.1.2.17.3 Member Function Documentation

#### 8.1.2.17.3.1 capacity()

```
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 102 of file [static\\_pipe.hpp](#).

```
00102                                     {
00103     return Capacity;
00104 }
```

#### 8.1.2.17.3.2 get\_access()

```
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 89 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](#).

```

00089                                     {
00090     static_assert(Target == access::target::pipe
00091                   || Target == access::target::blocking_pipe,
00092                   "get_access(handler) with pipes can only deal with "
00093                   "access::pipe or access::blocking_pipe");
00094     return { implementation, command_group_handler };
00095 }

```

## 8.1.2.17.4 Member Data Documentation

## 8.1.2.17.4.1 implementation\_t

```

template<typename T , std::size_t Capacity>
friend cl::sycl::static_pipe< T, Capacity >::implementation_t [private]

```

Definition at line 64 of file [static\\_pipe.hpp](#).

## 8.1.3 Typedef Documentation

## 8.1.3.1 buffer\_allocator

```

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

#include <include/CL/sycl/allocator.hpp>

```

The allocator objects give the programmer some control on how the memory is allocated inside SYCL.

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

The allocator used for the `buffer` inside SYCL

Just use the default allocator for now.

Reuse the C++ default allocator.

Definition at line 30 of file [allocator.hpp](#).

### 8.1.3.2 image\_allocator

```
template<typename T >
using cl::sycl::image_allocator = typedef std::allocator<T>

#include <include/CL/sycl/allocator.hpp>
```

The allocator used for the image inside SYCL.

Just use the default allocator for now.

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

### 8.1.3.3 map\_allocator

```
template<typename T >
using cl::sycl::map_allocator = typedef std::allocator<T>

#include <include/CL/sycl/allocator.hpp>
```

The allocator used to map the memory at the same place.

Just use the default allocator for now.

**Todo** : implement and clarify the specification. It looks like it is not really an allocator according the current spec

Definition at line 49 of file [allocator.hpp](#).

## 8.1.4 Function Documentation

### 8.1.4.1 add\_buffer\_to\_task()

```
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]

#include <include/CL/sycl/buffer/detail/buffer_base.hpp>
```

Referenced by [cl::sycl::detail::buffer\\_base::add\\_to\\_task\(\)](#).

Here is the caller graph for this function:





8.1.4.2 `buffer_add_to_task()`

```
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>`

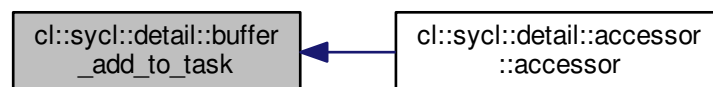
**Todo** To remove with some refactoring

Definition at line 436 of file `buffer.hpp`.

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

```
00438     {
00439     return buf->add_to_task(command_group_handler, is_write_mode);
00440 }
```

Here is the caller graph for this function:

8.1.4.3 `get_pipe_detail()`

```
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 478 of file `accessor.hpp`.

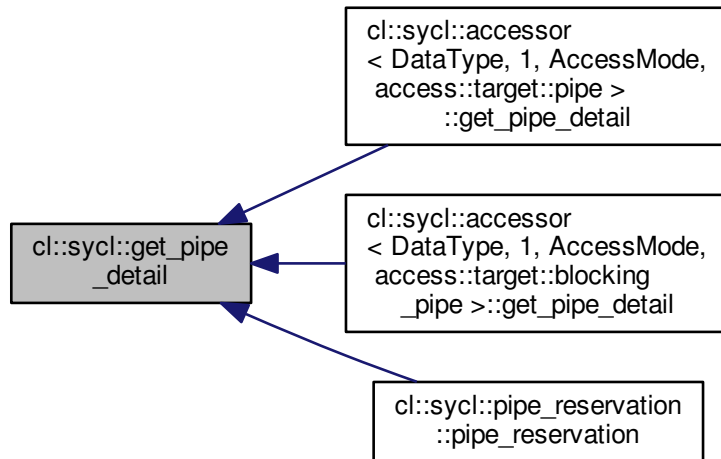
Referenced by `cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >::get_pipe_detail()`, `cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >::get_pipe_detail()`, and `cl::sycl::pipe_reservation< PipeAccessor >::pipe_reservation()`.

```

00478                                     {
00479     return a.get_pipe_detail();
00480 }

```

Here is the caller graph for this function:



#### 8.1.4.4 waiter()

```

template<typename T , int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↵
const_t<T>>>
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 79 of file `buffer_waiter.hpp`.

Referenced by `cl::sycl::buffer< T, Dimensions, Allocator >::buffer()`.

```

00079                                     {
00080     return new buffer_waiter<T, Dimensions, Allocator> { b };
00081 }

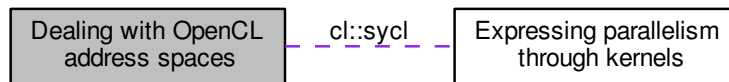
```

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::ocl\\_type< T, AS >](#)  
Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device. [More...](#)
- struct [cl::sycl::detail::ocl\\_type< T, constant\\_address\\_space >](#)  
Add an attribute for `__constant` address space. [More...](#)
- struct [cl::sycl::detail::ocl\\_type< T, generic\\_address\\_space >](#)  
Add an attribute for `__generic` address space. [More...](#)
- struct [cl::sycl::detail::ocl\\_type< T, global\\_address\\_space >](#)  
Add an attribute for `__global` address space. [More...](#)
- struct [cl::sycl::detail::ocl\\_type< T, local\\_address\\_space >](#)  
Add an attribute for `__local` address space. [More...](#)
- struct [cl::sycl::detail::ocl\\_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::constant\\_ptr](#) = [constant](#)< T \* >  
*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::global\\_ptr](#) = [global](#)< T \* >  
*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::local\\_ptr](#) = [local](#)< T \* >  
*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 T >  
 using [cl::sycl::private\\_ptr](#) = [priv](#)< T \* >  
*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::ocl\_type

```
template<typename T, address_space AS>  
struct cl::sycl::detail::ocl_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 type

```
template<typename T, address_space AS>  
using cl::sycl::detail::ocl_type< T, AS >::type = T
```

Definition at line 28 of file [address\\_space.hpp](#).

#### 8.2.2.2 struct cl::sycl::detail::ocl\_type< T, constant\_address\_space >

```
template<typename T>  
struct cl::sycl::detail::ocl_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 type

```
template<typename T >
using cl::sycl::detail::ocl_type< T, constant_address_space >::type = T
```

Definition at line 40 of file [address\\_space.hpp](#).

#### 8.2.2.3 struct cl::sycl::detail::ocl\_type< T, generic\_address\_space >

```
template<typename T>
struct cl::sycl::detail::ocl_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 type

```
template<typename T >
using cl::sycl::detail::ocl_type< T, generic_address_space >::type = T
```

Definition at line 52 of file [address\\_space.hpp](#).

#### 8.2.2.4 struct cl::sycl::detail::ocl\_type< T, global\_address\_space >

```
template<typename T>
struct cl::sycl::detail::ocl_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 type

```
template<typename T >
using cl::sycl::detail::ocl_type< T, global_address_space >::type = T
```

Definition at line 64 of file [address\\_space.hpp](#).

## 8.2.2.5 struct cl::sycl::detail::ocl\_type&lt; T, local\_address\_space &gt;

```
template<typename T>
struct cl::sycl::detail::ocl_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 type

```
template<typename T >
using cl::sycl::detail::ocl_type< T, local_address_space >::type = T
```

Definition at line 76 of file [address\\_space.hpp](#).

## 8.2.2.6 struct cl::sycl::detail::ocl\_type&lt; T, private\_address\_space &gt;

```
template<typename T>
struct cl::sycl::detail::ocl_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 type

```
template<typename T >
using cl::sycl::detail::ocl_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 >
```

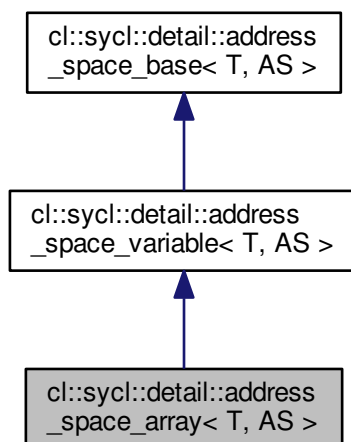
Implementation of an array variable with an OpenCL address space.

## 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 |

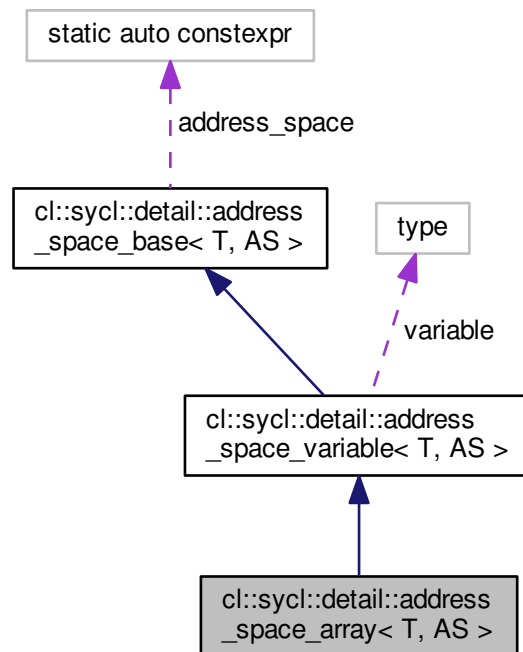
Definition at line 95 of file [address\\_space.hpp](#).

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\_extents\_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 super

```
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 311 of file [address\\_space.hpp](#).

#### 8.2.2.7.2 Constructor & Destructor Documentation

##### 8.2.2.7.2.1 address\_space\_array() [1/2]

```
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 319 of file [address\\_space.hpp](#).

```
00319         {
00320     std::copy(std::begin(array), std::end(array), std::begin(super::variable));
00321 };
```

##### 8.2.2.7.2.2 address\_space\_array() [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 328 of file [address\\_space.hpp](#).

```
00328         {
00329     std::copy(std::begin(list), std::end(list), std::begin(super::variable));
00330 };
```

#### 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.

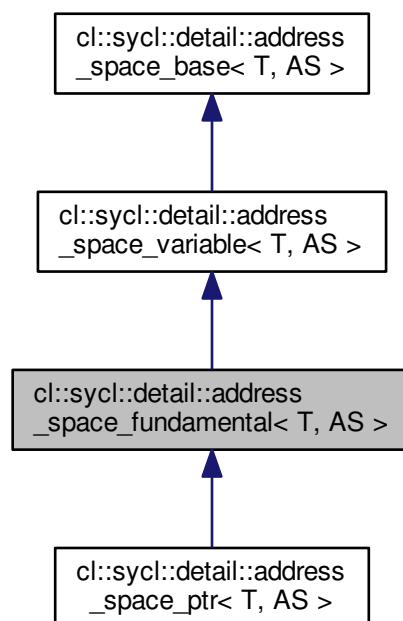
## Parameters

|           |   |
|-----------|---|
| <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 |

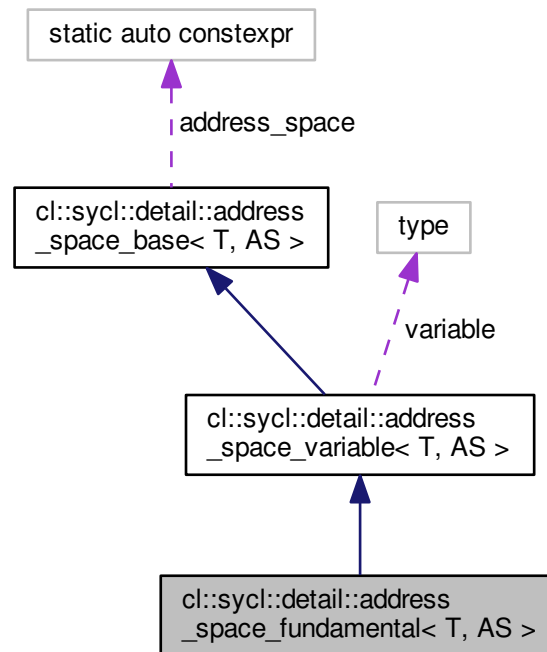
**Todo** Verify/improve to deal with const/volatile?

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 super

```
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 &amp; Destructor Documentation

## 8.2.2.8.2.1 address\_space\_fundamental() [1/2]

```
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 address\_space\_fundamental() [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 |

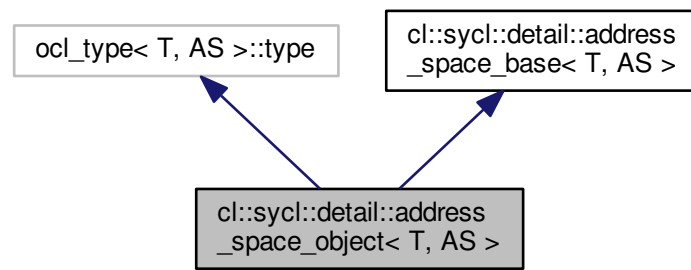
The class implementation is just inheriting of *T* so that all methods and non-member operators on *T* work also on `address_space_object<T>`

**Todo** Verify/improve to deal with const/volatile?

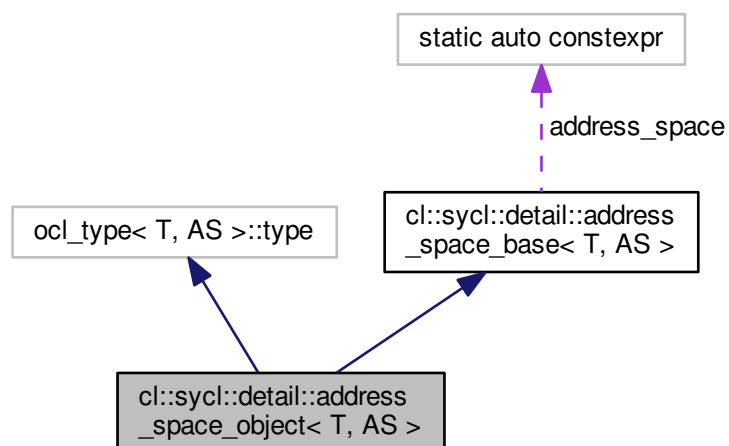
**Todo** what about *T* having some final methods?

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 `opengl_type` = typename `ocl_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 `opengl_type`

```
template<typename T , address_space AS>
using cl::sycl::detail::address_space_object< T, AS >::opengl_type = typename ocl_type<T,
AS>::type
```

Store the base type of the object with OpenCL address space modifier.

**Todo** Add to the specification

Definition at line 356 of file `address_space.hpp`.

## 8.2.2.9.2 Constructor &amp; Destructor Documentation

8.2.2.9.2.1 `address_space_object()`

```
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 367 of file `address_space.hpp`.

```
00367 : opengl_type(v) { }
```

### 8.2.2.9.3 Member Function Documentation

#### 8.2.2.9.3.1 operator opengl\_type &()

```
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 375 of file [address\\_space.hpp](#).

```
00375 { 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

|                |                     |
|----------------|---------------------|
| <code>T</code> | is the pointer type |
|----------------|---------------------|

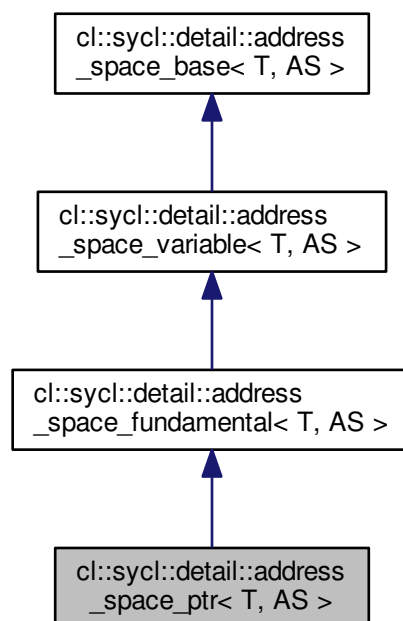
Note that if `T` is not a pointer type, it is an error.

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

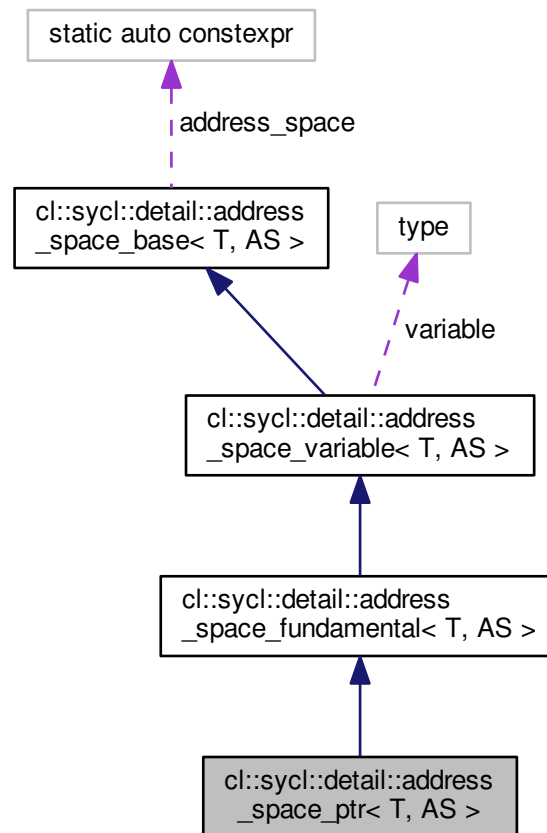
Definition at line 104 of file [address\\_space.hpp](#).



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.*
- using `pointer_t` = `typename super::address_space_fundamental::type`
- using `reference_t` = `typename std::remove_pointer_t< pointer_t > &`

### 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 `pointer_t`

```
template<typename T, address_space AS>
using cl::sycl::detail::address_space_ptr< T, AS >::pointer_t = typename super::address_↵
space_fundamental::type
```

Definition at line 288 of file [address\\_space.hpp](#).

8.2.2.10.1.2 `reference_t`

```
template<typename T, address_space AS>
using cl::sycl::detail::address_space_ptr< T, AS >::reference_t = typename std::remove_↵
pointer_t<pointer_t>&
```

Definition at line 289 of file [address\\_space.hpp](#).

8.2.2.10.1.3 `super`

```
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 &amp; Destructor Documentation

8.2.2.10.2.1 `address_space_ptr()`

```
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]
```

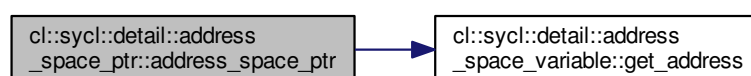
Allow initialization of a pointer type from the address of an element with the same type and address space.

Definition at line 294 of file [address\\_space.hpp](#).

References [cl::sycl::detail::address\\_space\\_variable< T, AS >::get\\_address\(\)](#).

```
00295      : 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.

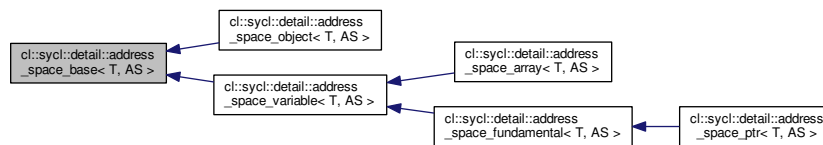
## Parameters

|           |   |
|-----------|---|
| <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 |

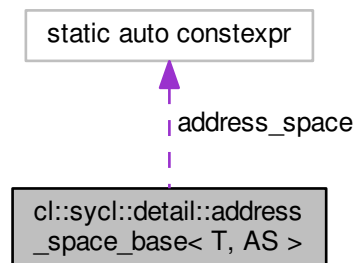
**Todo** Verify/improve to deal with const/volatile?

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 `opencl_type` = `typename ocl_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 opengl\_type

```
template<typename T , address_space AS>
using cl::sycl::detail::address_space_base< T, AS >::opengl_type = typename ocl_type<T, AS>↵
::type
```

Store the base type of the object with OpenCL address space modifier.

**Todo** Add to the specification

Definition at line 146 of file [address\\_space.hpp](#).

#### 8.2.2.11.1.2 type

```
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 address\_space

```
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 >
```

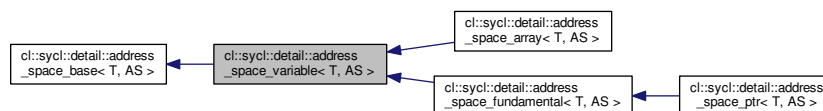
Implementation of a variable with an OpenCL address space.

## Parameters

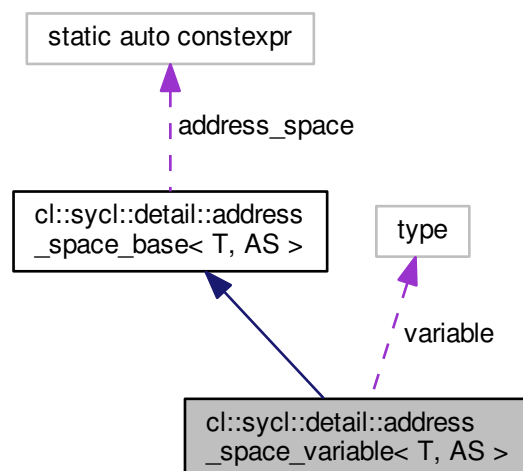
|           |   |
|-----------|---|
| <i>T</i>  | is the type of the basic object to be created |
| <i>AS</i> | is the address space to place the object into |

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 ocl_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 opcnl\\_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.*
- [opcnl\\_type \\* get\\_address](#) ()  
*Return the address of the value to implement pointers.*

## Protected Attributes

- [opcnl\\_type](#) variable

## Additional Inherited Members

### 8.2.2.12.1 Member Typedef Documentation

#### 8.2.2.12.1.1 opcnl\_type

```
template<typename T , address_space AS>
using cl::sycl::detail::address_space_variable< T, AS >::opcnl_type = typename ocl_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 super

```
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 `address_space_variable()` [1/2]

```
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 `address_space_variable()` [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 `get_address()`

```
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 operator opocl\_type &()

```
template<typename T , address_space AS>
cl::sycl::detail::address_space_variable< T, AS >::operator opocl_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 `opocl_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 variable

```
template<typename T , address_space AS>
opocl_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 addr\_space

```
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 constant

```
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 constant\_ptr

```
template<typename T >
using cl::sycl::constant_ptr = typedef constant<T*>

#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 63 of file [address\\_space.hpp](#).

### 8.2.3.4 generic

```
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 71 of file [address\\_space.hpp](#).

### 8.2.3.5 global

```
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 79 of file [address\\_space.hpp](#).

### 8.2.3.6 global\_ptr

```
template<typename T >
using cl::sycl::global_ptr = typedef global<T*>

#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 88 of file [address\\_space.hpp](#).

### 8.2.3.7 local

```
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 96 of file [address\\_space.hpp](#).

## 8.2.3.8 local\_ptr

```
template<typename T >
using cl::sycl::local_ptr = typedef local<T*>

#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 104 of file [address\\_space.hpp](#).

## 8.2.3.9 multi\_ptr

```
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 132 of file [address\\_space.hpp](#).

## 8.2.3.10 priv

```
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

|          |                           |
|----------|---------------------------|
| <i>T</i> | is the type of the object |
|----------|---------------------------|

Definition at line 112 of file [address\\_space.hpp](#).

#### 8.2.3.11 private\_ptr

```
template<typename T >
using cl::sycl::private_ptr = typedef priv<T*>

#include <include/CL/sycl/address_space.hpp>
```

Declare a variable to be in the OpenCL private address space.

##### Parameters

|          |                           |
|----------|---------------------------|
| <i>T</i> | is the type of the object |
|----------|---------------------------|

Definition at line 120 of file [address\\_space.hpp](#).

### 8.2.4 Enumeration Type Documentation

#### 8.2.4.1 address\_space

```
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 make\_multi()

```
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

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

**Todo** Implement the case with a plain pointer

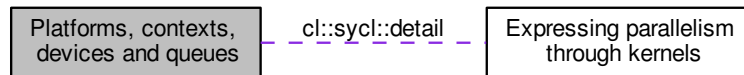
Definition at line 142 of file `address_space.hpp`.

```
00142                                     {
00143     return pointer;
00144 }
```

## 8.3 Platforms, contexts, devices and queues

SYCL host context.

Collaboration diagram for Platforms, contexts, devices and queues:



### Namespaces

- [cl::sycl::detail](#)
- [cl::sycl::info](#)

### Classes

- class [cl::sycl::detail::context](#)
- class [cl::sycl::detail::host\\_context](#)
- 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...](#)*



## 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.*
- using `cl::sycl::info::gl_context_interop` = `bool`
- using `cl::sycl::info::device_fp_config` = `unsigned int`
- using `cl::sycl::info::device_exec_capabilities` = `unsigned int`
- using `cl::sycl::info::device_queue_properties` = `unsigned int`

## Enumerations

- enum `cl::sycl::info::context` : `int` { `cl::sycl::info::context::reference_count`, `cl::sycl::info::context::num_devices`, `cl::sycl::info::context::devices`, `cl::sycl::info::context::gl_interop` }  
*Context information descriptors.*
- 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_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`,



- `template<>`  
`auto cl::sycl::device::get_info< info::device::name > () const`
- `template<>`  
`auto cl::sycl::device::get_info< info::device::profile > () const`
- `static vector_class< device > cl::sycl::device::get_devices (info::device_type device_type=info::device_↵  
type::all) TRISYCL_WEAK_ATTRIB_SUFFIX`  
*Return a list of all available devices.*
- `vector_class< cl::sycl::device > cl::sycl::detail::host_platform::get_devices (info::device_type device_type)`  
`const override`  
*Get all the available devices for the host platform.*
- `vector_class< cl::sycl::device > cl::sycl::detail::opencl_platform::get_devices (info::device_type device_type)`  
`const override`  
*Get all the available devices for this OpenCL platform.*

## Variables

- `TRISYCL_WEAK_ATTRIB_PREFIX detail::cache< cl_context, detail::opencl_context > opencl_context↵  
::cache cl::sycl::detail::TRISYCL_WEAK_ATTRIB_SUFFIX`

### 8.3.1 Detailed Description

SYCL host context.

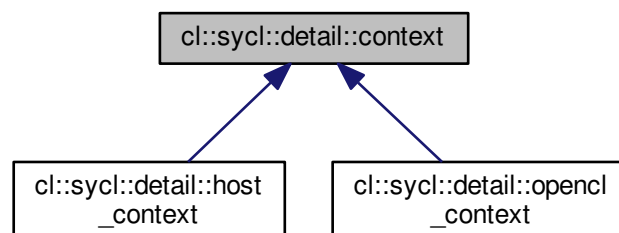
**Todo** The implementation is quite minimal for now. :-)

### 8.3.2 Class Documentation

#### 8.3.2.1 class `cl::sycl::detail::context`

Definition at line 24 of file `context.hpp`.

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



## Public Member Functions

- virtual `cl_context` `get ()` const =0  
Return the underlying `cl_context` of the `cl::sycl::context`.
- virtual `boost::compute::context &` `get_boost_compute ()`=0  
Return the underlying `boost::compute::context` of the `cl::sycl::context`.
- virtual `boost::compute::command_queue &` `get_boost_queue ()`=0  
Return the underlying `boost::compute::command_queue` associated with the context.
- virtual `bool` `is_host ()` const =0  
Returns true is the context is a SYCL host context.
- virtual `cl::sycl::platform` `get_platform ()` const =0  
Returns the SYCL platform that the context is initialized for.
- virtual `vector_class< cl::sycl::device >` `get_devices ()` const =0  
Returns the set of devices that are part of this context.
- virtual `~context ()`  
Virtual to call the real destructor.

## 8.3.2.1.1 Constructor &amp; Destructor Documentation

## 8.3.2.1.1.1 ~context()

```
virtual cl::sycl::detail::context::~~context ( ) [inline], [virtual]
```

Virtual to call the real destructor.

Definition at line 58 of file `context.hpp`.

```
00058 {}
```

## 8.3.2.1.2 Member Function Documentation

## 8.3.2.1.2.1 get()

```
virtual cl_context cl::sycl::detail::context::get ( ) const [pure virtual]
```

Return the underlying `cl_context` of the `cl::sycl::context`.

Implemented in `cl::sycl::detail::opencl_context`, and `cl::sycl::detail::host_context`.

## 8.3.2.1.2.2 get\_boost\_compute()

```
virtual boost::compute::context& cl::sycl::detail::context::get_boost_compute ( ) [pure virtual]
```

Return the underlying `boost::compute::context` of the `cl::sycl::context`.

Implemented in `cl::sycl::detail::opencl_context`, and `cl::sycl::detail::host_context`.

#### 8.3.2.1.2.3 `get_boost_queue()`

```
virtual boost::compute::command_queue& cl::sycl::detail::context::get_boost_queue ( ) [pure virtual]
```

Return the underlying `boost::compute::command_queue` associated with the context.

Implemented in [cl::sycl::detail::opencl\\_context](#), and [cl::sycl::detail::host\\_context](#).

#### 8.3.2.1.2.4 `get_devices()`

```
virtual vector_class<cl::sycl::device> cl::sycl::detail::context::get_devices ( ) const [pure virtual]
```

Returns the set of devices that are part of this context.

**Todo** virtual cannot be templated template <[info::context](#) Param> typename info::param\_traits<info::context, Param>::type get\_info() const = 0;

Implemented in [cl::sycl::detail::host\\_context](#), and [cl::sycl::detail::opencl\\_context](#).

#### 8.3.2.1.2.5 `get_platform()`

```
virtual cl::sycl::platform cl::sycl::detail::context::get_platform ( ) const [pure virtual]
```

Returns the SYCL platform that the context is initialized for.

Implemented in [cl::sycl::detail::opencl\\_context](#), and [cl::sycl::detail::host\\_context](#).

#### 8.3.2.1.2.6 `is_host()`

```
virtual bool cl::sycl::detail::context::is_host ( ) const [pure virtual]
```

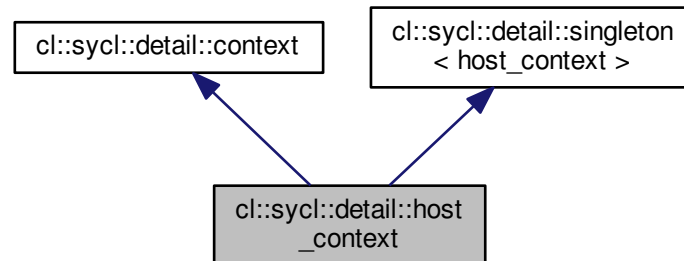
Returns true is the context is a SYCL host context.

Implemented in [cl::sycl::detail::host\\_context](#), and [cl::sycl::detail::opencl\\_context](#).

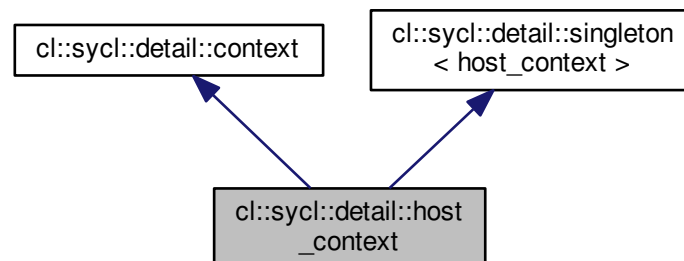
### 8.3.2.2 class `cl::sycl::detail::host_context`

Definition at line 32 of file [host\\_context.hpp](#).

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



Collaboration diagram for `cl::sycl::detail::host_context`:



#### Public Member Functions

- `cl_context` [get](#) () const override  
*Return the underlying `cl_context` of the `cl::sycl::context`.*
- `boost::compute::context` & [get\\_boost\\_compute](#) () override  
*Return the SYCL platform that the context is initialized for.*
- `boost::compute::command_queue` & [get\\_boost\\_queue](#) () override  
*Return the internal OpenCL queue that is associated to the host context.*
- `bool` [is\\_host](#) () const override  
*Return true since the context is a SYCL host context.*
- `cl::sycl::platform` [get\\_platform](#) () const override  
*Return the platform of the context.*
- `vector_class< cl::sycl::device >` [get\\_devices](#) () const override  
*Returns the set of devices that are part of this context.*

## Additional Inherited Members

### 8.3.2.2.1 Member Function Documentation

#### 8.3.2.2.1.1 get()

```
cl_context cl::sycl::detail::host_context::get ( ) const [inline], [override], [virtual]
```

Return the underlying `cl_context` of the `cl::sycl::context`.

This throws an error since there is no OpenCL context associated to the host device.

Implements `cl::sycl::detail::context`.

Definition at line 43 of file `host_context.hpp`.

```
00043                                     {
00044     throw non_cl_error("The host context has no OpenCL context");
00045 }
```

#### 8.3.2.2.1.2 get\_boost\_compute()

```
boost::compute::context& cl::sycl::detail::host_context::get_boost_compute ( ) [inline],
[override], [virtual]
```

Return the SYCL platform that the context is initialized for.

This throws an error since there is no `boost::compute` context associated to the host device.

Implements `cl::sycl::detail::context`.

Definition at line 53 of file `host_context.hpp`.

```
00053                                     {
00054     throw non_cl_error("The host context has no OpenCL context");
00055 }
```

### 8.3.2.2.1.3 get\_boost\_queue()

```
boost::compute::command_queue& cl::sycl::detail::host_context::get_boost_queue ( ) [inline],
[override], [virtual]
```

Return the internal OpenCL queue that is associated to the host context.

This throws an error since there is no `boost::compute::command_queue` associated to the host context.

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

Definition at line 64 of file [host\\_context.hpp](#).

Referenced by [cl::sycl::detail::buffer\\_base::sync\\_with\\_host\(\)](#).

```
00064                                     {
00065     throw non_cl_error("The host context cannot have an OpenCL queue");
00066 }
```

Here is the caller graph for this function:



### 8.3.2.2.1.4 get\_devices()

```
vector_class<cl::sycl::device> cl::sycl::detail::host_context::get_devices ( ) const [inline],
[override], [virtual]
```

Returns the set of devices that are part of this context.

It should only return the host device itself.

**Todo** To be implemented

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

Definition at line 107 of file [host\\_context.hpp](#).

```
00107                                     {
00108     // Return just the host device
00109     return { {} };
00110 }
```



#### 8.3.2.2.1.5 `get_platform()`

```
cl::sycl::platform cl::sycl::detail::host_context::get_platform ( ) const [inline], [override],  
[virtual]
```

Return the platform of the context.

Return synchronous errors via the SYCL exception class.

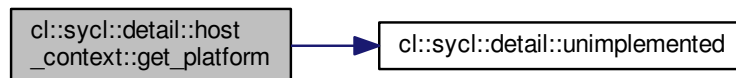
Implements [cl::sycl::detail::context](#).

Definition at line 80 of file [host\\_context.hpp](#).

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

```
00080                                     {  
00081     // Return the host platform  
00082     return {};  
00083 }
```

Here is the call graph for this function:



#### 8.3.2.2.1.6 `is_host()`

```
bool cl::sycl::detail::host_context::is_host ( ) const [inline], [override], [virtual]
```

Return true since the context is a SYCL host context.

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

Definition at line 71 of file [host\\_context.hpp](#).

```
00071                                     {  
00072     return true;  
00073 }
```

### 8.3.2.3 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.

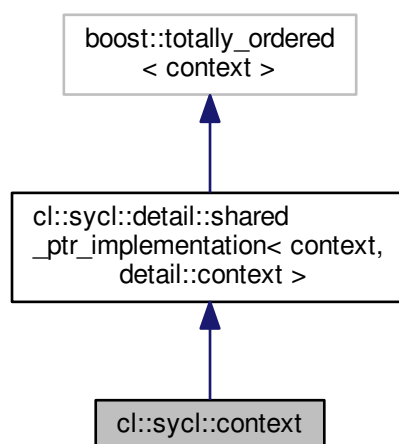
On destruction `clReleaseContext` is called.

The default context is the SYCL host context containing only the SYCL host device.

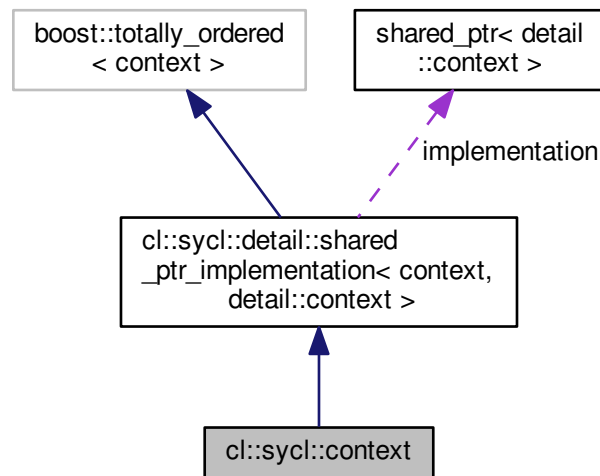
**Todo** The implementation is quite minimal for now.

Definition at line 48 of file [context.hpp](#).

Inheritance diagram for `cl::sycl::context`:



Collaboration diagram for `cl::sycl::context`:



#### 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)`  
Make a SYCL context from an OpenCL context.
- `context (const boost::compute::context &c, async_handler asyncHandler=nullptr)`  
Build a SYCL context from a Boost.Compute context.
- `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 constructor that chooses the context according the heuristics of the default selector.
- `cl_context get () const`  
Return the underlying `cl_context` object, after retaining the `cl_context`.
- `boost::compute::context &get_boost_compute () const`  
Return the underlying `boost::compute::context` of the `cl::sycl::context`.
- `boost::compute::command_queue &get_boost_queue () const`  
Return the internal queue that is associated to the context and used by triSYCL to move data between some different contexts for example.
- `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.*

#### Private Types

- using [implementation\\_t](#) = [detail::shared\\_ptr\\_implementation< context, detail::context >](#)

### Additional Inherited Members

#### 8.3.2.3.1 Member Typedef Documentation

##### 8.3.2.3.1.1 implementation\_t

```
using cl::sycl::context::implementation_t = detail::shared_ptr_implementation<context, detail::context> [private]
```

Definition at line 56 of file [context.hpp](#).

#### 8.3.2.3.2 Constructor & Destructor Documentation

##### 8.3.2.3.2.1 context() [1/8]

```
cl::sycl::context::context (
    async_handler asyncHandler ) [inline], [explicit]
```

Constructs a context object for SYCL host using an async\_handler for handling asynchronous errors.

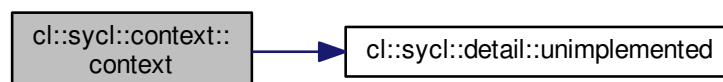
Note that the default case asyncHandler = nullptr is handled by the default constructor.

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

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

```
00069                                     {
00070     detail::unimplemented();
00071 }
```

Here is the call graph for this function:



**8.3.2.3.2 context()** [2/8]

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

Make a SYCL context from an OpenCL context.

The constructor executes a retain on the `cl_context`.

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

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

```
00084      : context { boost::compute::context { clContext }, asyncHandler } {}
```

**8.3.2.3.3 context()** [3/8]

```
cl::sycl::context::context (
    const boost::compute::context & c,
    async_handler asyncHandler = nullptr ) [inline]
```

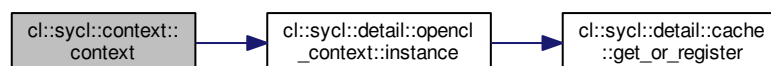
Build a SYCL context from a Boost.Compute context.

Definition at line 88 of file [context.hpp](#).

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

```
00090      : implementation_t { detail::opengl_context::instance(c
    ) } {}
```

Here is the call graph for this function:



**8.3.2.3.2.4 context()** [4/8]

```
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.

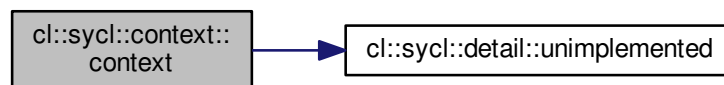
Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the [async\\_↔](#) handler, if provided.

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

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

```
00103                                     {
00104     detail::unimplemented\(\);
00105 }
```

Here is the call graph for this function:

**8.3.2.3.2.5 context()** [5/8]

```
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.

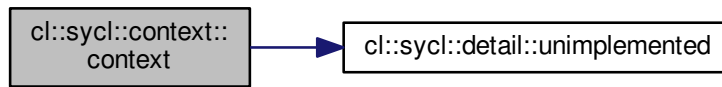
Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the [async\\_↔](#) handler, if provided.

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

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

```
00115                                     {
00116     detail::unimplemented\(\);
00117 }
```

Here is the call graph for this function:



#### 8.3.2.3.2.6 context() [6/8]

```

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.

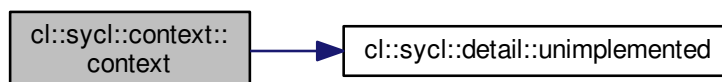
Definition at line 125 of file [context.hpp](#).

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

```

00127                                     {
00128     detail::unimplemented();
00129 }
  
```

Here is the call graph for this function:



**8.3.2.3.2.7 context()** [7/8]

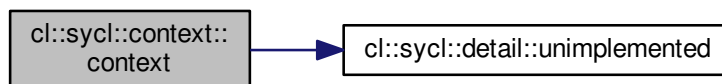
```
cl::sycl::context::context (
    const vector_class< device > & deviceList,
    info::gl_context_interop interopFlag,
    async_handler asyncHandler = nullptr ) [inline]
```

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

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

```
00142                                     {
00143     detail::unimplemented();
00144 }
```

Here is the call graph for this function:

**8.3.2.3.2.8 context()** [8/8]

```
cl::sycl::context::context ( ) [inline]
```

Default constructor that chooses the context according the heuristics of the default selector.

Return synchronous errors via the SYCL exception class.

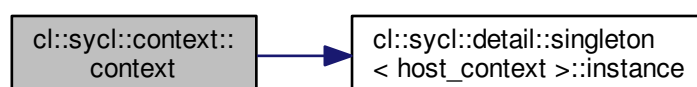
Get the default constructors back.

Definition at line 153 of file [context.hpp](#).

References [cl::sycl::detail::singleton< host\\_context >::instance\(\)](#).

```
00153 : implementation_t { detail::host_context::instance() } {}
```

Here is the call graph for this function:





## 8.3.2.3.3 Member Function Documentation

8.3.2.3.3.1 `get()`

```
cl_context cl::sycl::context::get ( ) const [inline]
```

Return the underlying `cl_context` object, after retaining the `cl_context`.

Retains a reference to the returned `cl_context` object.

Caller should release it when finished.

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

References [cl::sycl::detail::shared\\_ptr\\_implementation< context, detail::context >::implementation](#).

```
00164         {
00165         return implementation->get();
00166     }
```

8.3.2.3.3.2 `get_boost_compute()`

```
boost::compute::context& cl::sycl::context::get_boost_compute ( ) const [inline]
```

Return the underlying `boost::compute::context` of the [cl::sycl::context](#).

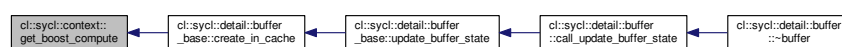
Definition at line 171 of file [context.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< context, detail::context >::implementation](#).

Referenced by [cl::sycl::detail::buffer\\_base::create\\_in\\_cache\(\)](#).

```
00171         {
00172         return implementation->get_boost_compute();
00173     }
```

Here is the caller graph for this function:



### 8.3.2.3.3.3 `get_boost_queue()`

```
boost::compute::command_queue& cl::sycl::context::get_boost_queue ( ) const [inline]
```

Return the internal queue that is associated to the context and used by triSYCL to move data between some different contests for example.

Definition at line 179 of file [context.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< context, detail::context >::implementation](#).

Referenced by [cl::sycl::detail::buffer\\_base::update\\_buffer\\_state\(\)](#).

```
00179                                     {
00180     return implementation->get_boost_queue();
00181 }
```

Here is the caller graph for this function:



### 8.3.2.3.3.4 `get_devices()`

```
vector_class<device> cl::sycl::context::get_devices ( ) const [inline]
```

Returns the set of devices that are part of this context.

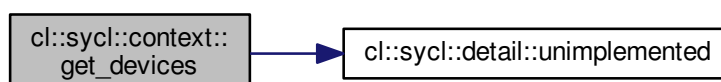
**Todo** To be implemented

Definition at line 202 of file [context.hpp](#).

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

```
00202                                     {
00203     detail::unimplemented();
00204     return {};
00205 }
```

Here is the call graph for this function:



8.3.2.3.3.5 `get_info()`

```
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.

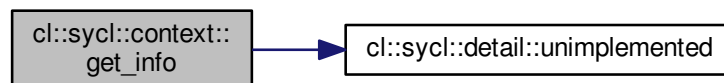
**Todo** To be implemented

Definition at line 213 of file `context.hpp`.

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

```
00213                                     {
00214     detail::unimplemented();
00215     return {};
00216 }
```

Here is the call graph for this function:

8.3.2.3.3.6 `get_platform()`

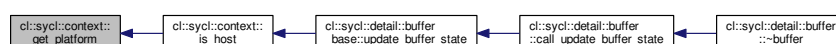
```
platform cl::sycl::context::get_platform ( )
```

Returns the SYCL platform that the context is initialized for.

**Todo** To be implemented

Referenced by `is_host()`.

Here is the caller graph for this function:



### 8.3.2.3.3.7 is\_host()

```
bool cl::sycl::context::is_host ( ) const [inline]
```

Specifies whether the context is in SYCL Host Execution Mode.

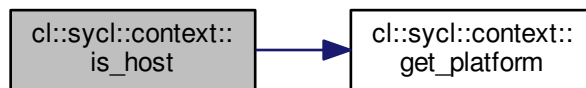
Definition at line 186 of file [context.hpp](#).

References [get\\_platform\(\)](#), and [cl::sycl::detail::shared\\_ptr\\_implementation< context, detail::context >::implementation](#).

Referenced by [cl::sycl::detail::buffer\\_base::update\\_buffer\\_state\(\)](#).

```
00186         {
00187     return implementation->is_host();
00188     }
```

Here is the call graph for this function:



Here is the caller graph for this function:

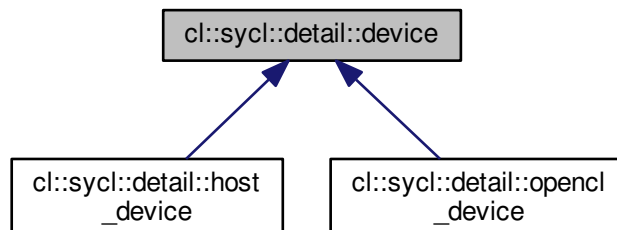


### 8.3.2.4 class cl::sycl::detail::device

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 `boost::compute::device & get_boost_compute ()`=0  
*Return the underlying Boost.Compute device, if any.*
- 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 ()`

## 8.3.2.4.1 Constructor &amp; Destructor Documentation

## 8.3.2.4.1.1 ~device()

```
virtual cl::sycl::detail::device::~device ( ) [inline], [virtual]
```

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

```
00070 {}
```

## 8.3.2.4.2 Member Function Documentation

## 8.3.2.4.2.1 get()

```
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](#).

## 8.3.2.4.2.2 get\_boost\_compute()

```
virtual boost::compute::device& cl::sycl::detail::device::get_boost_compute ( ) [pure virtual]
```

Return the underlying Boost.Compute device, if any.

Implemented in [cl::sycl::detail::host\\_device](#), and [cl::sycl::detail::opencl\\_device](#).

**8.3.2.4.2.3 get\_platform()**

```
virtual cl::sycl::platform cl::sycl::detail::device::get_platform ( ) const [pure virtual]
```

Return the platform of device.

Implemented in [cl::sycl::detail::host\\_device](#), and [cl::sycl::detail::opencl\\_device](#).

**8.3.2.4.2.4 has\_extension()**

```
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::host\\_device](#), and [cl::sycl::detail::opencl\\_device](#).

**8.3.2.4.2.5 is\_accelerator()**

```
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::host\\_device](#), and [cl::sycl::detail::opencl\\_device](#).

**8.3.2.4.2.6 is\_cpu()**

```
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::host\\_device](#), and [cl::sycl::detail::opencl\\_device](#).

**8.3.2.4.2.7 is\_gpu()**

```
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::host\\_device](#), and [cl::sycl::detail::opencl\\_device](#).

#### 8.3.2.4.2.8 `is_host()`

```
virtual bool cl::sycl::detail::device::is_host ( ) const [pure virtual]
```

Return true if the device is a SYCL host device.

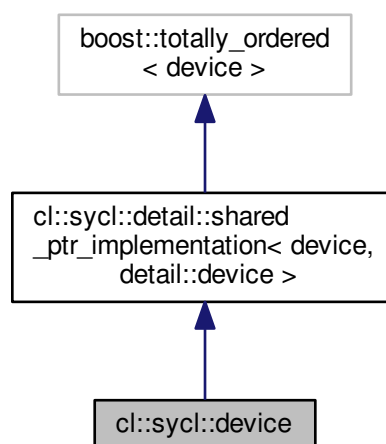
Implemented in [cl::sycl::detail::host\\_device](#), and [cl::sycl::detail::opencl\\_device](#).

#### 8.3.2.5 `class cl::sycl::device`

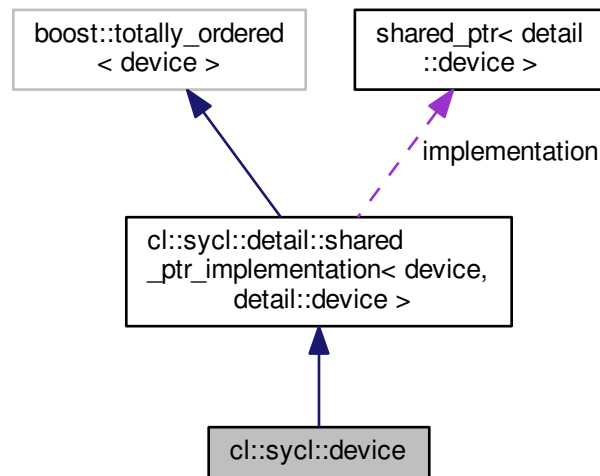
SYCL device.

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.
- `boost::compute::device get_boost_compute () const`  
Return the underlying Boost.Compute device if it is an OpenCL device.
- `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) TRISYCL\_  
\_WEAK\_ATTRIB\_SUFFIX  
 Return a list of all available devices.`

#### Private Types

- using `implementation\_t = detail::shared\_ptr\_implementation< device, detail::device >`

### Additional Inherited Members

#### 8.3.2.5.1 Member Typedef Documentation

##### 8.3.2.5.1.1 `implementation_t`

```
using cl::sycl::device::implementation\_t = detail::shared\_ptr\_implementation<device, detail::device> [private]
```

Definition at line 48 of file [device.hpp](#).

#### 8.3.2.5.2 Constructor & Destructor Documentation

##### 8.3.2.5.2.1 `device()` [1/4]

```
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:



**8.3.2.5.2.2 device()** [2/4]

```
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 } } {}
```

**8.3.2.5.2.3 device()** [3/4]

```
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.

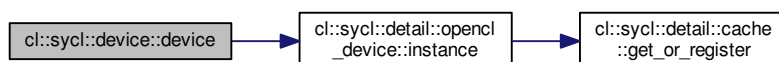
Return synchronous errors via the SYCL exception class.

Definition at line 79 of file [device.hpp](#).

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

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

Here is the call graph for this function:



## 8.3.2.5.2.4 device() [4/4]

```
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.

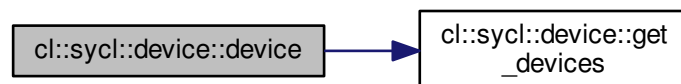
**Todo** Make it non-explicit in the specification?

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:



## 8.3.2.5.3 Member Function Documentation

### 8.3.2.5.3.1 get()

```
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     }
```

### 8.3.2.5.3.2 get\_boost\_compute()

```
boost::compute::device cl::sycl::device::get_boost_compute ( ) const [inline]
```

Return the underlying Boost.Compute device if it is an OpenCL device.

This is a triSYCL extension

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

References [cl::sycl::detail::shared\\_ptr\\_implementation< device, detail::device >::implementation](#).

Referenced by [cl::sycl::detail::openccl\\_queue::instance\(\)](#).

```
00134     {
00135     return implementation->get_boost_compute ();
00136     }
```

Here is the caller graph for this function:



**8.3.2.5.3.3** `get_info()` [1/2]

```
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 211 of file `device.hpp`.

References `get_info()`.

```
00211                                     {
00212     //return implementation->get_info<Param>(param);
00213 }
```

Here is the call graph for this function:

**8.3.2.5.3.4** `get_info()` [2/2]

```
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**

Referenced by `get_info()`.

Here is the caller graph for this function:



#### 8.3.2.5.3.5 get\_platform()

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

Return the platform of device.

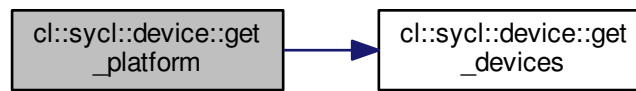
Return synchronous errors via the SYCL exception class.

Definition at line 188 of file [device.hpp](#).

References [cl::sycl::info::all](#), [get\\_devices\(\)](#), [cl::sycl::detail::shared\\_ptr\\_implementation< device, detail::device >::implementation](#), and [TRISYCL\\_WEAK\\_ATTRIB\\_SUFFIX](#).

```
00188         {
00189     return implementation->get_platform();
00190     }
```

Here is the call graph for this function:



#### 8.3.2.5.3.6 has\_extension()

```
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 233 of file [device.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< device, detail::device >::implementation](#).

```
00233                                     {
00234     return implementation->has_extension(extension);
00235     }
```

8.3.2.5.3.7 `is_accelerator()`

```
bool cl::sycl::device::is_accelerator ( ) const [inline]
```

Return true if the device is an OpenCL accelerator device.

Definition at line 159 of file [device.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< device, detail::device >::implementation](#).

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

```
00159         {
00160     return implementation->is_accelerator();
00161     }
```

Here is the caller graph for this function:

8.3.2.5.3.8 `is_cpu()`

```
bool cl::sycl::device::is_cpu ( ) const [inline]
```

Return true if the device is an OpenCL CPU device.

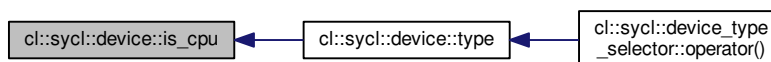
Definition at line 147 of file [device.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< device, detail::device >::implementation](#).

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

```
00147         {
00148     return implementation->is_cpu();
00149     }
```

Here is the caller graph for this function:



### 8.3.2.5.3.9 is\_gpu()

```
bool cl::sycl::device::is_gpu ( ) const [inline]
```

Return true if the device is an OpenCL GPU device.

Definition at line 153 of file [device.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< device, detail::device >::implementation](#).

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

```
00153         {
00154     return implementation->is_gpu();
00155     }
```

Here is the caller graph for this function:



### 8.3.2.5.3.10 is\_host()

```
bool cl::sycl::device::is_host ( ) const [inline]
```

Return true if the device is the SYCL host device.

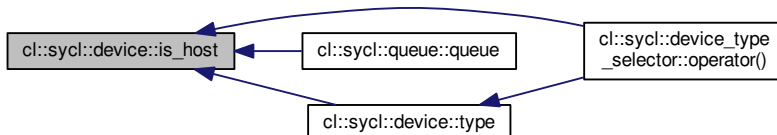
Definition at line 141 of file [device.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< device, detail::device >::implementation](#).

Referenced by [cl::sycl::device\\_type\\_selector::operator\(\)](#), [cl::sycl::queue::queue\(\)](#), and [type\(\)](#).

```
00141         {
00142     return implementation->is_host();
00143     }
```

Here is the caller graph for this function:





## 8.3.2.5.3.11 type()

```
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

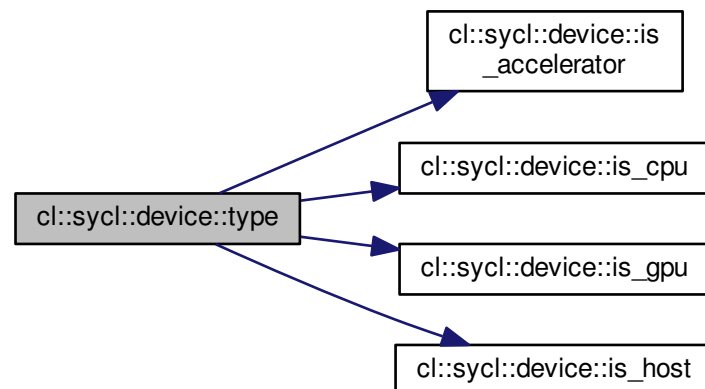
Definition at line 169 of file device.hpp.

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()`.

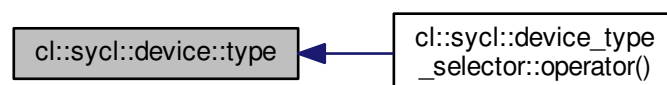
Referenced by `cl::sycl::device_type_selector::operator()`.

```
00169         {
00170             if (is_host())
00171                 return info::device_type::host;
00172             else if (is_cpu())
00173                 return info::device_type::cpu;
00174             else if (is_gpu())
00175                 return info::device_type::gpu;
00176             else if (is_accelerator())
00177                 return info::device_type::accelerator;
00178             else
00179                 // \todo Put a SYCL exception
00180                 throw std::domain_error("Unknown cl::sycl::info::device_type");
00181         }
```

Here is the call graph for this function:



Here is the caller graph for this function:



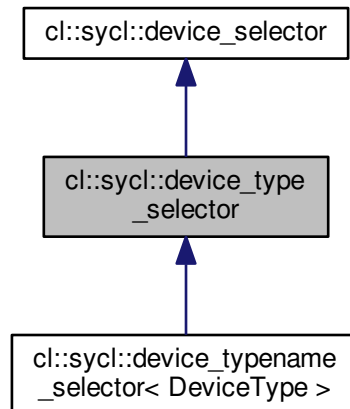
### 8.3.2.6 class `cl::sycl::device_type_selector`

A device selector by device\_type.

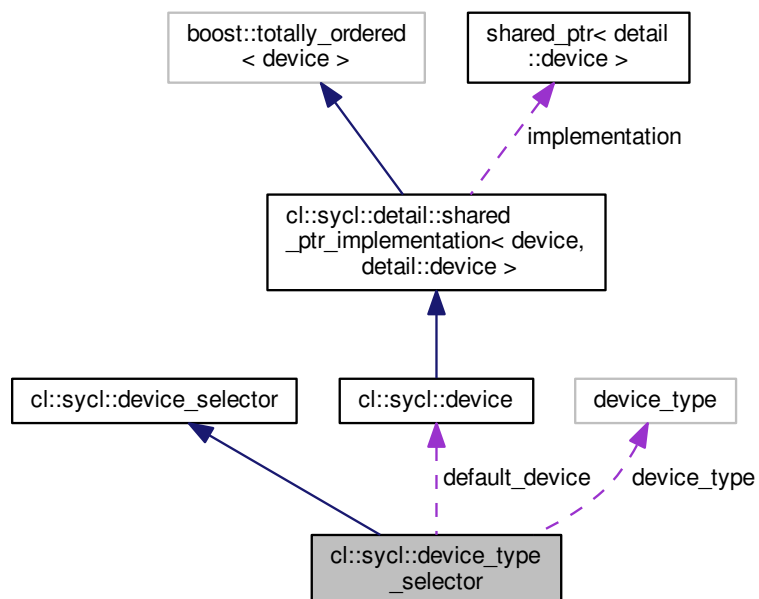
**Todo** To be added to the specification

Definition at line 32 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.*

## 8.3.2.6.1 Constructor &amp; Destructor Documentation

## 8.3.2.6.1.1 device\_type\_selector()

```
cl::sycl::device_type_selector::device_type_selector (
    info::device_type device_type ) [inline]
```

Definition at line 48 of file [device\\_selector\\_tail.hpp](#).

References [cl::sycl::info::defaults](#).

```
00049     : device_type { device_type } {
00050         // The default device selection heuristic
00051     #ifdef TRISYCL_OPENCL
00052         if (device_type == info::device_type::defaults) {
00053             // Ask Boost.Compute for the default OpenCL device
00054             try {
00055                 default_device = boost::compute::system::default_device();
00056             }
00057             catch (...) {
00058                 /* If there is no OpenCL device, just keep the
00059                    default-constructed device, which is the host device */
00060             }
00061         }
00062     #endif
00063     }
```

## 8.3.2.6.2 Member Function Documentation

### 8.3.2.6.2.1 operator()

```
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.

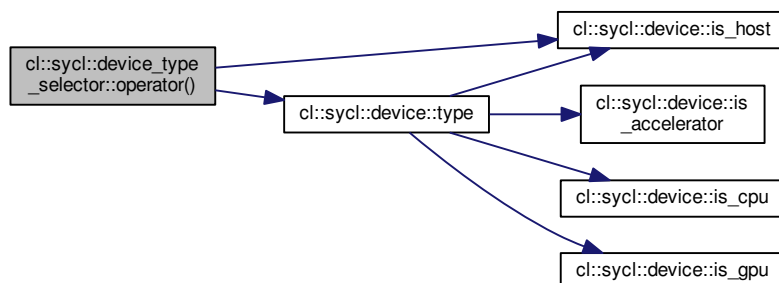
Implements `cl::sycl::device_selector`.

Definition at line 67 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()`.

```
00067 {
00068     if (device_type == info::device_type::all)
00069         // All devices fit all
00070         return 1;
00071
00072     if (device_type == info::device_type::defaults)
00073         // Only select the default device
00074         return dev == default_device ? 1 : -1;
00075
00076     if (device_type == info::device_type::opencl)
00077         // For now, any non host device is an OpenCL device
00078         return dev.is_host() ? -1 : 1;
00079
00080     return dev.type() == device_type ? 1 : -1;
00081 }
```

Here is the call graph for this function:



### 8.3.2.6.3 Member Data Documentation

#### 8.3.2.6.3.1 default\_device

```
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 44 of file `device_selector_tail.hpp`.

## 8.3.2.6.3.2 device\_type

```
info::device_type cl::sycl::device_type_selector::device_type [private]
```

The device\_type to select.

Definition at line 37 of file [device\\_selector\\_tail.hpp](#).

## 8.3.2.7 class cl::sycl::device\_typename\_selector

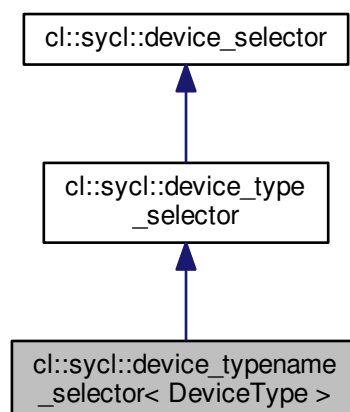
```
template<info::device_type DeviceType>  
class cl::sycl::device_typename_selector< DeviceType >
```

Select a device by template device\_type parameter.

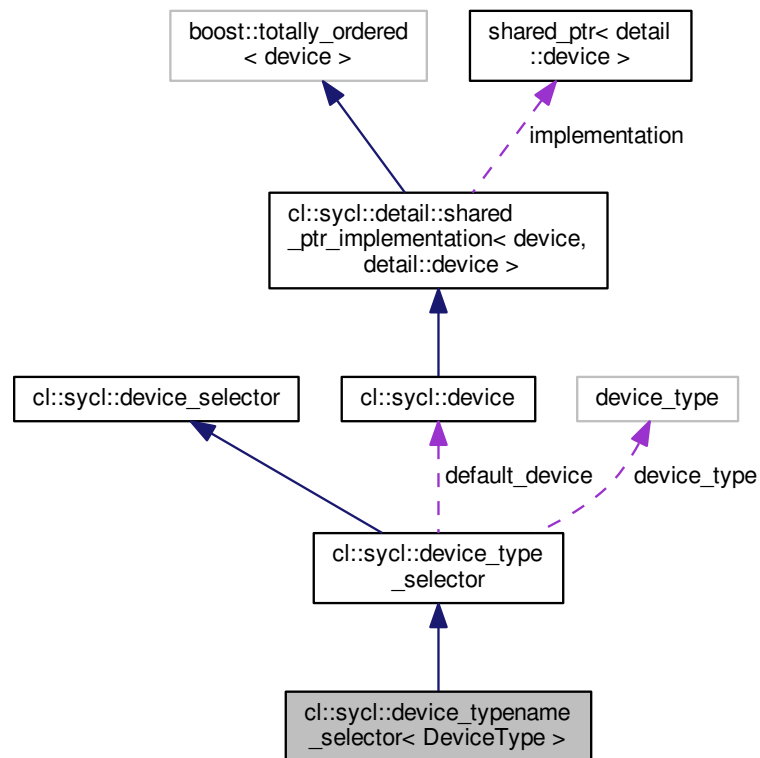
**Todo** To be added to the specification

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

Inheritance diagram for cl::sycl::device\_typename\_selector< DeviceType >:



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



## Public Member Functions

- [device\\_typename\\_selector\(\)](#)

### 8.3.2.7.1 Constructor & Destructor Documentation

#### 8.3.2.7.1.1 device\_typename\_selector()

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

Definition at line 95 of file [device\\_selector\\_tail.hpp](#).

```
00095 : device_type_selector { DeviceType } {}
```

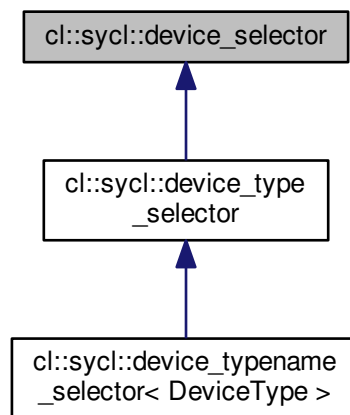
8.3.2.8 class `cl::sycl::device_selector`

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.*

## 8.3.2.8.1 Constructor &amp; Destructor Documentation

8.3.2.8.1.1 `~device_selector()`

```
virtual cl::sycl::device_selector::~~device_selector ( ) [inline], [virtual]
```

Virtual destructor so the final destructor can be called if any.

Definition at line 52 of file [device\\_selector.hpp](#).

```
00052 {}
```

### 8.3.2.8.2 Member Function Documentation

#### 8.3.2.8.2.1 operator()

```
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:



#### 8.3.2.8.2.2 select\_device()

```
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)`

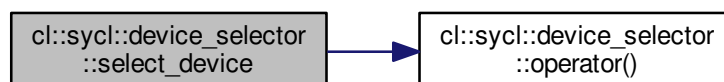
**Todo** Remove this from specification

Definition at line 35 of file [device\\_selector.hpp](#).

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

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

Here is the call graph for this function:





8.3.2.9 class `cl::sycl::handler`

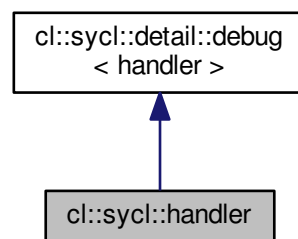
Command group handler class.

A command group handler object can only be constructed by the SYCL runtime.

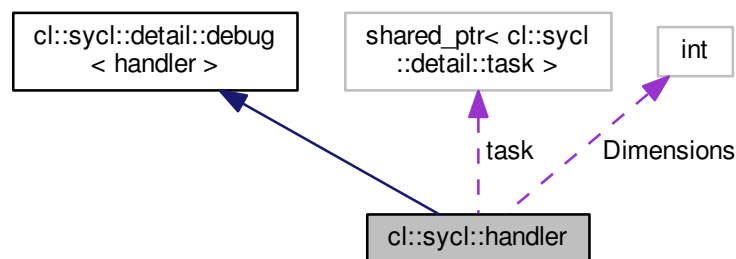
All of the accessors defined in the command group scope take as a parameter an instance of the command group handler and all the kernel invocation functions are methods of this class.

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

Inheritance diagram for `cl::sycl::handler`:



Collaboration diagram for `cl::sycl::handler`:



## Public Member Functions

- [handler](#) (const std::shared\_ptr< [detail::queue](#) > &q)
- template<typename DataType , int 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 accessor kernel arg for an OpenCL kernel which is used through the SYCL/OpenCL interop interface.*
- template<typename T , typename = std::enable\_if\_t<is\_wrapper<T>::value>>  
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 with a wrapper type.*

- `template<typename T >`  
`std::enable_if_t<!is_wrapper< T >::value > set_arg` (int arg\_index, T &&scalar\_value)

*Set kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interoperability interface without a wrapper type.*

- `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.*

- `int ParallelForFunctor void parallel_for` (nd\_range< Dimensions > r, ParallelForFunctor f)
- `template<typename KernelName = std::nullptr_t, int 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.*

- `template<typename KernelName = std::nullptr_t, int Dimensions = 1, typename ParallelForFunctor >`  
`void parallel_for_work_group` (range< Dimensions > r1, range< Dimensions > r2, 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 sycl\_kernel)

*Kernel invocation method of a kernel defined as pointer to a kernel object, described in detail in 3.5.3.*

## Public Attributes

- `std::shared_ptr< detail::task > task`  
*Attach the task and accessors to it.*
- `int 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.*

### 8.3.2.9.1 Constructor & Destructor Documentation

#### 8.3.2.9.1.1 handler()

```
cl::sycl::handler::handler (
    const std::shared_ptr< detail::queue > & q ) [inline]
```

Definition at line 62 of file `handler.hpp`.

References `Dimensions`, and `cl::sycl::access::global_buffer`.

```
00062                                     {
00063     // Create a new task for this command_group
00064     task = std::make_shared<detail::task>(q);
00065 }
```

## 8.3.2.9.2 Member Function Documentation

## 8.3.2.9.2.1 dispatch\_set\_arg()

```
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 133 of file [handler.hpp](#).

References [set\\_arg\(\)](#).

Referenced by [set\\_args\(\)](#).

```
00133
00134 // Use an intermediate tuple to ease individual argument access {
00135 auto &&t = std::make_tuple(std::forward<Ts>(args)...);
00136 // Dispatch individual set_arg() for each argument
00137 auto just_to_evaluate = {
00138     0 /**< At least 1 element to deal with empty set_args() *//,
00139     ( set_arg(Is, std::forward<Ts>(std::get<Is>(t))), 0)...
00140 };
00141 // Remove the warning about unused variable
00142 static_cast<void>(just_to_evaluate);
00143 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.3.2.9.2.2 parallel\_for()

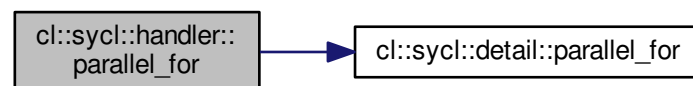
```
int ParallelForFunctor void cl::sycl::handler::parallel_for (
    nd_range< Dimensions > r,
    ParallelForFunctor f ) [inline]
```

Definition at line 285 of file [handler.hpp](#).

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

```
00285                                     {
00286     task->schedule(detail::trace_kernel<KernelName>([=] {
00287         detail::parallel_for(r, f);
00288     }));
00289 }
```

Here is the call graph for this function:



### 8.3.2.9.2.3 parallel\_for\_work\_group() [1/2]

```
template<typename KernelName = std::nullptr_t, int 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 |

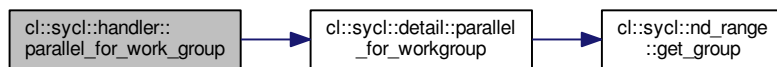
Definition at line 316 of file [handler.hpp](#).

References [cl::sycl::detail::parallel\\_for\\_workgroup\(\)](#).

Referenced by [parallel\\_for\\_work\\_group\(\)](#).

```
00317
00318     task->schedule(detail::trace_kernel<KernelName>{ [=] {
00319         detail::parallel_for_workgroup(r, f); }});
00320 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.3.2.9.2.4 parallel\_for\_work\_group() [2/2]

```
template<typename KernelName = std::nullptr_t, int Dimensions = 1, typename ParallelForFunctor
>
void cl::sycl::handler::parallel_for_work_group (
    range< Dimensions > r1,
    range< Dimensions > r2,
    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 347 of file [handler.hpp](#).

References [parallel\\_for\\_work\\_group\(\)](#).

```

00348
00349     parallel_for_work_group(nd_range<Dimensions> { r1, r2 }, f);
00350 }
```

Here is the call graph for this function:



### 8.3.2.9.2.5 set\_arg() [1/3]

```

template<typename DataType , int 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 accessor 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 87 of file [handler.hpp](#).

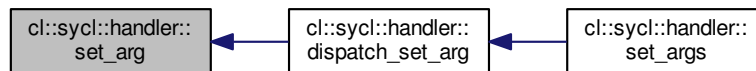
Referenced by [dispatch\\_set\\_arg\(\)](#).

```

00088                                     {
00089     /* Think about setting the kernel argument before actually calling
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         task->get_kernel().get_boost_compute()
00096         .set_arg(arg_index, acc_obj.implementation->get_cl_buffer());
00097     });
00098 }

```

Here is the caller graph for this function:



#### 8.3.2.9.2.6 set\_arg() [2/3]

```

template<typename T , typename = std::enable_if_t<is_wrapper<T>::value>>
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 with a wrapper type.

Definition at line 105 of file [handler.hpp](#).

```

00105                                     {
00106     /* Explicitly capture task by copy instead of having this captured
00107        by reference and task by reference by side effect */
00108     task->add_prelude([=, task = task] {
00109         task->get_kernel().get_boost_compute()
00110         .set_arg(arg_index, scalar_value.unwrap());
00111     });
00112 }

```

#### 8.3.2.9.2.7 set\_arg() [3/3]

```

template<typename T >
std::enable_if_t<!is_wrapper<T>::value> 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 without a wrapper type.

Definition at line 120 of file [handler.hpp](#).

```

00120                                     {
00121     /* Explicitly capture task by copy instead of having this captured
00122        by reference and task by reference by side effect */
00123     task->add_prelude([=, task = task] {
00124         task->get_kernel().get_boost_compute()
00125         .set_arg(arg_index, scalar_value);
00126     });
00127 }

```

### 8.3.2.9.2.8 set\_args()

```
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.

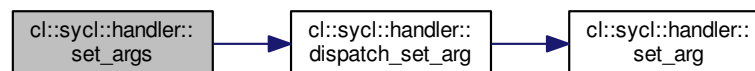
**Todo** Update the specification to add this function according to [https://cvs.khronos.org/bugzilla/show\\_bug.cgi?id=15978](https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15978) proposal

Definition at line 154 of file `handler.hpp`.

References `dispatch_set_arg()`.

```
00154         {
00155         /* Construct a set of increasing argument index to be able to call
00156            the real set_arg */
00157         dispatch_set_arg(std::index_sequence_for<Ts...>{},
00158                         std::forward<Ts>(args)...);
00159     }
```

Here is the call graph for this function:



### 8.3.2.9.2.9 single\_task() [1/2]

```
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 <code>single_task</code>            |
| <i>KernelName</i> | is a class type that defines the name to be used for the underlying kernel |



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

```
00177         {
00178     task->schedule(detail::trace_kernel<KernelName>(F));
00179 }
```

#### 8.3.2.9.2.10 single\_task() [2/2]

```
void cl::sycl::handler::single_task (
    kernel sycl_kernel ) [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

Definition at line 359 of file [handler.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< Parent, Implementation >::implementation](#).

```
00359         {
00360     /* For now just use the usual host task system to schedule
00361     manually the OpenCL kernels instead of using OpenCL event-based
00362     scheduling
00363
00364     \todo Move the tracing inside the kernel implementation
00365
00366     \todo Simplify this 2 step ugly interface
00367     */
00368     task->set_kernel(sycl_kernel.implementation);
00369     task->schedule(detail::trace_kernel<kernel>([=, t = task] {
00370         sycl_kernel.implementation->single_task(t, t->get_queue());
00371     }));
00372 }
```

#### 8.3.2.9.2.11 TRISYCL\_parallel\_for\_functor\_GLOBAL()

```
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 dimensions. Kernel invocation method of a kernel defined as a lambda or functor, for the specified [nd\\_range](#) and given an [nd\\_item](#) for indexing in the indexing space defined by the [nd\\_range](#).

If it is a lambda function or the if the functor type is globally visible there is no need for the developer to provide a kernel name type (typename KernelName) for it, as described in detail in 3.5.3.

#### 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 |

### 8.3.2.9.3 Member Data Documentation

#### 8.3.2.9.3.1 Dimensions

```
int cl::sycl::handler::Dimensions
```

Definition at line 283 of file [handler.hpp](#).

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

#### 8.3.2.9.3.2 task

```
std::shared_ptr<detail::task> cl::sycl::handler::task
```

Attach the task and accessors to it.

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

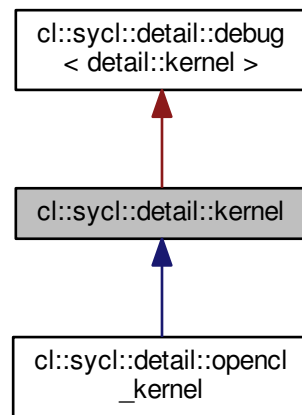
Referenced by [cl::sycl::detail::add\\_buffer\\_to\\_task\(\)](#).

### 8.3.2.10 class cl::sycl::detail::kernel

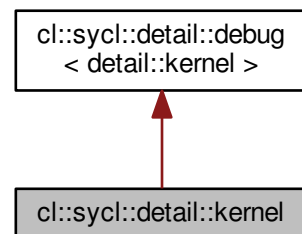
Abstract SYCL kernel.

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.*
- virtual void [single\\_task](#) (std::shared\_ptr< [detail::task](#) > [task](#), std::shared\_ptr< [detail::queue](#) > q)=0  
*Launch a single task of the kernel.*
- [TRISYCL\\_ParallelForKernel\\_RANGE](#) (1) [TRISYCL\\_ParallelForKernel\\_RANGE](#)(2) [TRISYCL\\_ParallelForKernel\\_RANGE](#)(3) virtual `~kernel`()  
*Return the context that this kernel is defined for.*

### 8.3.2.10.1 Member Function Documentation

#### 8.3.2.10.1.1 `get()`

```
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](#).

#### 8.3.2.10.1.2 `get_boost_compute()`

```
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](#).

#### 8.3.2.10.1.3 `single_task()`

```
virtual void cl::sycl::detail::kernel::single_task (
    std::shared_ptr< detail::task > task,
    std::shared_ptr< detail::queue > q ) [pure virtual]
```

Launch a single task of the kernel.

Implemented in [cl::sycl::detail::opencl\\_kernel](#).

#### 8.3.2.10.1.4 `TRISYCL_ParallelForKernel_RANGE()`

```
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 67 of file [kernel.hpp](#).

```
00080          {}
```

8.3.2.11 class `cl::sycl::kernel`

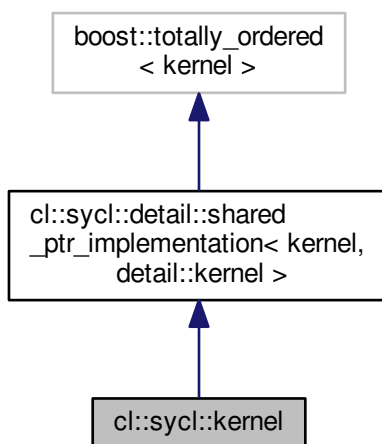
SYCL kernel.

**Todo** To be implemented

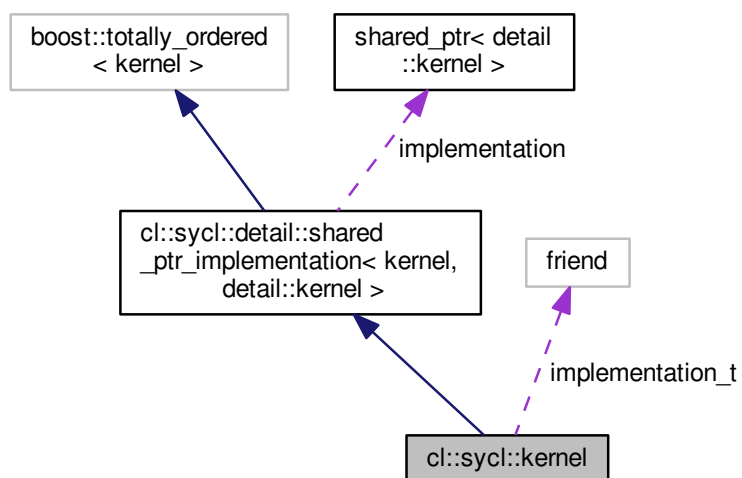
**Todo** Check specification

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` = typename `kernel::shared_ptr_implementation`

## Private Attributes

- friend `implementation_t`

## Friends

- class `handler`

## Additional Inherited Members

### 8.3.2.11.1 Member Typedef Documentation

#### 8.3.2.11.1.1 implementation\_t

```
using cl::sycl::kernel::implementation_t = typename kernel::shared_ptr_implementation [private]
```

Definition at line 44 of file `kernel.hpp`.

### 8.3.2.11.2 Constructor & Destructor Documentation

#### 8.3.2.11.2.1 kernel() [1/3]

```
cl::sycl::kernel::kernel ( ) [delete]
```

The default object is not valid because there is no program or.

`cl_kernel`

associated with it

**8.3.2.11.2.2 kernel()** [2/3]

```
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 69 of file [kernel.hpp](#).

```
00069 : kernel { boost::compute::kernel { k } } {}
```

**8.3.2.11.2.3 kernel()** [3/3]

```
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.

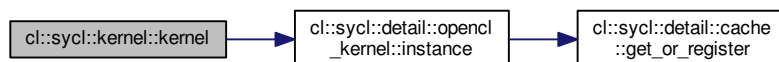
Return synchronous errors via the SYCL exception class.

Definition at line 78 of file [kernel.hpp](#).

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

```
00079 : implementation_t { detail::openccl_kernel::instance(k)
    } {}
```

Here is the call graph for this function:

**8.3.2.11.3 Member Function Documentation**

#### 8.3.2.11.3.1 get()

```
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 87 of file [kernel.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< kernel, detail::kernel >::implementation](#), and [cl::sycl::detail::unimplemented\(\)](#).

```
00087         {
00088     return implementation->get();
00089     }
```

Here is the call graph for this function:



#### 8.3.2.11.4 Friends And Related Function Documentation

##### 8.3.2.11.4.1 handler

```
friend class handler [friend]
```

Definition at line 47 of file [kernel.hpp](#).

#### 8.3.2.11.5 Member Data Documentation

##### 8.3.2.11.5.1 implementation\_t

```
friend cl::sycl::kernel::implementation_t [private]
```

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

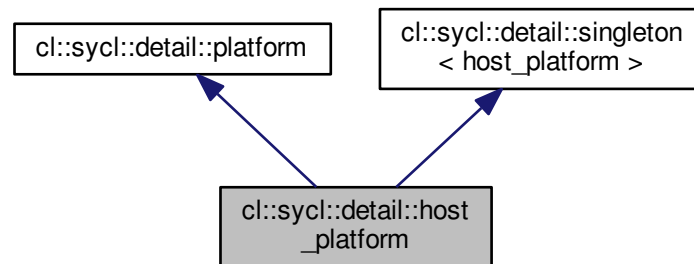


8.3.2.12 class `cl::sycl::detail::host_platform`

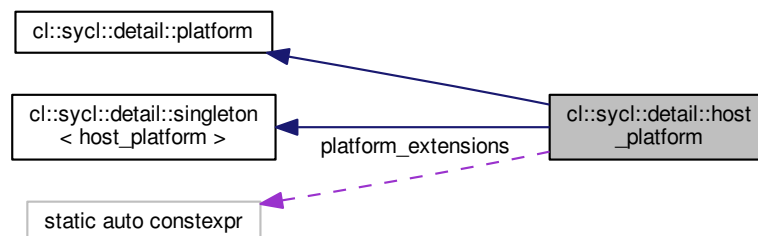
SYCL host platform.

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.*
- `boost::compute::platform &` [get\\_boost\\_compute](#) () const override  
*Return the underlying Boost.Compute 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.*
- `vector_class< cl::sycl::device >` [get\\_devices](#) (`info::device_type` device\_type) const override  
*Get all the available devices for the host platform.*

## Static Private Attributes

- static auto constexpr [platform\\_extensions](#) = "Xilinx\_blocking\_pipes"

## Additional Inherited Members

### 8.3.2.12.1 Member Function Documentation

#### 8.3.2.12.1.1 get()

```
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 }
```

#### 8.3.2.12.1.2 get\_boost\_compute()

```
boost::compute::platform& cl::sycl::detail::host_platform::get_boost_compute ( ) const [inline],
[override], [virtual]
```

Return the underlying Boost.Compute platform.

This throws an error since there is no Boost Compute platform associated to the host platform.

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

Definition at line 55 of file [host\\_platform.hpp](#).

```
00055                                     {
00056     throw
00057         non_cl_error("The host device has no underlying Boost Compute platform");
00058 }
```

8.3.2.12.1.3 `get_info_string()`

```
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 71 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`.

```
00071                                     {
00072     switch (param) {
00073     case info::platform::profile:
00074         /* Well... Is the host platform really a full profile whereas it
00075          is not really OpenCL? */
00076         return "FULL_PROFILE";
00077
00078     case info::platform::version:
00079         // \todo I guess it should include the software version too...
00080         return "2.2";
00081
00082     case info::platform::name:
00083         return "triSYCL host platform";
00084
00085     case info::platform::vendor:
00086         return "triSYCL Open Source project";
00087
00088     case info::platform::extensions:
00089         return platform_extensions;
00090
00091     default:
00092         // \todo Define some SYCL exception type for this type of errors
00093         throw std::invalid_argument {
00094             "Unknown parameter value for SYCL platform information" };
00095     }
00096 }
```

8.3.2.12.1.4 `has_extension()`

```
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

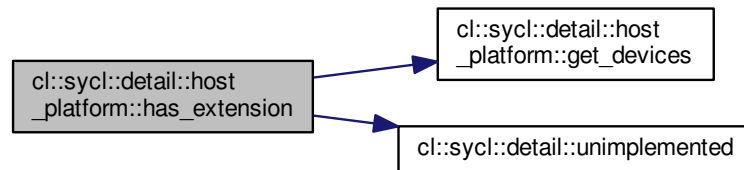
Implements `cl::sycl::detail::platform`.

Definition at line 103 of file `host_platform.hpp`.

References `get_devices()`, and `cl::sycl::detail::unimplemented()`.

```
00103                                     {
00104     detail::unimplemented();
00105     return {};
00106 }
```

Here is the call graph for this function:



#### 8.3.2.12.1.5 is\_host()

```
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 63 of file [host\\_platform.hpp](#).

```
00063         {
00064     return true;
00065 }
```

#### 8.3.2.12.2 Member Data Documentation

##### 8.3.2.12.2.1 platform\_extensions

```
auto constexpr cl::sycl::detail::host_platform::platform_extensions = "Xilinx_blocking_pipes"
[static], [private]
```

Definition at line 35 of file [host\\_platform.hpp](#).

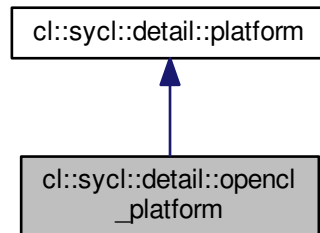
Referenced by [get\\_info\\_string\(\)](#).

8.3.2.13 class `cl::sycl::detail::openccl_platform`

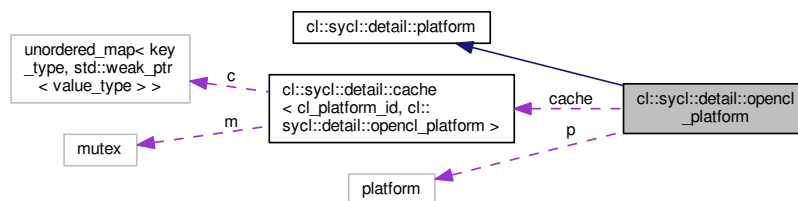
SYCL OpenCL platform.

Definition at line 36 of file [openccl\\_platform.hpp](#).

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



Collaboration diagram for `cl::sycl::detail::openccl_platform`:



## Public Member Functions

- `cl_platform_id` [get](#) () const override  
*Return the `cl_platform_id` of the underlying OpenCL platform.*
- const `boost::compute::platform` & [get\\_boost\\_compute](#) () const override  
*Return the underlying Boost.Compute 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.*
- `vector_class< cl::sycl::device >` [get\\_devices](#) (`info::device_type` device\_type) const override  
*Get all the available devices for this OpenCL platform.*
- `~openccl_platform` () override  
*Unregister from the cache on destruction.*

### Static Public Member Functions

- static `std::shared_ptr< opencil\_platform > instance` (const boost::compute::platform &p)

### Private Member Functions

- [opencil\\_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::opencil\\_platform](#) > [cache](#)  
*A cache to always return the same live platform for a given OpenCL platform.*

## 8.3.2.13.1 Constructor & Destructor Documentation

### 8.3.2.13.1.1 opencil\_platform()

```
cl::sycl::detail::opencil_platform::opencil_platform (
    const boost::compute::platform & p ) [inline], [private]
```

Only the instance factory can built it.

Definition at line 105 of file [opencil\\_platform.hpp](#).

```
00105 : p { p } {}
```

### 8.3.2.13.1.2 ~opencil\_platform()

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

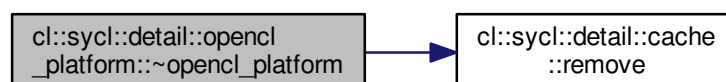
Unregister from the cache on destruction.

Definition at line 110 of file [opencil\\_platform.hpp](#).

References [cache](#), [cl::sycl::detail::cache< Key, Value >::remove\(\)](#), [TRISYCL\\_WEAK\\_ATTRIB\\_PREFIX](#), and [cl::sycl::detail::TRISYCL\\_WEAK\\_ATTRIB\\_SUFFIX](#).

```
00110 {
00111     cache.remove(p.id());
00112 }
```

Here is the call graph for this function:



## 8.3.2.13.2 Member Function Documentation

8.3.2.13.2.1 `get()`

```
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 }
```

8.3.2.13.2.2 `get_boost_compute()`

```
const boost::compute::platform& cl::sycl::detail::opencl_platform::get_boost_compute ( ) const
[inline], [override], [virtual]
```

Return the underlying Boost.Compute platform.

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

Definition at line 57 of file [opencl\\_platform.hpp](#).

References [p](#).

Referenced by [get\\_devices\(\)](#).

```
00057                                     {
00058     return p;
00059 }
```

Here is the caller graph for this function:



### 8.3.2.13.2.3 `get_info_string()`

```
string_class cl::sycl::detail::opencl_platform::get_info_string (
    info::platform param ) const [inline], [override], [virtual]
```

Returning the information string parameters for the OpenCL platform.

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

Definition at line 69 of file [opencl\\_platform.hpp](#).

```
00069                                     {
00070     /* Use the fact that the triSYCL info values are the same as the
00071        OpenCL ones used in Boost.Compute to just cast the enum class
00072        to the int value */
00073     return p.get_info<std::string>(static_cast<cl_platform_info>(param));
00074 }
```

### 8.3.2.13.2.4 `has_extension()`

```
bool cl::sycl::detail::opencl_platform::has_extension (
    const string_class & extension ) const [inline], [override], [virtual]
```

Specify whether a specific extension is supported on the platform.

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

Definition at line 78 of file [opencl\\_platform.hpp](#).

```
00078                                     {
00079     return p.supports_extension(extension);
00080 }
```

### 8.3.2.13.2.5 `instance()`

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

Definition at line 85 of file [opencl\\_platform.hpp](#).

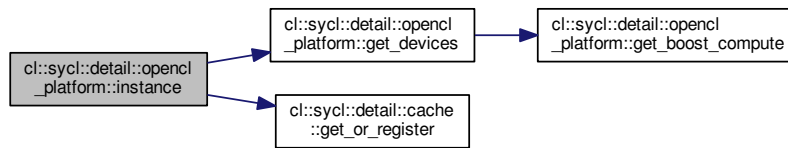
References [get\\_devices\(\)](#), and [cl::sycl::detail::cache< Key, Value >::get\\_or\\_register\(\)](#).

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

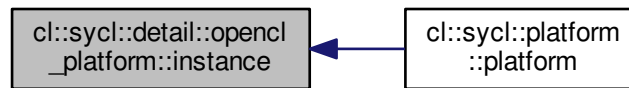
```
00085                                     {
00086     return cache.get_or_register(p.id(),
00087        [&] { return new opencl_platform {
00088        p }; });
00088 }
```



Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.3.2.13.2.6 is\_host()

```
bool cl::sycl::detail::opengl_platform::is_host ( ) const [inline], [override], [virtual]
```

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

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

Definition at line 63 of file [opengl\\_platform.hpp](#).

```
00063                                     {
00064     return false;
00065 }
```

#### 8.3.2.13.3 Member Data Documentation

##### 8.3.2.13.3.1 cache

```
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 OpenCL platform.

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

Definition at line 46 of file [opengl\\_platform.hpp](#).

Referenced by [~opengl\\_platform\(\)](#).

## 8.3.2.13.3.2 p

```
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](#).

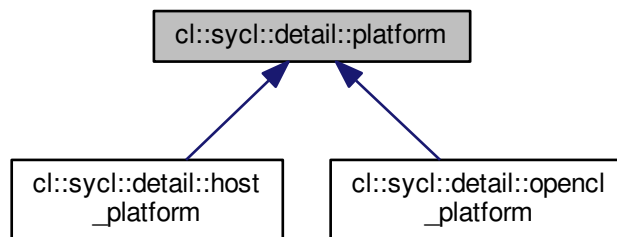
Referenced by [get\\_boost\\_compute\(\)](#).

## 8.3.2.14 class cl::sycl::detail::platform

An abstract class representing various models of SYCL platforms.

Definition at line 28 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 const `boost::compute::platform` & [get\\_boost\\_compute](#) () const =0  
*Return the underlying Boost.Compute platform, if any.*
- 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 `vector_class`< `device` > [get\\_devices](#) (`info::device_type` device\_type) const =0  
*Get all the available devices for this platform.*
- virtual `~platform` ()

## 8.3.2.14.1 Constructor &amp; Destructor Documentation

**8.3.2.14.1.1** `~platform()`

```
virtual cl::sycl::detail::platform::~~platform ( ) [inline], [virtual]
```

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

```
00065 {}
```

**8.3.2.14.2** Member Function Documentation**8.3.2.14.2.1** `get()`

```
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](#).

**8.3.2.14.2.2** `get_boost_compute()`

```
virtual const boost::compute::platform& cl::sycl::detail::platform::get_boost_compute ( )
const [pure virtual]
```

Return the underlying Boost.Compute platform, if any.

Implemented in [cl::sycl::detail::opencl\\_platform](#), and [cl::sycl::detail::host\\_platform](#).

**8.3.2.14.2.3** `get_devices()`

```
virtual vector_class<device> cl::sycl::detail::platform::get_devices (
    info::device_type device_type ) const [pure virtual]
```

Get all the available devices for this platform.

**Parameters**

|    |                    |   |
|----|--------------------|---|
| in | <i>device_type</i> | is the device type to filter the selection or <a href="#">info::device_type::all</a> by default to return all the devices |
|----|--------------------|---|

**Returns**

the device list

Implemented in [cl::sycl::detail::host\\_platform](#), and [cl::sycl::detail::opencl\\_platform](#).

#### 8.3.2.14.2.4 get\_info\_string()

```
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::host\\_platform](#), and [cl::sycl::detail::opencl\\_platform](#).

#### 8.3.2.14.2.5 has\_extension()

```
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](#).

#### 8.3.2.14.2.6 is\_host()

```
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::host\\_platform](#), and [cl::sycl::detail::opencl\\_platform](#).

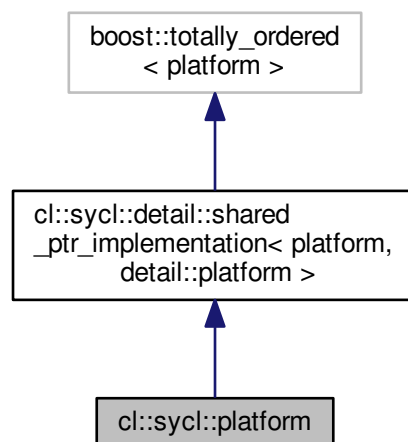
### 8.3.2.15 class cl::sycl::platform

Abstract the OpenCL platform.

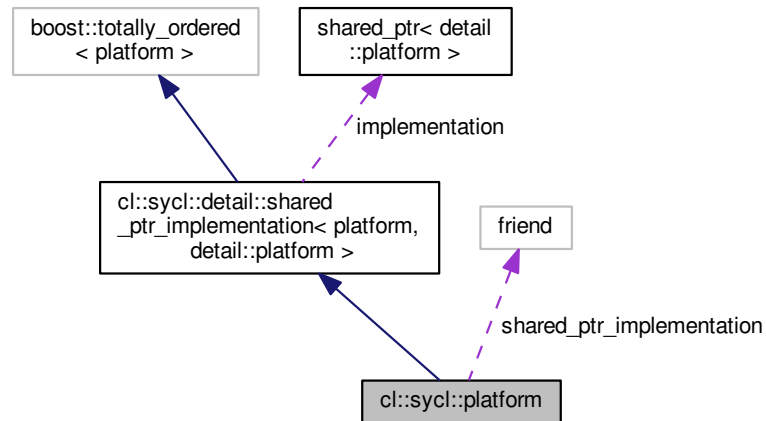
**Todo** triSYCL Implementation

Definition at line 42 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.*
- `const boost::compute::platform get_boost_compute () const`  
*Return the underlying Boost.Compute platform if it is an 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.*
- `vector_class< device > get_devices (info::device_type device_type=info::device_type::all) const`  
*Get all the available devices for this platform.*

#### Static Public Member Functions

- `static vector_class< platform > get_platforms ()`  
*Get the list of all the platforms available to the application.*

## Private Attributes

- friend [shared\\_ptr\\_implementation](#)

## Additional Inherited Members

### 8.3.2.15.1 Constructor & Destructor Documentation

#### 8.3.2.15.1.1 platform() [1/4]

```
cl::sycl::platform::platform ( ) [inline]
```

Default constructor for platform which is the host platform.

Returns errors via the SYCL exception class.

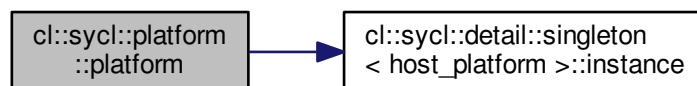
Definition at line 59 of file [platform.hpp](#).

References [cl::sycl::detail::singleton< host\\_platform >::instance\(\)](#).

Referenced by [cl::sycl::detail::opengl\\_device::get\\_platform\(\)](#).

```
00059      :
00060      shared_ptr_implementation {
        detail::host_platform::instance() } {}
```

Here is the call graph for this function:



Here is the caller graph for this function:



## 8.3.2.15.1.2 platform() [2/4]

```
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 71 of file [platform.hpp](#).

```
00072      : platform { boost::compute::platform { platform_id } } {}
```

## 8.3.2.15.1.3 platform() [3/4]

```
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.

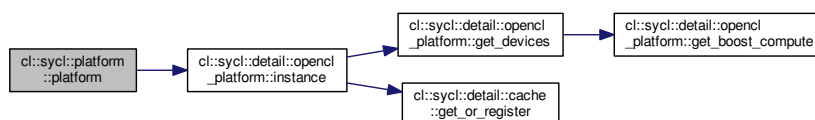
Return synchronous errors via the SYCL exception class.

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

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

```
00082      : shared_ptr_implementation {
    detail::opencl_platform::instance(p) } {}
```

Here is the call graph for this function:



#### 8.3.2.15.1.4 platform() [4/4]

```
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.

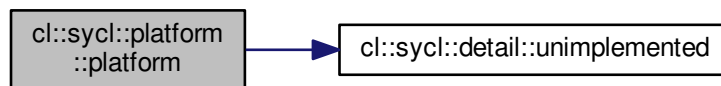
Returns errors via the SYCL exception class.

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

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

```
00091                                     {
00092     detail::unimplemented();
00093 }
```

Here is the call graph for this function:



### 8.3.2.15.2 Member Function Documentation

#### 8.3.2.15.2.1 get()

```
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 104 of file [platform.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< platform, detail::platform >::implementation](#).

```
00104     {
00105     return implementation->get();
00106 }
```



**8.3.2.15.2.2** `get_boost_compute()`

```
const boost::compute::platform cl::sycl::platform::get_boost_compute ( ) const [inline]
```

Return the underlying Boost.Compute platform if it is an OpenCL platform.

This is a triSYCL extension

Definition at line 114 of file [platform.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< platform, detail::platform >::implementation](#).

```
00114                                     {
00115     return implementation->get_boost_compute();
00116 }
```

**8.3.2.15.2.3** `get_devices()`

```
vector_class<device> cl::sycl::platform::get_devices (
    info::device_type device_type = info::device_type::all ) const [inline]
```

Get all the available devices for this platform.

**Parameters**

|    |                    |   |
|----|--------------------|---|
| in | <i>device_type</i> | is the device type to filter the selection or <a href="#">info::device_type::all</a> by default to return all the devices |
|----|--------------------|---|

**Returns**

the device list

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

References [cl::sycl::detail::shared\\_ptr\\_implementation< platform, detail::platform >::implementation](#).

```
00180                                     {
00181     return implementation->get_devices(device_type);
00182 }
```

**8.3.2.15.2.4** `get_info()` [1/2]

```
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 140 of file [platform.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< platform, detail::platform >::implementation](#).

```
00140                                     {
00141     // Only strings are needed here
00142     return implementation->get_info_string(param);
00143 }
```

#### 8.3.2.15.2.5 get\_info() [2/2]

```
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 149 of file [platform.hpp](#).

```
00149                                     {
00150     /* Forward to the implementation without using template parameter
00151      but with a parameter instead, since it is incompatible with
00152      virtual function and because fortunately only strings are
00153      needed here */
00154     return get_info<typename info::param_traits<info::platform,
00155                                         Param>::type>(Param);
00156 }
```

#### 8.3.2.15.2.6 get\_platforms()

```
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 121 of file [platform.hpp](#).

```
00121                                     {
00122     // Start with the default platform
00123     vector_class<platform> platforms { {} };
00124
00125     #ifdef TRISYCL_OPENCL
00126     // Then add all the OpenCL platforms
00127     for (const auto &d : boost::compute::system::platforms())
00128         platforms.emplace_back(d);
00129     #endif
00130
00131     return platforms;
00132 }
```

#### 8.3.2.15.2.7 has\_extension()

```
bool cl::sycl::platform::has_extension (
    const string_class & extension ) const [inline]
```

Test if an extension is available on the platform.

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

References [cl::sycl::detail::shared\\_ptr\\_implementation< platform, detail::platform >::implementation](#).

```
00160                                     {
00161     return implementation->has_extension(extension);
00162 }
```

#### 8.3.2.15.2.8 is\_host()

```
bool cl::sycl::platform::is_host ( ) const [inline]
```

Test if this platform is a host platform.

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

References [cl::sycl::detail::shared\\_ptr\\_implementation< platform, detail::platform >::implementation](#).

```
00166                                     {
00167     return implementation->is_host();
00168 }
```

### 8.3.2.15.3 Member Data Documentation

#### 8.3.2.15.3.1 shared\_ptr\_implementation

```
friend cl::sycl::platform::shared_ptr_implementation [private]
```

Definition at line 48 of file [platform.hpp](#).

### 8.3.2.16 class `cl::sycl::queue`

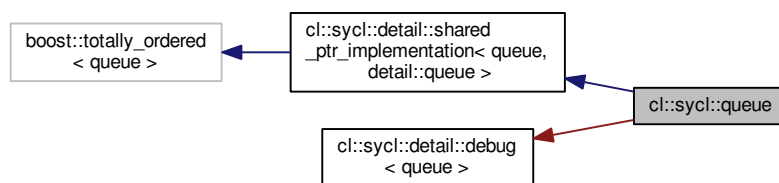
SYCL queue, similar to the OpenCL queue concept.

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

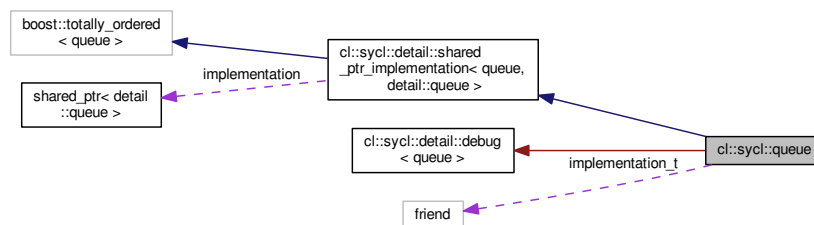
**Todo** All the queue methods should return a `queue&` instead of `void` to it is possible to chain operations

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](#) &d, [async\\_handler](#) asyncHandler=nullptr)  
A queue is created for a SYCL device.
- [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.*
- boost::compute::command\_queue `get_boost_compute` () const  
*Return the underlying Boost.Compute command queue if it is an OpenCL queue.*
- `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` = typename `queue::shared_ptr_implementation`

#### Private Attributes

- friend `implementation_t`

#### Additional Inherited Members

##### 8.3.2.16.1 Member Typedef Documentation

### 8.3.2.16.1.1 implementation\_t

using `cl::sycl::queue::implementation_t` = typename `queue::shared_ptr_implementation` [private]

Definition at line 87 of file `queue.hpp`.

### 8.3.2.16.2 Constructor & Destructor Documentation

#### 8.3.2.16.2.1 queue() [1/9]

`cl::sycl::queue::queue ( )` [inline]

Default constructor for platform which is the host platform.

Returns errors via the SYCL exception class.

**Todo** Check with the specification if it is the host queue or the one related to the default device selector.

Definition at line 107 of file `queue.hpp`.

```
00107 : implementation_t { new detail::host_queue } {}
```

#### 8.3.2.16.2.2 queue() [2/9]

`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 126 of file `queue.hpp`.

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

```
00126                                     : queue { } {
00127     detail::unimplemented();
00128 }
```

Here is the call graph for this function:



**8.3.2.16.2.3 queue()** [3/9]

```
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 139 of file [queue.hpp](#).

```
00142      : queue { device { deviceSelector }, asyncHandler } { }
```

**8.3.2.16.2.4 queue()** [4/9]

```
cl::sycl::queue::queue (
    const device & d,
    async\_handler asyncHandler = nullptr ) [inline]
```

A queue is created for a SYCL device.

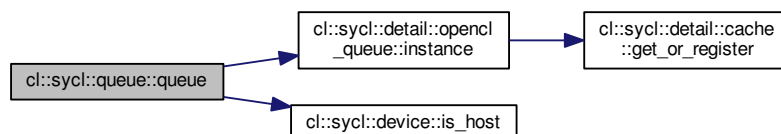
Return asynchronous errors via the `async_handler` callback function.

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

References [cl::sycl::detail::opencl\\_queue::instance\(\)](#), and [cl::sycl::device::is\\_host\(\)](#).

```
00151                                     : implementation\_t {
00152     #ifdef TRISYCL_OPENCL
00153         d.is_host()
00154         ? std::shared_ptr<detail::queue>{ new detail::host_queue }
00155         : detail::opencl\_queue::instance(d)
00156     #else
00157         new detail::host_queue
00158     #endif
00159
00160     } { }
```

Here is the call graph for this function:



#### 8.3.2.16.2.5 queue() [5/9]

```
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 174 of file [queue.hpp](#).

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

```
00176                                     : queue { } {
00177     detail::unimplemented();
00178 }
```

Here is the call graph for this function:



#### 8.3.2.16.2.6 queue() [6/9]

```
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 190 of file [queue.hpp](#).

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



```

00192                                     : queue { } {
00193     detail::unimplemented();
00194 }

```

Here is the call graph for this function:



#### 8.3.2.16.2.7 queue() [7/9]

```

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 208 of file `queue.hpp`.

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

```

00211                                     : queue { } {
00212     detail::unimplemented();
00213 }

```

Here is the call graph for this function:



**8.3.2.16.2.8 queue()** [8/9]

```
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 226 of file [queue.hpp](#).

```
00227      : queue { boost::compute::command_queue { q }, ah } {}
```

**8.3.2.16.2.9 queue()** [9/9]

```
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.

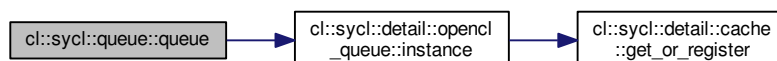
**Todo** Deal with handler

Definition at line 238 of file [queue.hpp](#).

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

```
00239      : implementation_t { detail::openccl_queue::instance(q) }
      {}
```

Here is the call graph for this function:



#### 8.3.2.16.3 Member Function Documentation

##### 8.3.2.16.3.1 get()

```
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 254 of file [queue.hpp](#).

```
00254         {
00255     return implementation->get();
00256     }
```

##### 8.3.2.16.3.2 get\_boost\_compute()

```
boost::compute::command_queue cl::sycl::queue::get_boost_compute ( ) const [inline]
```

Return the underlying Boost.Compute command queue if it is an OpenCL queue.

This is a triSYCL extension

Definition at line 264 of file [queue.hpp](#).

```
00264         {
00265     return implementation->get_boost_compute();
00266     }
```

##### 8.3.2.16.3.3 get\_context()

```
context cl::sycl::queue::get_context ( ) const [inline]
```

Return the SYCL queue's context.

Report errors using SYCL exception classes.

Definition at line 274 of file [queue.hpp](#).

```
00274         {
00275     return implementation->get_context();
00276     }
```

#### 8.3.2.16.3.4 get\_device()

```
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 283 of file [queue.hpp](#).

```
00283         {
00284     return implementation->get_device();
00285     }
```

#### 8.3.2.16.3.5 get\_info()

```
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 335 of file [queue.hpp](#).

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

```
00335                                     {
00336     detail::unimplemented();
00337     return {};
00338     }
```

Here is the call graph for this function:



#### 8.3.2.16.3.6 is\_host()

```
bool cl::sycl::queue::is_host ( ) const [inline]
```

Return whether the queue is executing on a SYCL host device.

Definition at line 289 of file [queue.hpp](#).

```
00289         {
00290     return implementation->is_host();
00291     }
```

**8.3.2.16.3.7 submit()** [1/2]

```
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 353 of file `queue.hpp`.

```
00353                                     {
00354     handler command_group_handler { implementation };
00355     cgf(command_group_handler);
00356     return {};
00357 }
```

**8.3.2.16.3.8 submit()** [2/2]

```
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 369 of file `queue.hpp`.

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

```
00369                                     {
00370     detail::unimplemented();
00371     // Since it is not implemented, always submit on the main queue
00372     return submit(cgf);
00373 }
```

Here is the call graph for this function:



#### 8.3.2.16.3.9 throw\_asynchronous()

```
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.

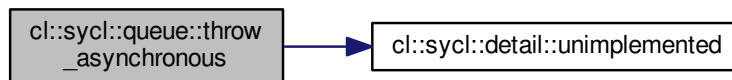
If no `async_handler` was provided then asynchronous exceptions will be lost.

Definition at line 328 of file [queue.hpp](#).

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

```
00328         {  
00329     detail::unimplemented();  
00330     }
```

Here is the call graph for this function:



#### 8.3.2.16.3.10 wait()

```
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 299 of file [queue.hpp](#).

```
00299     {  
00300     implementation->wait_for_kernel_execution();  
00301     }
```

**8.3.2.16.3.11 wait\_and\_throw()**

```
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.

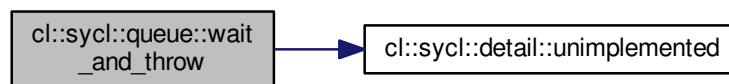
If no `async_handler` was provided then asynchronous exceptions will be lost.

Definition at line 314 of file [queue.hpp](#).

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

```
00314         {
00315         // \todo Implement the throw part of wait_and_throw
00316         wait();
00317         detail::unimplemented();
00318     }
```

Here is the call graph for this function:

**8.3.2.16.4 Member Data Documentation****8.3.2.16.4.1 implementation\_t**

```
friend cl::sycl::queue::implementation_t [private]
```

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

**8.3.3 Typedef Documentation**

#### 8.3.3.1 `cpu_selector`

```
using cl::sycl::cpu_selector = typedef device_tynename_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 133 of file `device_selector_tail.hpp`.

#### 8.3.3.2 `default_selector`

```
using cl::sycl::default_selector = typedef device_tynename_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.

To influence the default device selection, use the Boost.Compute environment variables:

- `BOOST_COMPUTE_DEFAULT_DEVICE`
- `BOOST_COMPUTE_DEFAULT_DEVICE_TYPE`
- `BOOST_COMPUTE_DEFAULT_PLATFORM`
- `BOOST_COMPUTE_DEFAULT_VENDOR`

Definition at line 115 of file `device_selector_tail.hpp`.

#### 8.3.3.3 `device_exec_capabilities`

```
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 device\_fp\_config

```
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 device\_queue\_properties

```
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 gl\_context\_interop

```
using cl::sycl::info::gl_context_interop = typedef bool  
  
#include <include/CL/sycl/info/context.hpp>
```

Definition at line 22 of file [context.hpp](#).

#### 8.3.3.7 gpu\_selector

```
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 125 of file [device\\_selector\\_tail.hpp](#).

#### 8.3.3.8 host\_selector

```
using cl::sycl::host_selector = typedef device_typename_selector<info::device_type::host>  
  
#include <include/CL/sycl/device_selector/detail/device_selector_tail.hpp>
```

Selects the SYCL host CPU device that does not require an OpenCL runtime.

Definition at line 139 of file [device\\_selector\\_tail.hpp](#).

### 8.3.4 Enumeration Type Documentation

#### 8.3.4.1 context

```
enum cl::sycl::info::context : int [strong]

#include <include/CL/sycl/info/context.hpp>
```

Context information descriptors.

**Todo** Should be unsigned int to be consistent with others?

##### Enumerator

|                 |  |
|-----------------|--|
| reference_count |  |
| num_devices     |  |
| devices         |  |
| gl_interop      |  |

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

```
00028         : int {
00029     reference_count,
00030     num_devices,
00031     devices,
00032     gl_interop
00033 };
```

#### 8.3.4.2 device

```
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 |  |

## Enumerator

|                                  |  |
|----------------------------------|--|
| max_work_item_dimensions         |  |
| max_work_item_sizes              |  |
| max_work_group_size              |  |
| preferred_vector_width_char      |  |
| preferred_vector_width_short     |  |
| preferred_vector_width_int       |  |
| preferred_vector_width_long_long |  |
| preferred_vector_width_float     |  |
| preferred_vector_width_double    |  |
| preferred_vector_width_half      |  |
| native_vector_witdth_char        |  |
| native_vector_witdth_short       |  |
| native_vector_witdth_int         |  |
| native_vector_witdth_long_long   |  |
| native_vector_witdth_float       |  |
| native_vector_witdth_double      |  |
| native_vector_witdth_half        |  |
| max_clock_frequency              |  |
| address_bits                     |  |
| max_mem_alloc_size               |  |
| image_support                    |  |
| max_read_image_args              |  |
| max_write_image_args             |  |
| image2d_max_height               |  |
| image2d_max_width                |  |
| image3d_max_height               |  |
| image3d_max_widht                |  |
| image3d_mas_depth                |  |
| image_max_buffer_size            |  |
| image_max_array_size             |  |
| max_samplers                     |  |
| max_parameter_size               |  |
| mem_base_addr_align              |  |
| single_fp_config                 |  |
| double_fp_config                 |  |
| global_mem_cache_type            |  |
| global_mem_cache_line_size       |  |
| global_mem_cache_size            |  |
| global_mem_size                  |  |
| max_constant_buffer_size         |  |
| max_constant_args                |  |
| local_mem_type                   |  |
| local_mem_size                   |  |
| error_correction_support         |  |
| host_unified_memory              |  |
| profiling_timer_resolution       |  |
| endian_little                    |  |

## Enumerator

|                             |  |
|-----------------------------|--|
| is_available                |  |
| is_compiler_available       |  |
| is_linker_available         |  |
| execution_capabilities      |  |
| queue_properties            |  |
| built_in_kernels            |  |
| platform                    |  |
| name                        |  |
| vendor                      |  |
| driver_version              |  |
| profile                     |  |
| device_version              |  |
| opengl_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 };

```

#### 8.3.4.3 device\_affinity\_domain

```
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 };

```

#### 8.3.4.4 device\_execution\_capabilities

```
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 };
```

#### 8.3.4.5 device\_partition\_property

```
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 };
```

#### 8.3.4.6 device\_partition\_type

```
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 };

```

## 8.3.4.7 device\_type

```
enum cl::sycl::info::device_type : unsigned int [strong]
```

```
#include <include/CL/sycl/info/device.hpp>
```

Type of devices.

To be used either to define a device type or to select more broadly a kind of device

**Todo** To be moved in the specification from platform to device

**Todo** Add opencl to the specification

**Todo** there is no accelerator\_selector and custom\_accelerator

## Enumerator

|             |  |
|-------------|--|
| cpu         |  |
| gpu         |  |
| accelerator |  |
| custom      |  |
| defaults    |  |
| host        |  |
| opencl      |  |
| all         |  |

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

```

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

```

#### 8.3.4.8 fp\_config

```
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 };

```

#### 8.3.4.9 global\_mem\_cache\_type

```
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 };
```

#### 8.3.4.10 local\_mem\_type

```
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 };
```

#### 8.3.4.11 platform

```
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 <code>string_class</code> ) 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 <code>string_class</code> )                   |
| TRISYCL_SKIP_OPENCL                         | Returns the name of the platform (as a <code>string_class</code> )  |
| TRISYCL_SKIP_OPENCL                         | Returns the string provided by the platform vendor (as a <code>string_class</code> )  |
| TRISYCL_SKIP_OPENCL<br>Generated by Doxygen | Returns a space-separated list of extension names supported by the platform (as a <code>string_class</code> )                               |

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 `device::get_info< info::device::device_type >()`

```

template<>
auto cl::sycl::device::get_info< info::device::device_type > ( ) const [inline]

#include <include/CL/sycl/device.hpp>

```

Definition at line 270 of file [device.hpp](#).

References [cl::sycl::info::cpu](#).

```

00270         {
00271     return info::device_type::cpu;
00272 }

```

#### 8.3.5.2 `device::get_info< info::device::local_mem_size >()`

```
template<>
auto cl::sycl::device::get_info< info::device::local_mem_size > ( ) const [inline]

#include <include/CL/sycl/device.hpp>
```

Definition at line 275 of file [device.hpp](#).

```
00275                                     {
00276     return size_t { 32000 };
00277 }
```

#### 8.3.5.3 `device::get_info< info::device::max_compute_units >()`

```
template<>
auto cl::sycl::device::get_info< info::device::max_compute_units > ( ) const [inline]

#include <include/CL/sycl/device.hpp>
```

Definition at line 265 of file [device.hpp](#).

```
00265                                     {
00266     return size_t { 8 };
00267 }
```

#### 8.3.5.4 `device::get_info< info::device::max_mem_alloc_size >()`

```
template<>
auto cl::sycl::device::get_info< info::device::max_mem_alloc_size > ( ) const [inline]

#include <include/CL/sycl/device.hpp>
```

Definition at line 280 of file [device.hpp](#).

```
00280                                     {
00281     return size_t { 32000 };
00282 }
```

#### 8.3.5.5 `device::get_info< info::device::max_work_group_size >()`

```
template<>
auto cl::sycl::device::get_info< info::device::max_work_group_size > ( ) const [inline]

#include <include/CL/sycl/device.hpp>
```

Definition at line 260 of file [device.hpp](#).

```
00260                                     {
00261     return size_t { 8 };
00262 }
```

#### 8.3.5.6 `device::get_info< info::device::name >()`

```
template<>
auto cl::sycl::device::get_info< info::device::name > ( ) const [inline]

#include <include/CL/sycl/device.hpp>
```

Definition at line 290 of file [device.hpp](#).

```
00290                                     {
00291     return string_class {};
00292 }
```

#### 8.3.5.7 `device::get_info< info::device::profile >()`

```
template<>
auto cl::sycl::device::get_info< info::device::profile > ( ) const [inline]

#include <include/CL/sycl/device.hpp>
```

Definition at line 295 of file [device.hpp](#).

```
00295                                     {
00296     return string_class { "FULL_PROFILE" };
00297 }
```

**8.3.5.8** `device::get_info< info::device::vendor >()`

```
template<>
auto cl::sycl::device::get_info< info::device::vendor > ( ) const [inline]

#include <include/CL/sycl/device.hpp>
```

Definition at line 285 of file [device.hpp](#).

```
00285                                     {
00286     return string_class {};
```

```
00287 }
```

**8.3.5.9** `get_devices()` [1/3]

```
vector_class< cl::sycl::device > cl::sycl::detail::opencl_platform::get_devices (
    info::device_type device_type ) const [inline], [override], [virtual]

#include <include/CL/sycl/platform/detail/opencl_platform.hpp>
```

Get all the available devices for this OpenCL platform.

Returns a vector class containing all SYCL devices associated with this OpenCL platform.

**Parameters**

|    |                    |   |
|----|--------------------|---|
| in | <i>device_type</i> | is the device type to filter the selection or <a href="#">info::device_type::all</a> by default to return all the devices |
|----|--------------------|---|

**Returns**

the device list

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

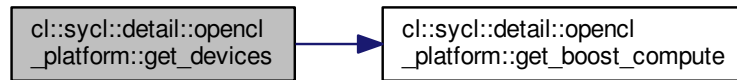
Definition at line 32 of file [opencl\\_platform\\_tail.hpp](#).

References [cl::sycl::detail::opencl\\_platform::get\\_boost\\_compute\(\)](#).

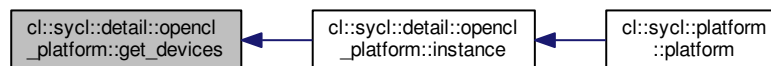
Referenced by [cl::sycl::detail::opencl\\_platform::instance\(\)](#).

```
00032                                     {
00033     vector_class<cl::sycl::device> devices;
00034     device_type_selector ds { device_type };
00035     // Add the desired OpenCL devices
00036     for (const auto &d : get_boost_compute().devices()) {
00037         // Get the SYCL device from the Boost Compute device
00038         cl::sycl::device sycl_dev { d };
00039         /* Return the devices with the good criterion according to the selector.
00040          By calling devices on the \c boost::compute::platform we know that
00041          we iterate only over the device belonging to the current platform,
00042          */
00043         if (ds(sycl_dev) > 0)
00044             devices.push_back(sycl_dev);
00045     }
00046     return devices;
00047 }
00048 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.3.5.10 `get_devices()` [2/3]

```
vector_class< cl::sycl::device > cl::sycl::detail::host_platform::get_devices (
    info::device_type device_type ) const [inline], [override], [virtual]
```

```
#include <include/CL/sycl/platform/detail/host_platform.hpp>
```

Get all the available devices for the host platform.

Get all the available devices for this platform.

##### Parameters

|    |                          |  |
|----|--------------------------|--|
| in | <code>device_type</code> | is the device type to filter the selection or <code>info::device_type::all</code> by default to return all the devices |
|----|--------------------------|--|

##### Returns

the device list

If `get_devices` is called with the host platform and the right device type, returns the `host_device`.

Implements `cl::sycl::detail::platform`.

Definition at line 31 of file `host_platform_tail.hpp`.

Referenced by `cl::sycl::detail::host_platform::has_extension()`.

```

00031                                     {
00032     /** If \c get_devices is called with the host platform
00033         and the right device type, returns the host_device.
00034     */
00035     if (device_type_selector { device_type }(cl::sycl::device {}) > 0)
00036         // Return 1 default device, i.e. the host device
00037         return { {} };
00038     else
00039         // No matching device
00040         return {};
00041 }

```

Here is the caller graph for this function:



#### 8.3.5.11 get\_devices() [3/3]

```

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](#).

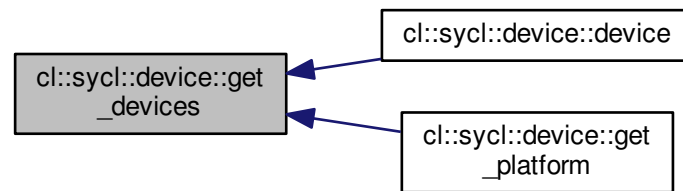
Referenced by [cl::sycl::device::device\(\)](#), and [cl::sycl::device::get\\_platform\(\)](#).

```

00026                                     {
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 }

```

Here is the caller graph for this function:



### 8.3.6 Variable Documentation

#### 8.3.6.1 TRISYCL\_WEAK\_ATTRIB\_SUFFIX

```
TRISYCL_WEAK_ATTRIB_PREFIX detail::cache< cl_command_queue, detail::opencl_queue > opencl_↵
queue::cache cl::sycl::detail::TRISYCL_WEAK_ATTRIB_SUFFIX
```

```
#include <include/CL/sycl/context/detail/opencl_context.hpp>
```

Definition at line 142 of file `opencl_context.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()`.



## 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 `expand()` [1/3]

```
template<typename V , typename Tuple , bool expansion = false>
static auto cl::sycl::detail::expand\_to\_vector< V, Tuple, expansion >::expand (
    Tuple t ) [inline], [static]
```

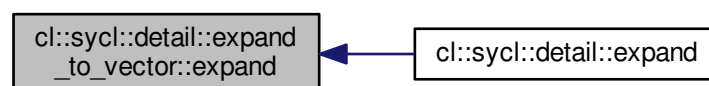
```
#include <include/CL/sycl/detail/array\_tuple\_helpers.hpp>
```

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** `expand()` [2/3]

```
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** `expand()` [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.

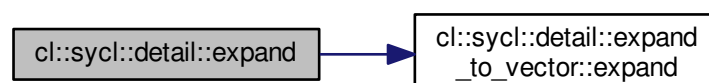
If there is only 1 initializer, this is a scalar initialization of a vector and the value is expanded to all the vector elements first.

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 fill\_tuple()

```
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 tuple\_to\_array()

```
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 tuple\_to\_array\_iterate()

```
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 Some helpers for the implementation

### Classes

- struct `cl::sycl::detail::container_element_aspect< T >`  
A mix-in to add some container element aspects. [More...](#)
- 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 `Dimensions = 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.
- `#define TRISYCL_LOGICAL_OPERATOR_VECTOR_OP(op)`

### 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.

#### 8.5.1 Detailed Description

#### 8.5.2 Class Documentation

##### 8.5.2.1 struct `cl::sycl::detail::container_element_aspect`

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

A mix-in to add some container element aspects.

Definition at line 23 of file [container\\_element\\_aspect.hpp](#).

#### Public Types

- using `value_type` = `T`
- using `pointer` = `value_type *`
- using `const_pointer` = `const value_type *`
- using `reference` = `value_type &`
- using `const_reference` = `const value_type &`

## 8.5.2.1.1 Member Typedef Documentation

## 8.5.2.1.1.1 const\_pointer

```
template<typename T>
using cl::sycl::detail::container_element_aspect< T >::const_pointer = const value_type*
```

Definition at line 27 of file [container\\_element\\_aspect.hpp](#).

## 8.5.2.1.1.2 const\_reference

```
template<typename T>
using cl::sycl::detail::container_element_aspect< T >::const_reference = const value_type&
```

Definition at line 29 of file [container\\_element\\_aspect.hpp](#).

## 8.5.2.1.1.3 pointer

```
template<typename T>
using cl::sycl::detail::container_element_aspect< T >::pointer = value_type*
```

Definition at line 26 of file [container\\_element\\_aspect.hpp](#).

## 8.5.2.1.1.4 reference

```
template<typename T>
using cl::sycl::detail::container_element_aspect< T >::reference = value_type&
```

Definition at line 28 of file [container\\_element\\_aspect.hpp](#).

## 8.5.2.1.1.5 value\_type

```
template<typename T>
using cl::sycl::detail::container_element_aspect< T >::value_type = T
```

Definition at line 25 of file [container\\_element\\_aspect.hpp](#).

## 8.5.2.2 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.

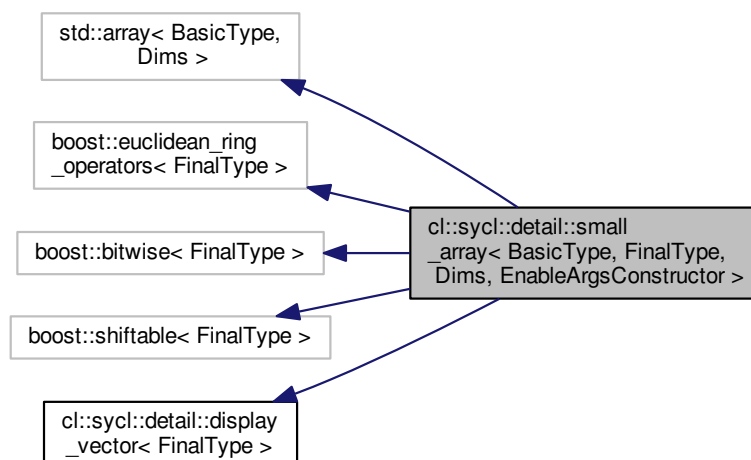
## Parameters

|                              |   |
|------------------------------|---|
| <i>BasicType</i>             | is the type element, such as int  |
| <i>Dims</i>                  | is the dimension number, typically between 1 and 3  |
| <i>FinalType</i>             | is the final type, such as range<> or id<>, so that boost::operator 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.

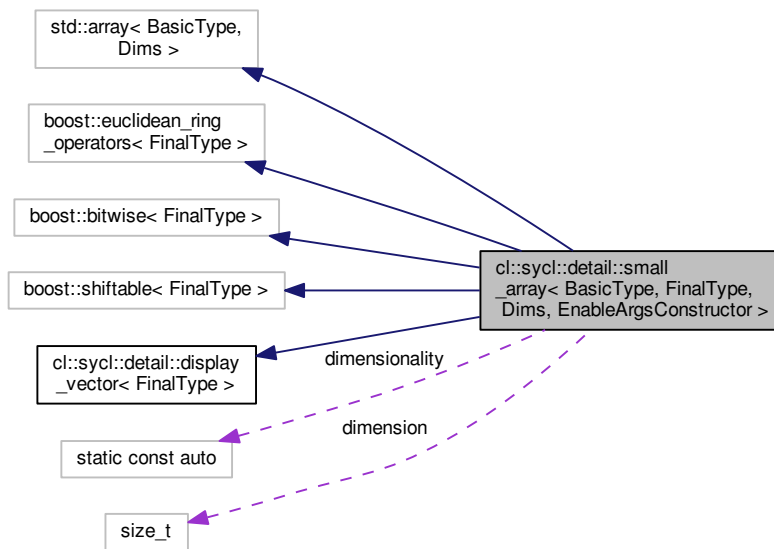
Definition at line 74 of file [small\\_array.hpp](#).

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.*
- `BasicType & x ()`  
*An accessor to the first variable of a small array.*
- `BasicType & y ()`  
*An accessor to the second variable of a small array.*
- `BasicType & z ()`  
*An accessor to the third variable of a small 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

### 8.5.2.2.1 Member Typedef Documentation

#### 8.5.2.2.1.1 [element\\_type](#)

```
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 94 of file [small\\_array.hpp](#).

### 8.5.2.2.2 Constructor & Destructor Documentation

#### 8.5.2.2.2.1 [small\\_array\(\)](#) [1/5]

```
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 103 of file [small\\_array.hpp](#).

```
00103                                     {
00104     // (*this)[0] is the first element of the underlying array
00105     std::copy_n(src, Dims, &(*this)[0]);
00106 }
```

#### 8.5.2.2.2.2 [small\\_array\(\)](#) [2/5]

```
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, SourceEnableArgs↵
Constructor > & src ) [inline]
```

A constructor from another [small\\_array](#) of the same size.

Definition at line 137 of file [small\\_array.hpp](#).

```
00140                                     {
00141     std::copy_n(&src[0], Dims, &(*this)[0]);
00142 }
```

**8.5.2.2.3** `small_array()` [3/5]

```
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 160 of file `small_array.hpp`.

```
00161 : std::array<BasicType, Dims> {
00162 // Allow a loss of precision in initialization with the static_cast
00163 { static_cast<BasicType>(args)... }
00164 }
```

**8.5.2.2.4** `small_array()` [4/5]

```
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 174 of file `small_array.hpp`.

```
00175 : std::array<BasicType, Dims>(src) {}
```

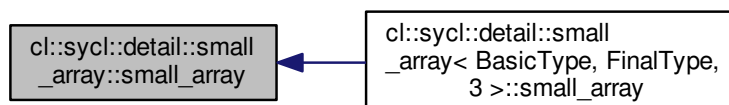
**8.5.2.2.5** `small_array()` [5/5]

```
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:



### 8.5.2.2.3 Member Function Documentation

#### 8.5.2.2.3.1 get()

```
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 185 of file [small\\_array.hpp](#).

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

#### 8.5.2.2.3.2 operator FinalType()

```
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 228 of file [small\\_array.hpp](#).

```
00228                                     {
00229     return *static_cast<FinalType *>(this);
00230 }
```

## 8.5.2.2.3.3 x()

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
BasicType& cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor
>::x ( ) [inline]
```

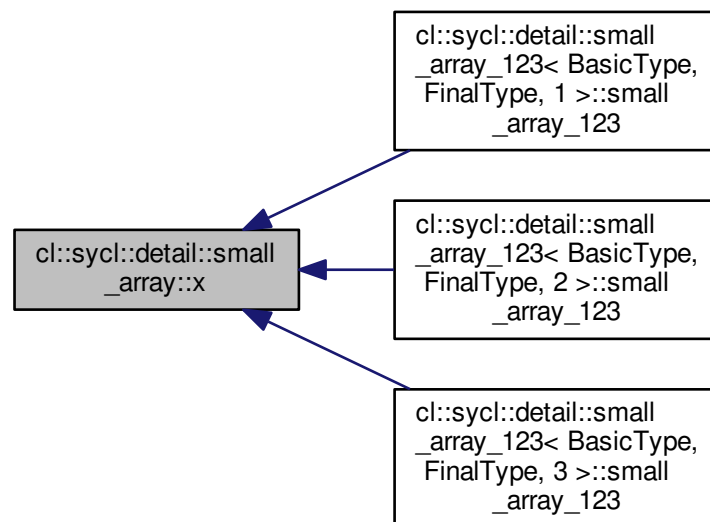
An accessor to the first variable of a small array.

Definition at line 111 of file [small\\_array.hpp](#).

Referenced by [cl::sycl::detail::small\\_array\\_123< BasicType, FinalType, 1 >::small\\_array\\_123\(\)](#), [cl::sycl::detail::small\\_array\\_123< BasicType, FinalType, 2 >::small\\_array\\_123\(\)](#), and [cl::sycl::detail::small\\_array\\_123< BasicType, FinalType, 3 >::small\\_array\\_123\(\)](#).

```
00111     {
00112     static_assert(Dims >= 1, "can't access to small_array[0] if Dims < 1");
00113     return (*this)[0];
00114 }
```

Here is the caller graph for this function:



## 8.5.2.2.3.4 y()

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
BasicType& cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor
>::y ( ) [inline]
```

An accessor to the second variable of a small array.

Definition at line 119 of file [small\\_array.hpp](#).

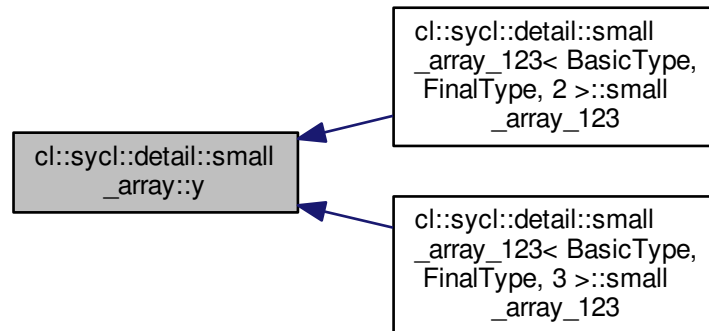
Referenced by [cl::sycl::detail::small\\_array\\_123< BasicType, FinalType, 2 >::small\\_array\\_123\(\)](#), and [cl::sycl::detail::small\\_array\\_123< BasicType, FinalType, 3 >::small\\_array\\_123\(\)](#).

```

00119     {
00120         static_assert(Dims >= 2, "can't access to small_array[1] if Dims < 2");
00121         return (*this)[1];
00122     }

```

Here is the caller graph for this function:



#### 8.5.2.2.3.5 z()

```

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

```

An accessor to the third variable of a small array.

Definition at line 127 of file [small\\_array.hpp](#).

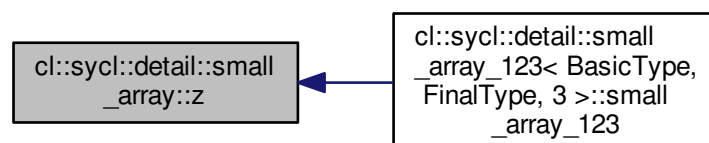
Referenced by [cl::sycl::detail::small\\_array\\_123< BasicType, FinalType, 3 >::small\\_array\\_123\(\)](#).

```

00127     {
00128         static_assert(Dims >= 3, "can't access to small_array[2] if Dims < 3");
00129         return (*this)[2];
00130     }

```

Here is the caller graph for this function:



## 8.5.2.2.4 Member Data Documentation

## 8.5.2.2.4.1 dimension

```
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 93 of file [small\\_array.hpp](#).

## 8.5.2.2.4.2 dimensionality

```
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 89 of file [small\\_array.hpp](#).

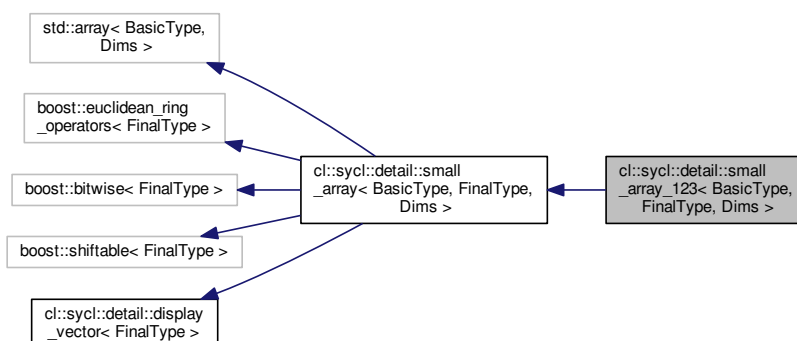
## 8.5.2.3 struct cl::sycl::detail::small\_array\_123

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

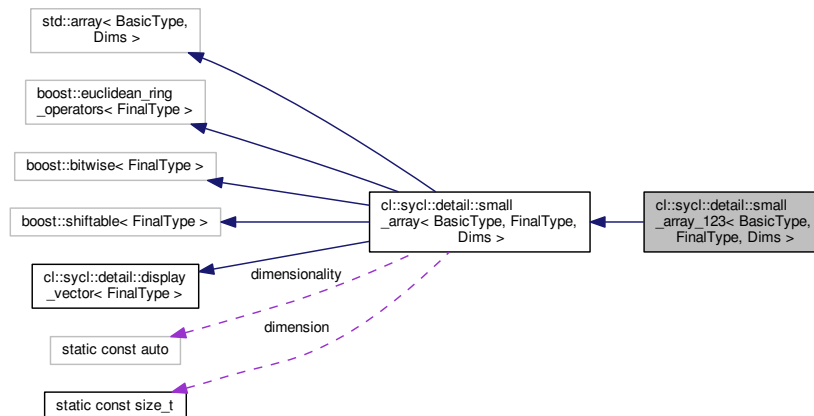
A small array of 1, 2 or 3 elements with the implicit constructors.

Definition at line 237 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

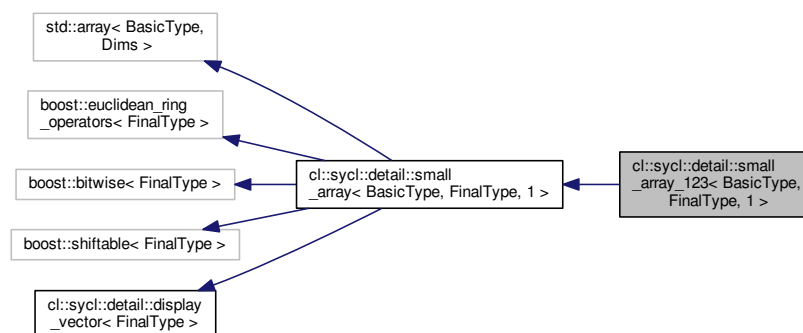
### 8.5.2.4 struct `cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >`

```
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 `Dimensions = 1`.

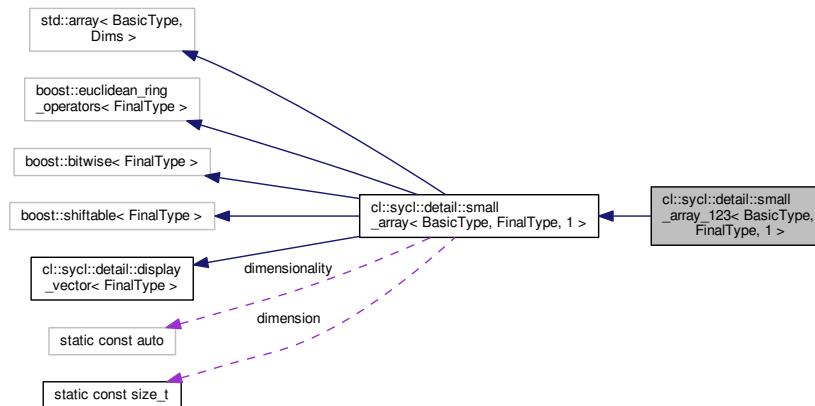
Definition at line 249 of file [small\\_array.hpp](#).

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

#### 8.5.2.4.1 Constructor & Destructor Documentation

##### 8.5.2.4.1.1 `small_array_123()` [1/2]

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >::small_array_123 (
    BasicType x ) [inline]
```

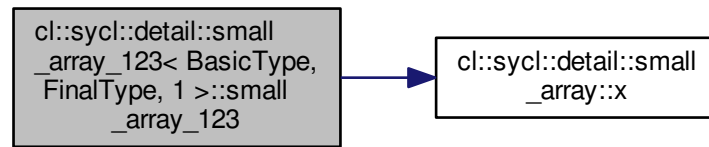
A 1-D constructor to have implicit conversion from from 1 integer and automatic inference of the dimensionality.

Definition at line 253 of file `small_array.hpp`.

References `cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::x()`.

```
00253                                     {
00254     (*this)[0] = x;
00255 }
```

Here is the call graph for this function:



#### 8.5.2.4.1.2 `small_array_123()` [2/2]

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >::small_array_123 ( ) [default]
```

Keep other constructors.

#### 8.5.2.4.2 Member Function Documentation

##### 8.5.2.4.2.1 `operator BasicType()`

```
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 265 of file [small\\_array.hpp](#).

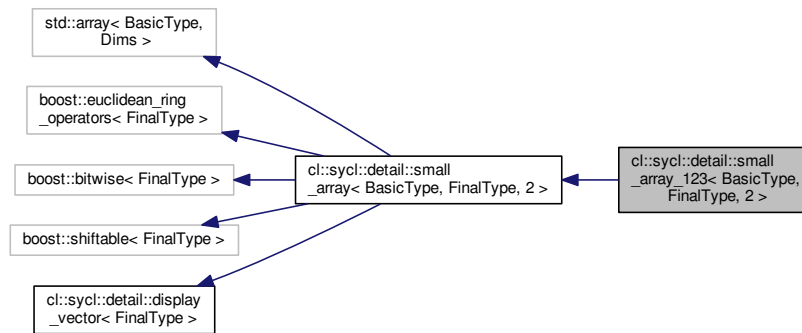
```
00265     {
00266         return (*this)[0];
00267     }
```

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

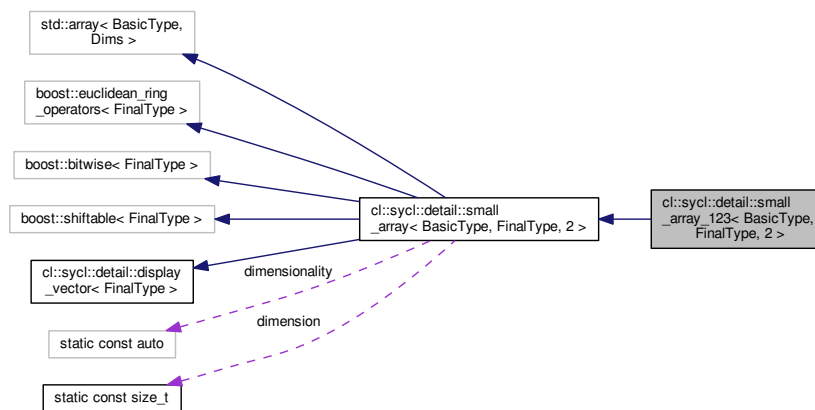
```
template<typename BasicType, typename FinalType>
struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >
```

Definition at line 272 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

### 8.5.2.5.1 Constructor & Destructor Documentation

#### 8.5.2.5.1.1 `small_array_123()` [1/3]

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >::small_array_123 (
    BasicType x,
    BasicType y ) [inline]
```

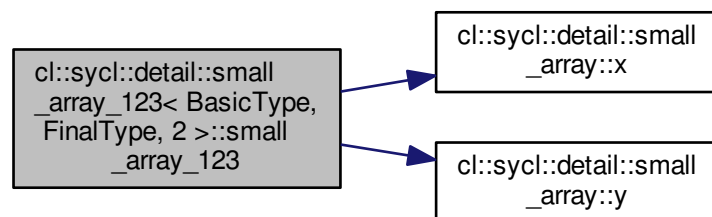
A 2-D constructor to have implicit conversion from from 2 integers and automatic inference of the dimensionality.

Definition at line 276 of file `small_array.hpp`.

References `cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::x()`, and `cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::y()`.

```
00276                                     {
00277     (*this)[0] = x;
00278     (*this)[1] = y;
00279 }
```

Here is the call graph for this function:



#### 8.5.2.5.1.2 `small_array_123()` [2/3]

```
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 287 of file `small_array.hpp`.

```
00287 : small_array_123 { e, e } { }
```

8.5.2.5.1.3 `small_array_123()` [3/3]

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >::small_array_123 ( ) [default]
```

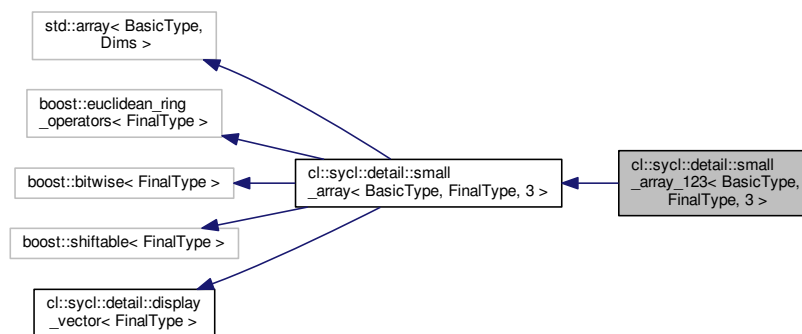
Keep other constructors.

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

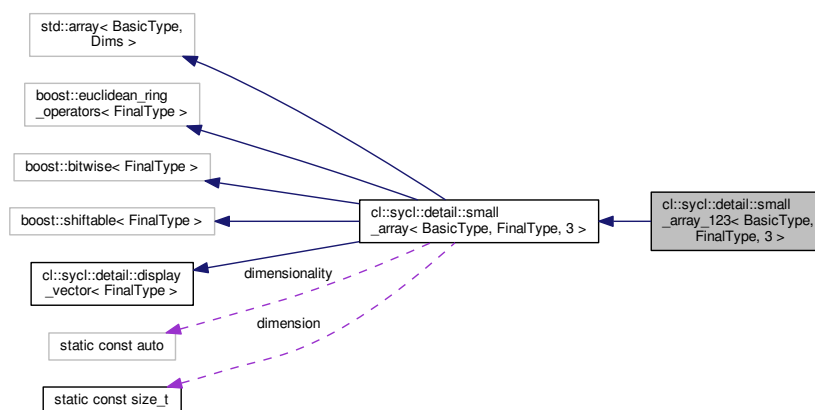
```
template<typename BasicType, typename FinalType>
struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >
```

Definition at line 298 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

### 8.5.2.6.1 Constructor & Destructor Documentation

#### 8.5.2.6.1.1 [small\\_array\\_123](#)() [1/3]

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >::small_array_123 (
    BasicType x,
    BasicType y,
    BasicType z ) [inline]
```

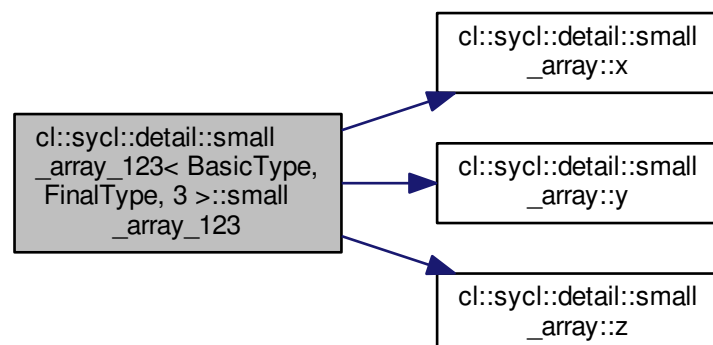
A 3-D constructor to have implicit conversion from from 3 integers and automatic inference of the dimensionality.

Definition at line 302 of file [small\\_array.hpp](#).

References [cl::sycl::detail::small\\_array< BasicType, FinalType, Dims, EnableArgsConstructor >::x\(\)](#), [cl::sycl::detail::small\\_array< BasicType, FinalType, Dims, EnableArgsConstructor >::y\(\)](#), and [cl::sycl::detail::small\\_array< BasicType, FinalType, Dims, EnableArgsConstructor >::z\(\)](#).

```
00302                                     {
00303     (*this)[0] = x;
00304     (*this)[1] = y;
00305     (*this)[2] = z;
00306 }
```

Here is the call graph for this function:



8.5.2.6.1.2 `small_array_123()` [2/3]

```
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 314 of file `small_array.hpp`.

```
00314 : small_array_123 { e, e, e } { }
```

8.5.2.6.1.3 `small_array_123()` [3/3]

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >::small_array_123 ( ) [default]
```

Keep other constructors.

## 8.5.3 Macro Definition Documentation

8.5.3.1 `TRISYCL_BOOST_OPERATOR_VECTOR_OP`

```
#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()`.

### 8.5.3.2 TRISYCL\_LOGICAL\_OPERATOR\_VECTOR\_OP

```
#define TRISYCL_LOGICAL_OPERATOR_VECTOR_OP (
    op )
```

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

#### Value:

```
FinalType operator op(const FinalType &rhs) {
    FinalType res;
    for (std::size_t i = 0; i != Dims; ++i)
        res[i] = (*this)[i] op rhs[i];
    return res;
}
```

Definition at line 41 of file [small\\_array.hpp](#).

Referenced by [cl::sycl::detail::small\\_array< BasicType, FinalType, 3 >::get\(\)](#).

## 8.5.4 Function Documentation

### 8.5.4.1 linear\_id()

```
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.

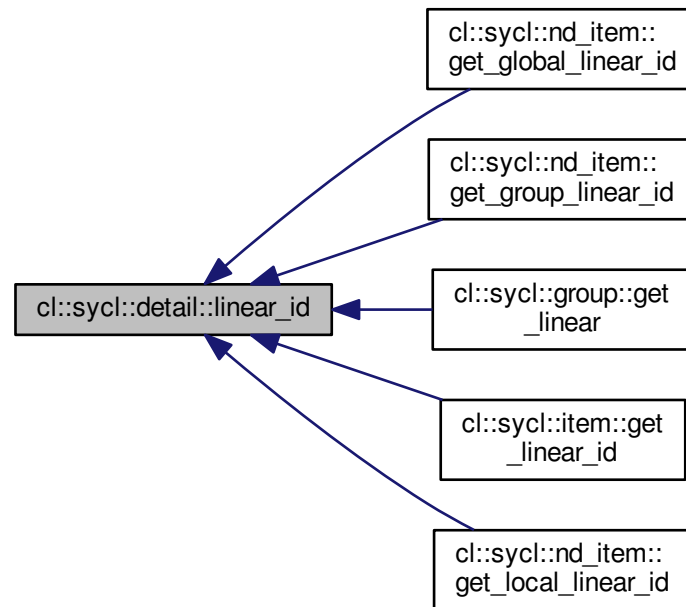
Definition at line 28 of file [linear\\_id.hpp](#).

Referenced by [cl::sycl::nd\\_item< Dimensions >::get\\_global\\_linear\\_id\(\)](#), [cl::sycl::nd\\_item< Dimensions >::get\\_group\\_linear\\_id\(\)](#), [cl::sycl::group< Dimensions >::get\\_linear\(\)](#), [cl::sycl::item< Dimensions >::get\\_linear\\_id\(\)](#), and [cl::sycl::nd\\_item< Dimensions >::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:



#### 8.5.4.2 unimplemented()

```
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`.

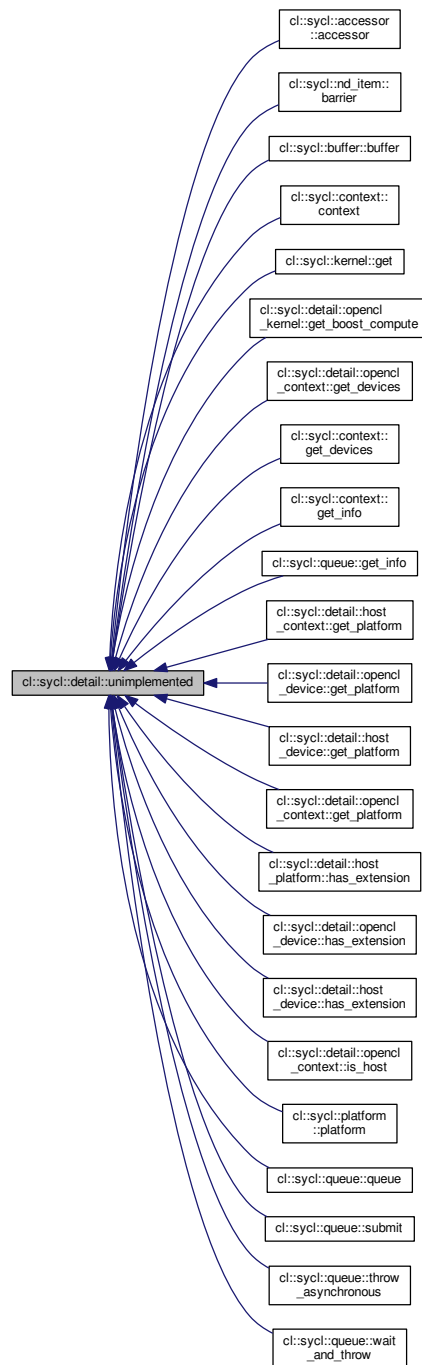
Referenced by `cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor()`, `cl::sycl::nd_item< Dimensions >::barrier()`, `cl::sycl::buffer< T, Dimensions, Allocator >::buffer()`, `cl::sycl::context::context()`, `cl::sycl::kernel::get()`, `cl::sycl::detail::opencil_kernel::get_boost_compute()`, `cl::sycl::detail::opencil_context::get_devices()`, `cl::sycl::context::get_devices()`, `cl::sycl::context::get_info()`, `cl::sycl::queue::get_info()`, `cl::sycl::detail::host_context::get_platform()`, `cl::sycl::detail::opencil_device::get_platform()`, `cl::sycl::detail::host_device::get_platform()`, `cl::sycl::detail::opencil_context::get_platform()`, `cl::sycl::detail::host_platform::has_extension()`, `cl::sycl::detail::opencil_device::has_extension()`, `cl::sycl::detail::host_device::has_extension()`, `cl::sycl::detail::opencil_context::is_host()`, `cl::sycl::platform::platform()`, `cl::sycl::queue::queue()`, `cl::sycl::queue::submit()`, `cl::sycl::queue::throw_asynchronous()`, and `cl::sycl::queue::wait_and_throw()`.

```

00025                                     {
00026 #ifndef NDEBUG
00027     std::cerr << "Error: using a non implemented feature!!!" << std::endl
00028     << "Please contribute to the open source implementation. :-)"
00029     << std::endl;
00030 #endif
00031 }

```

Here is the caller graph for this function:



## 8.6 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.*

#### 8.6.1 Detailed Description

#### 8.6.2 Class Documentation

##### 8.6.2.1 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

|                |   |
|----------------|---|
| <code>T</code> | is the real type name to be used in the debug output. |
|----------------|---|

Definition at line 68 of file [debug.hpp](#).

##### 8.6.2.2 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

|                |   |
|----------------|---|
| <code>T</code> | 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.*

### 8.6.2.2.1 Member Function Documentation

#### 8.6.2.2.1.1 display()

```
template<typename T>
void cl::sycl::detail::display_vector< T >::display ( ) const [inline]
```

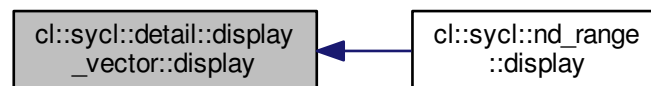
To debug and test.

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

Referenced by [cl::sycl::nd\\_range< Dimensions >::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:



### 8.6.3 Function Documentation

### 8.6.3.1 `trace_kernel()`

```
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.

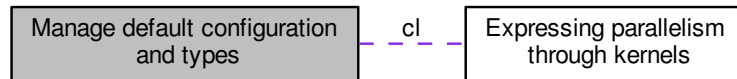
Definition at line 130 of file `debug.hpp`.

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 }
```

## 8.7 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 TRISYCL_CL_LANGUAGE_VERSION 220`  
*This implement triSYCL 2.2.*
- `#define __SYCL_SINGLE_SOURCE__`  
*This source is compiled by a single source compiler.*
- `#define TRISYCL_MAKE_BOOST_CIRCULARBUFFER_THREAD_SAFE`
- `#define TRISYCL_SKIP_OPENCL(x) x`  
*Define TRISYCL\_OPENCL to add OpenCL.*

#### 8.7.1 Detailed Description

#### 8.7.2 Macro Definition Documentation

##### 8.7.2.1 `__SYCL_SINGLE_SOURCE__`

```
#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](#).

### 8.7.2.2 CL\_SYCL\_LANGUAGE\_VERSION

```
#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](#).

### 8.7.2.3 TRISYCL\_CL\_LANGUAGE\_VERSION

```
#define TRISYCL_CL_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](#).

### 8.7.2.4 TRISYCL\_MAKE\_BOOST\_CIRCULARBUFFER\_THREAD\_SAFE

```
#define TRISYCL_MAKE_BOOST_CIRCULARBUFFER_THREAD_SAFE
```

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

Definition at line 33 of file [global\\_config.hpp](#).

### 8.7.2.5 TRISYCL\_SKIP\_OPENCL

```
#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 51 of file [global\\_config.hpp](#).

## 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

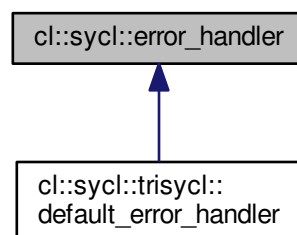
### 8.8.2 Class Documentation

#### 8.8.2.1 struct `cl::sycl::error_handler`

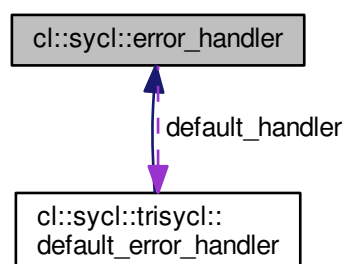
User supplied error handler to call a user-provided function when an error happens from a SYCL object that was constructed with this error handler.

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.*
- virtual [~error\\_handler](#) ()=0

## Static Public Attributes

- static [trisycl::default\\_error\\_handler](#) [default\\_handler](#)  
*Add a default\_handler to be used by default.*

### 8.8.2.1.1 Constructor & Destructor Documentation

#### 8.8.2.1.1.1 ~error\_handler()

```
virtual cl::sycl::error_handler::~~error_handler ( ) [pure virtual]
```

### 8.8.2.1.2 Member Function Documentation

#### 8.8.2.1.2.1 report\_error()

```
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](#).

### 8.8.2.1.3 Member Data Documentation

#### 8.8.2.1.3.1 default\_handler

```
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](#).

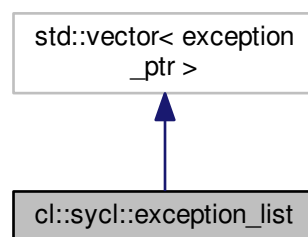
### 8.8.2.2 struct cl::sycl::exception\_list

Exception list to store several exceptions.

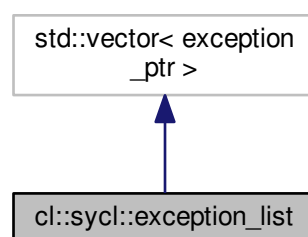
**Todo** Do we need to define it in SYCL or can we rely on plain C++17 one?

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

Inheritance diagram for cl::sycl::exception\_list:



Collaboration diagram for cl::sycl::exception\_list:

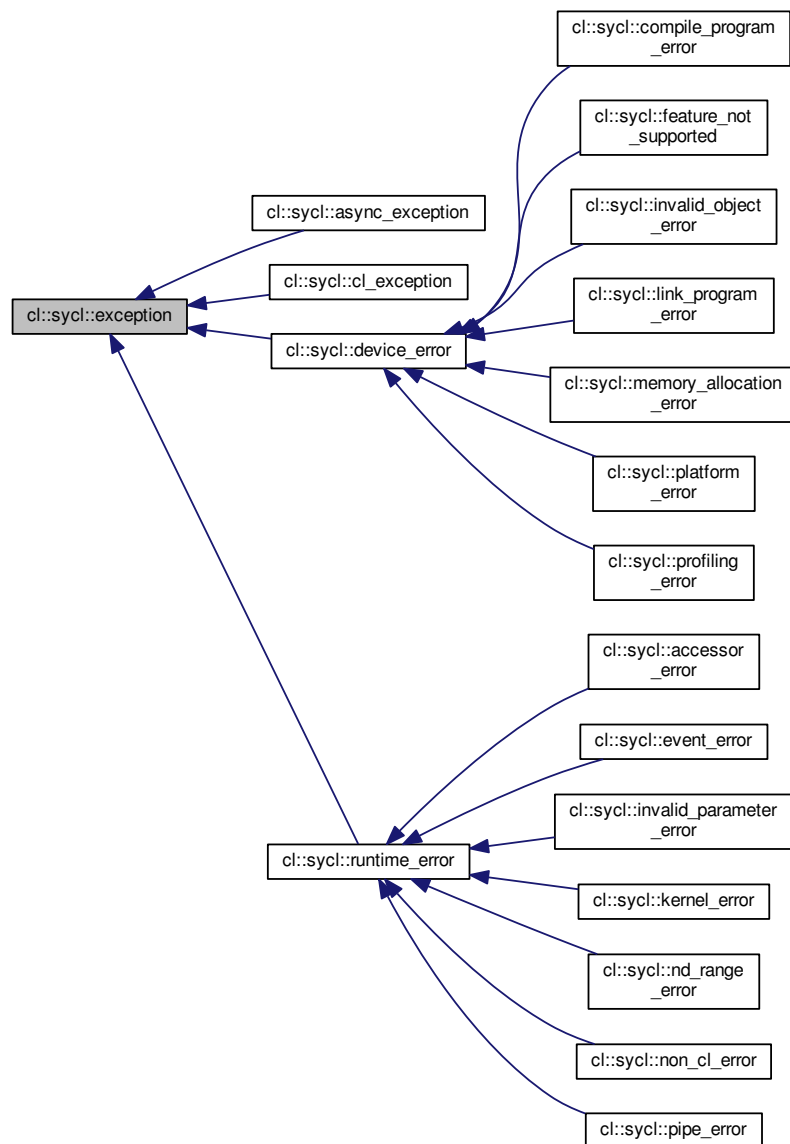


### 8.8.2.3 class cl::sycl::exception

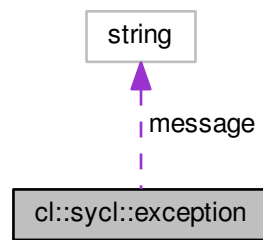
Encapsulate a SYCL error information.

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 exception()

```
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 what()

```
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 message

```
string_class cl::sycl::exception::message [private]
```

The error message to return.

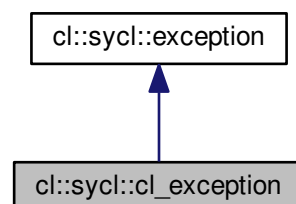
Definition at line 44 of file [exception.hpp](#).

#### 8.8.2.4 class cl::sycl::cl\_exception

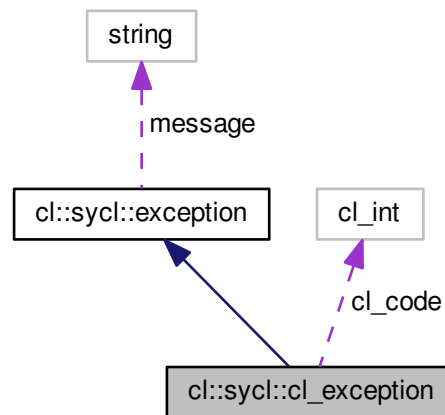
Returns the OpenCL error code encapsulated in the exception.

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_exception()`

```

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 get\_cl\_code()

```
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\_code

```
cl_int cl::sycl::cl_exception::cl_code [private]
```

The OpenCL error code to return.

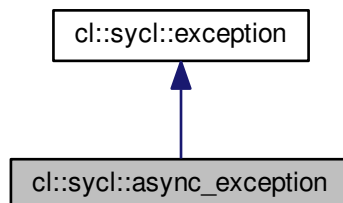
Definition at line 74 of file [exception.hpp](#).

#### 8.8.2.5 struct cl::sycl::async\_exception

An error stored in an [exception\\_list](#) for asynchronous errors.

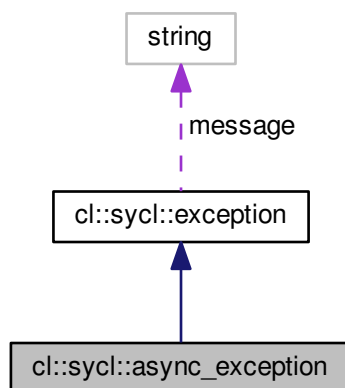
Definition at line 93 of file [exception.hpp](#).

Inheritance diagram for cl::sycl::async\_exception:





Collaboration diagram for `cl::sycl::async_exception`:

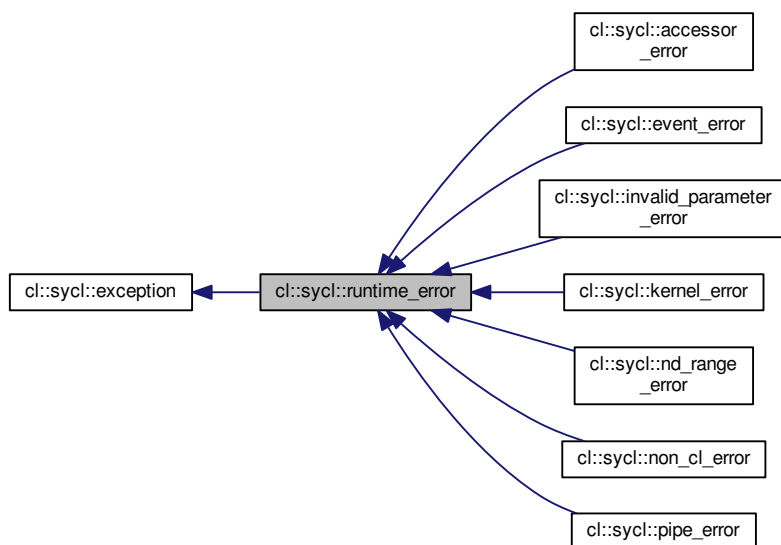


### Additional Inherited Members

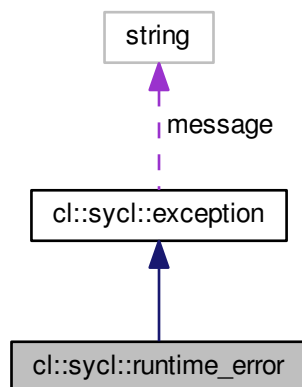
#### 8.8.2.6 class `cl::sycl::runtime_error`

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

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



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



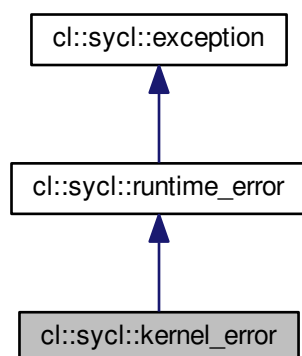
### Additional Inherited Members

#### 8.8.2.7 class `cl::sycl::kernel_error`

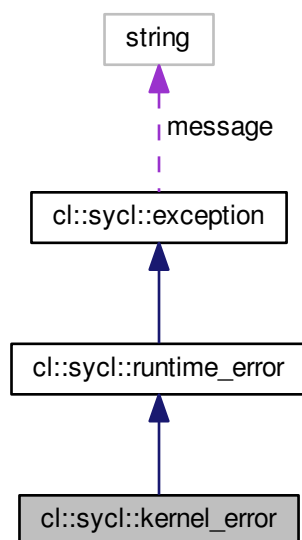
Error that occurred before or while enqueueing the SYCL kernel.

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

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



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



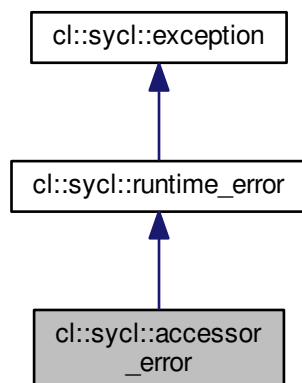
### Additional Inherited Members

#### 8.8.2.8 class `cl::sycl::accessor_error`

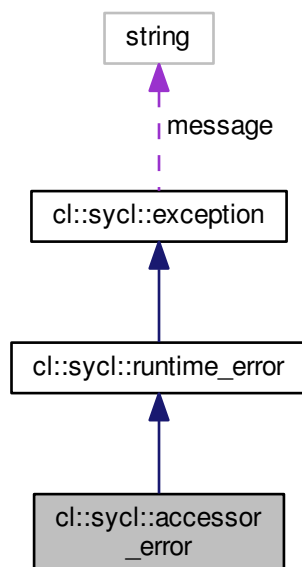
Error regarding the `cl::sycl::accessor` objects defined.

Definition at line 110 of file `exception.hpp`.

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



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



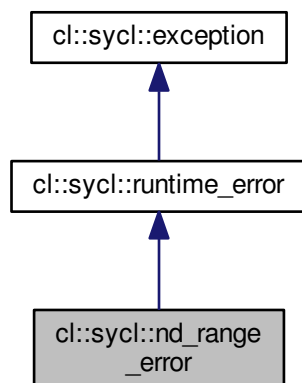
## Additional Inherited Members

### 8.8.2.9 class `cl::sycl::nd_range_error`

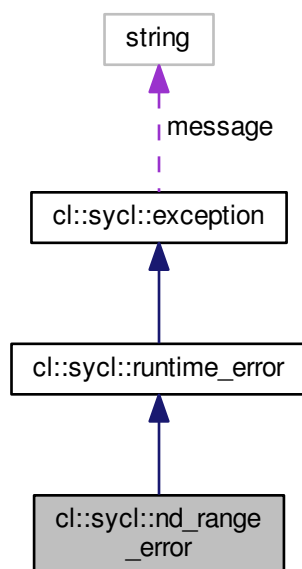
Error regarding the `cl::sycl::nd_range` specified for the SYCL kernel.

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

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



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



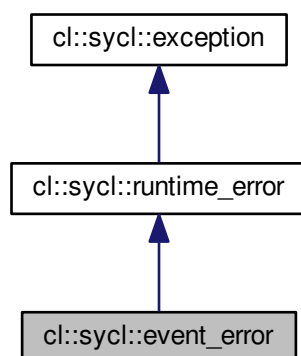
### Additional Inherited Members

#### 8.8.2.10 class `cl::sycl::event_error`

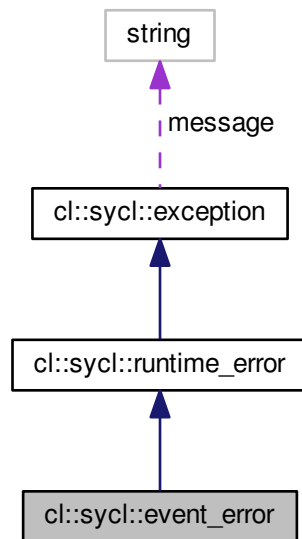
Error regarding associated `cl::sycl::event` objects.

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

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



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



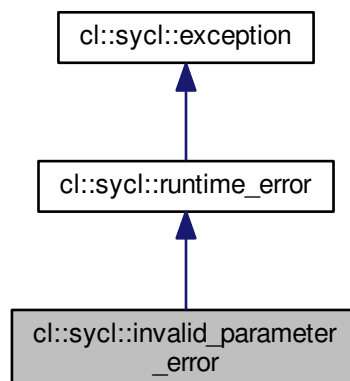
### Additional Inherited Members

#### 8.8.2.11 class `cl::sycl::invalid_parameter_error`

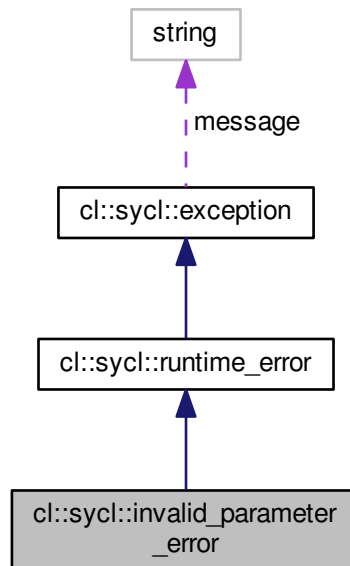
Error regarding parameters to the SYCL kernel, it may apply to any captured parameters to the kernel lambda.

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

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



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



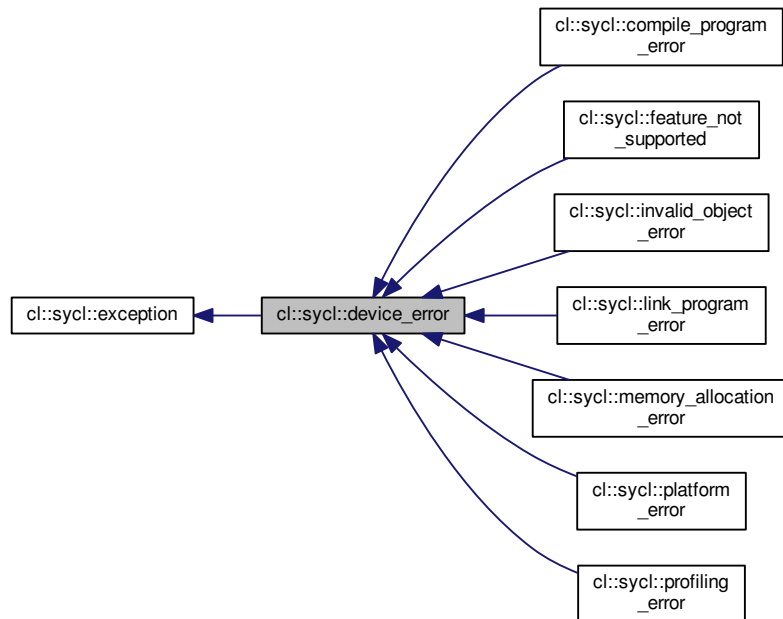
### Additional Inherited Members

#### 8.8.2.12 class `cl::sycl::device_error`

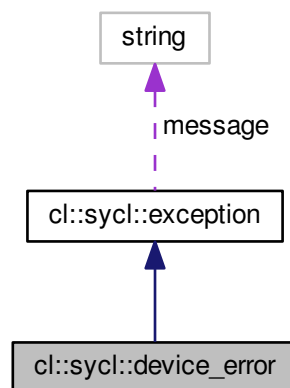
The SYCL device will trigger this exception on error.

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

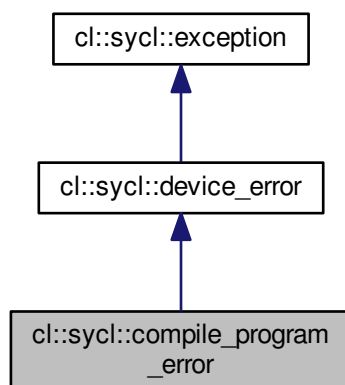
### 8.8.2.13 class `cl::sycl::compile_program_error`

Error while compiling the SYCL kernel to a SYCL device.

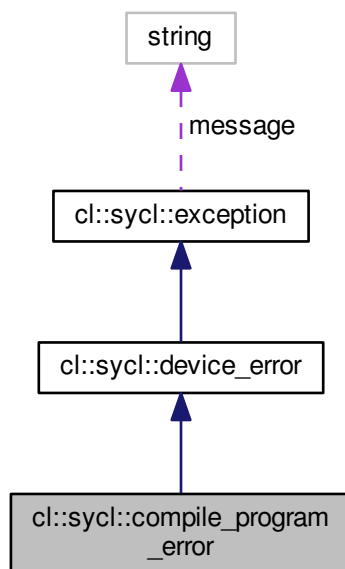


Definition at line 142 of file [exception.hpp](#).

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



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



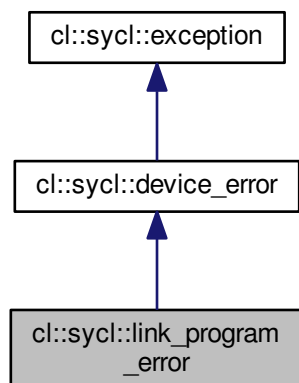
#### Additional Inherited Members

#### 8.8.2.14 class `cl::sycl::link_program_error`

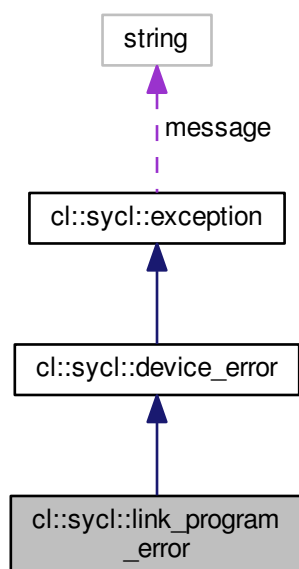
Error while linking the SYCL kernel to a SYCL device.

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

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



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



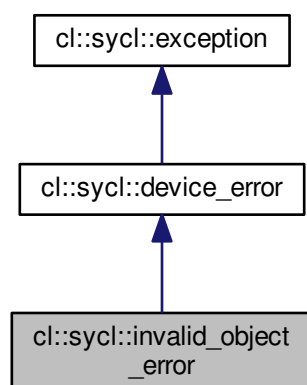
### Additional Inherited Members

#### 8.8.2.15 class `cl::sycl::invalid_object_error`

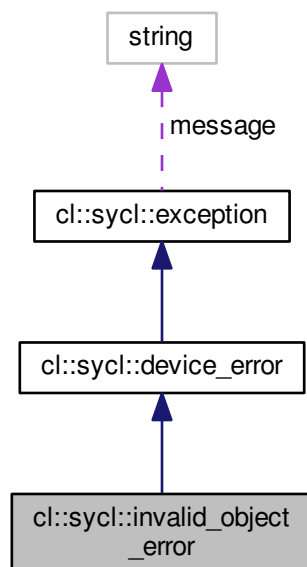
Error regarding any memory objects being used inside the kernel.

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

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



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



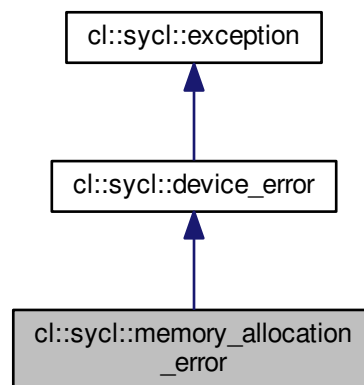
## Additional Inherited Members

### 8.8.2.16 class `cl::sycl::memory_allocation_error`

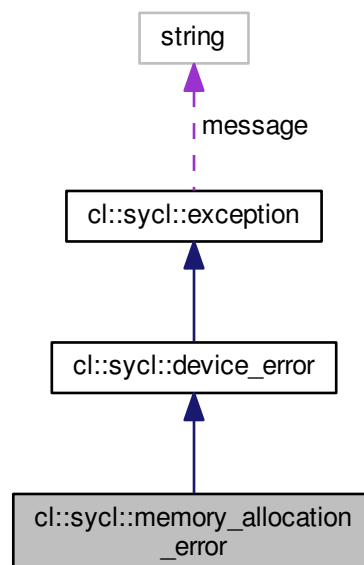
Error on memory allocation on the SYCL device for a SYCL kernel.

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

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



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



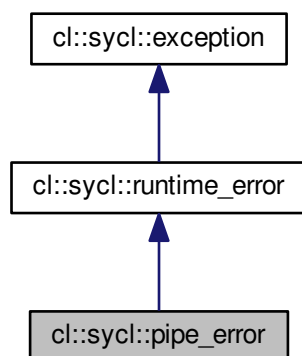
## Additional Inherited Members

### 8.8.2.17 class `cl::sycl::pipe_error`

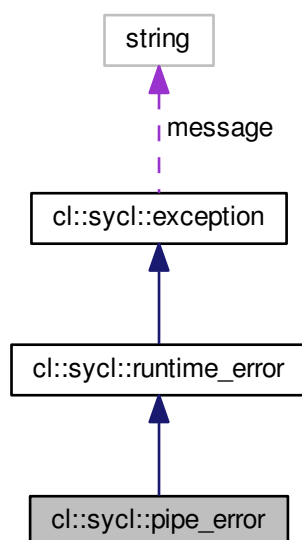
A failing pipe error will trigger this exception on error.

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

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



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



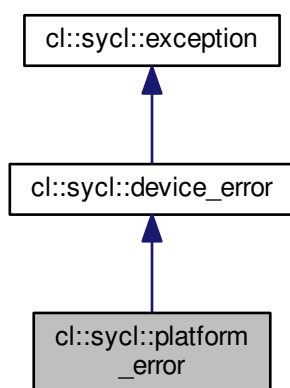
## Additional Inherited Members

### 8.8.2.18 class `cl::sycl::platform_error`

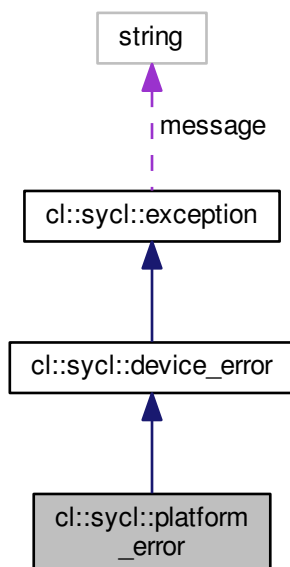
The SYCL platform will trigger this exception on error.

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

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



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



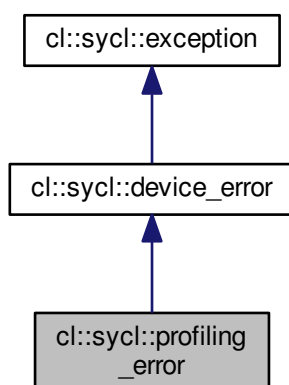
### Additional Inherited Members

#### 8.8.2.19 class `cl::sycl::profiling_error`

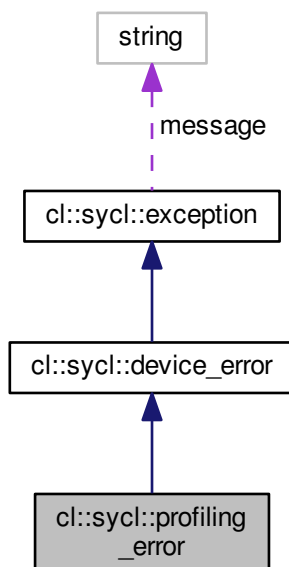
The SYCL runtime will trigger this error if there is an error when profiling info is enabled.

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

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



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



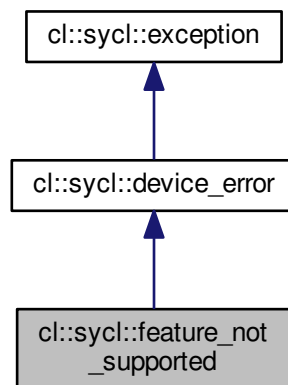
## Additional Inherited Members

### 8.8.2.20 class `cl::sycl::feature_not_supported`

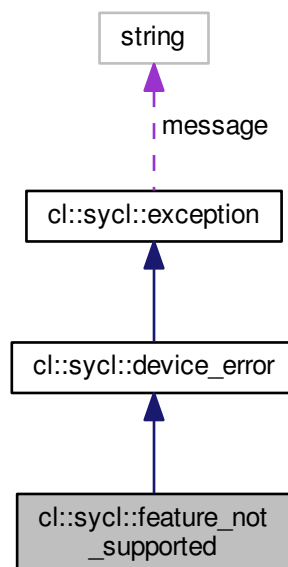
Exception thrown when an optional feature or extension is used in a kernel but its not available on the device the SYCL kernel is being enqueued on.

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.

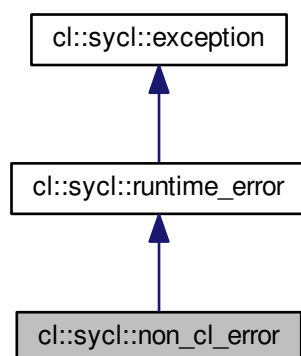
**Todo** Add to the specification

**Todo** Clean implementation

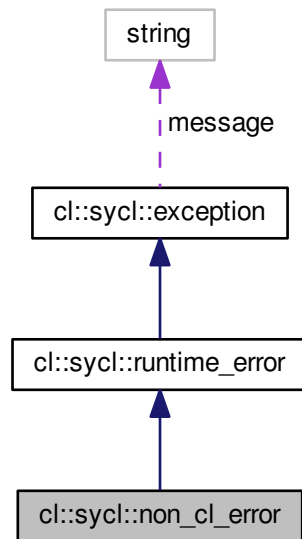
**Todo** Exceptions are named error in C++

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 `async_handler`

```
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 `exception_ptr`

```
using cl::sycl::exception_ptr = typedef std::exception_ptr
#include <include/CL/sycl/exception.hpp>
```

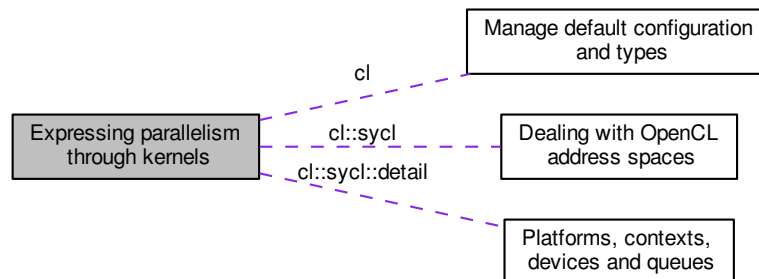
A shared pointer to an exception as in C++ specification.

**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< Dimensions >`  
*A group index used in a `parallel_for_workitem` to specify a work\_group. [More...](#)*
- class `cl::sycl::id< Dimensions >`  
*Define a multi-dimensional index, used for example to locate a work item. [More...](#)*
- class `cl::sycl::item< Dimensions >`  
*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< Dimensions >`  
*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< Dimensions >`  
*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< Dimensions >`  
*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<int Dimensions = 1, typename ParallelForFuncor , typename Id >  
 void `cl::sycl::detail::parallel_for` (`range`< Dimensions > `r`, ParallelForFuncor `f`, Id)  
*Implementation of a data parallel computation with parallelism specified at launch time by a `range`<>.*
- template<int Dimensions = 1, typename ParallelForFuncor >  
 void `cl::sycl::detail::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<int Dimensions = 1, typename ParallelForFuncor >  
 void `cl::sycl::detail::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<int Dimensions = 1, typename ParallelForFuncor >  
 void `cl::sycl::detail::parallel_for_global_offset` (`range`< Dimensions > `global_size`, `id`< Dimensions > `offset`, ParallelForFuncor `f`)  
*Implementation of `parallel_for` with a `range`<> and an offset.*
- template<int Dimensions = 1, typename ParallelForFuncor >  
 void `cl::sycl::detail::parallel_for` (`nd_range`< Dimensions > `r`, ParallelForFuncor `f`)  
*Implement a variation of `parallel_for` to take into account a `nd_range`<>*
- template<int Dimensions = 1, typename ParallelForFuncor >  
 void `cl::sycl::detail::parallel_for_workgroup` (`nd_range`< Dimensions > `r`, ParallelForFuncor `f`)  
*Implement the loop on the work-groups.*
- template<int Dimensions = 1, typename ParallelForFuncor >  
 void `cl::sycl::detail::parallel_for_workitem` (const `group`< Dimensions > &`g`, ParallelForFuncor `f`)  
*Implement the loop on the work-items inside a work-group.*
- template<int Dimensions = 1, typename ParallelForFuncor >  
 void `cl::sycl::parallel_for_work_item` (const `group`< Dimensions > &`g`, ParallelForFuncor `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

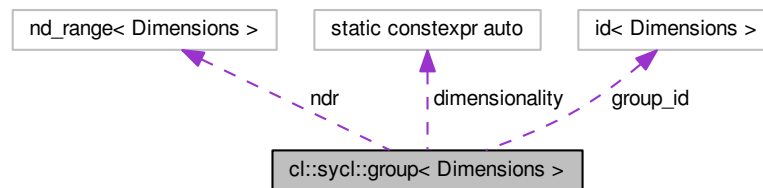
#### 8.9.2.1 struct `cl::sycl::group`

```
template<int Dimensions>
struct cl::sycl::group< Dimensions >
```

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< Dimensions >`:



#### Public Member Functions

- `group` (const `nd_range< Dimensions >` &`ndr`)  
Create a group from an `nd_range<>` with a 0 `id<>`
- `group` (const `id< Dimensions >` &`i`, const `nd_range< Dimensions >` &`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< Dimensions >` `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< Dimensions >` `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< Dimensions >` `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< Dimensions >` `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< Dimensions >` `get_offset` () const  
Get the offset of the `NDRange`.
- `size_t` `get_offset` (int dimension) const  
Get the offset of the `NDRange`.

- `nd_range< Dimensions > 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` = Dimensions

#### Private Attributes

- `id< Dimensions > group_id`  
*The coordinate of the group item.*
- `nd_range< Dimensions > 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 `group()` [1/3]

```
template<int Dimensions>
cl::sycl::group< Dimensions >::group (
    const nd_range< Dimensions > & 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 `group()` [2/3]

```
template<int Dimensions>
cl::sycl::group< Dimensions >::group (
    const id< Dimensions > & i,
    const nd_range< Dimensions > & 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 `group()` [3/3]

```
template<int Dimensions>
cl::sycl::group< Dimensions >::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 `get()` [1/2]

```
template<int Dimensions>
id<Dimensions> cl::sycl::group< Dimensions >::get ( ) const [inline]
```

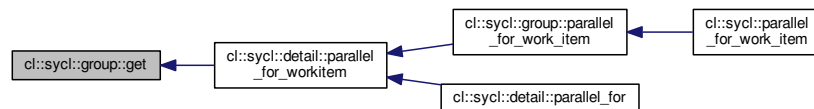
Return an id representing the index of the group within the `nd_range` for every dimension.

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 `get()` [2/2]

```
template<int Dimensions>
size_t cl::sycl::group< Dimensions >::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** `get_global_range()` [1/2]

```
template<int Dimensions>
range<Dimensions> cl::sycl::group< Dimensions >::get_global_range ( ) const [inline]
```

Get the local range for this work\_group.

Definition at line 122 of file [group.hpp](#).

```
00122                                     {
00123     return get_nd_range().get_global();
00124 }
```

**8.9.2.1.2.4** `get_global_range()` [2/2]

```
template<int Dimensions>
size_t cl::sycl::group< Dimensions >::get_global_range (
    int dimension ) const [inline]
```

Return element dimension from the constituent global range.

Definition at line 128 of file [group.hpp](#).

```
00128                                     {
00129     return get_global_range()[dimension];
00130 }
```

**8.9.2.1.2.5** `get_group_range()` [1/2]

```
template<int Dimensions>
range<Dimensions> cl::sycl::group< Dimensions >::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 `get_group_range()` [2/2]

```
template<int Dimensions>
size_t cl::sycl::group< Dimensions >::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 `get_linear()`

```
template<int Dimensions>
size_t cl::sycl::group< Dimensions >::get_linear ( ) const [inline]
```

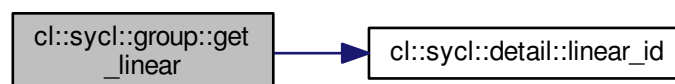
Get a linearized version of the group ID.

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

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

```
00172                                     {
00173     return detail::linear_id(get_group_range(), get());
00174 }
```

Here is the call graph for this function:



8.9.2.1.2.8 `get_local_range()` [1/2]

```
template<int Dimensions>
range<Dimensions> cl::sycl::group< Dimensions >::get_local_range ( ) const [inline]
```

Get the local range for this work\_group.

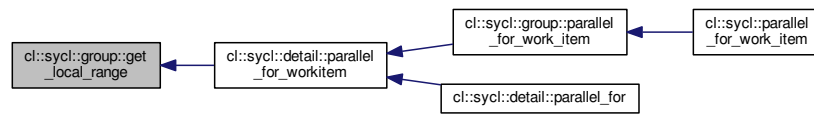
**Todo** Add to the specification

Definition at line 137 of file `group.hpp`.

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

```
00137 {
00138     return get_nd_range().get_local();
00139 }
```

Here is the caller graph for this function:

8.9.2.1.2.9 `get_local_range()` [2/2]

```
template<int Dimensions>
size_t cl::sycl::group< Dimensions >::get_local_range (
    int dimension ) const [inline]
```

Return element dimension from the constituent local range.

**Todo** Add to the specification

Definition at line 146 of file `group.hpp`.

```
00146 {
00147     return get_local_range()[dimension];
00148 }
```

8.9.2.1.2.10 `get_nd_range()`

```
template<int Dimensions>
nd_range<Dimensions> cl::sycl::group< Dimensions >::get_nd_range ( ) const [inline]
```

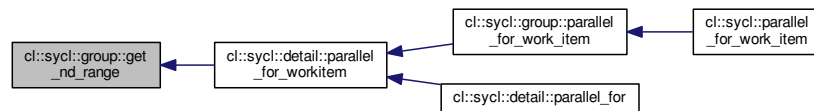
**Todo** Also provide this access to the current `nd_range`

Definition at line 166 of file `group.hpp`.

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

```
00166 { return ndr; }
```

Here is the caller graph for this function:

8.9.2.1.2.11 `get_offset()` [1/2]

```
template<int Dimensions>
id<Dimensions> cl::sycl::group< Dimensions >::get_offset ( ) const [inline]
```

Get the offset of the NDRange.

**Todo** Add to the specification

Definition at line 155 of file `group.hpp`.

```
00155 { return get_nd_range().get_offset(); }
```

#### 8.9.2.1.2.12 `get_offset()` [2/2]

```
template<int Dimensions>
size_t cl::sycl::group< Dimensions >::get_offset (
    int dimension ) const [inline]
```

Get the offset of the NDRange.

**Todo** Add to the specification

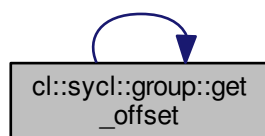
Definition at line 162 of file [group.hpp](#).

References [cl::sycl::group< Dimensions >::get\\_offset\(\)](#).

Referenced by [cl::sycl::group< Dimensions >::get\\_offset\(\)](#).

```
00162 { 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 operator[]()

```
template<int Dimensions>
auto& cl::sycl::group< Dimensions >::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 parallel\_for\_work\_item() [1/2]

```
template<int Dimensions>
void cl::sycl::group< Dimensions >::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

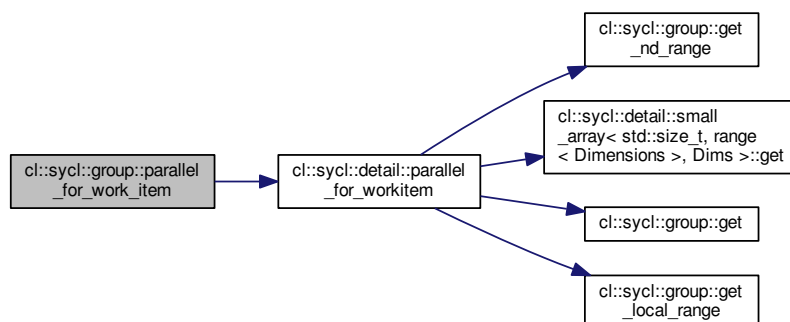
Definition at line 181 of file `group.hpp`.

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

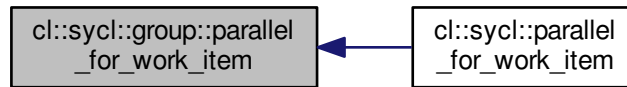
Referenced by `cl::sycl::parallel_for_work_item()`.

```
00182 {
00183     detail::parallel_for_workitem(*this, f);
00184 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.9.2.1.2.15 parallel\_for\_work\_item() [2/2]

```

template<int Dimensions>
void cl::sycl::group< Dimensions >::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

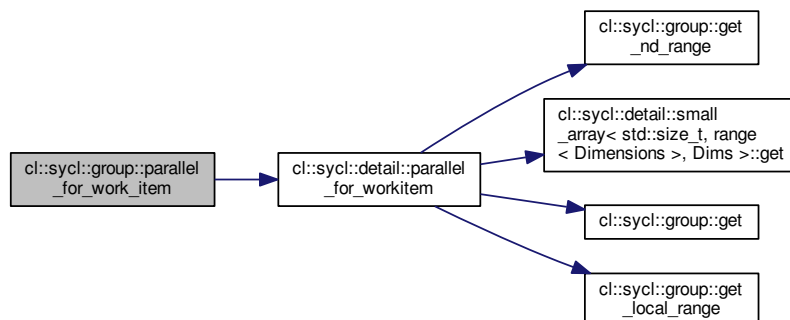
Definition at line 191 of file [group.hpp](#).

References [cl::sycl::detail::parallel\\_for\\_workitem\(\)](#).

```

00192     {
00193         auto item_adapter = [=] (nd_item<dimensionality> ndi) {
00194             item<dimensionality> i = ndi.get_item();
00195             f(i);
00196         };
00197         detail::parallel_for_workitem(*this, item_adapter);
00198     }
  
```

Here is the call graph for this function:



## 8.9.2.1.3 Member Data Documentation

## 8.9.2.1.3.1 dimensionality

```
template<int Dimensions>
constexpr auto cl::sycl::group< Dimensions >::dimensionality = Dimensions [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 group\_id

```
template<int Dimensions>
id<Dimensions> cl::sycl::group< Dimensions >::group_id [private]
```

The coordinate of the group item.

Definition at line 49 of file [group.hpp](#).

## 8.9.2.1.3.3 ndr

```
template<int Dimensions>
nd_range<Dimensions> cl::sycl::group< Dimensions >::ndr [private]
```

Keep a reference on the [nd\\_range](#) to serve potential query on it.

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

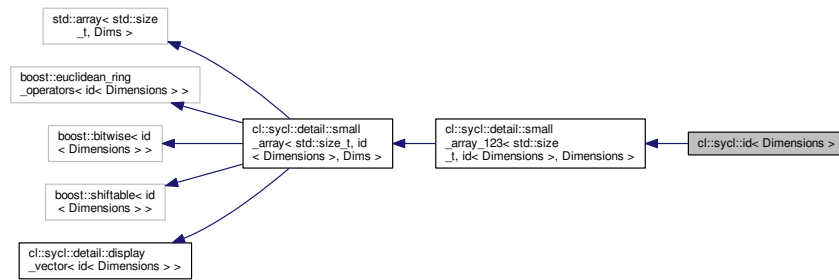
## 8.9.2.2 class cl::sycl::id

```
template<int Dimensions = 1>
class cl::sycl::id< Dimensions >
```

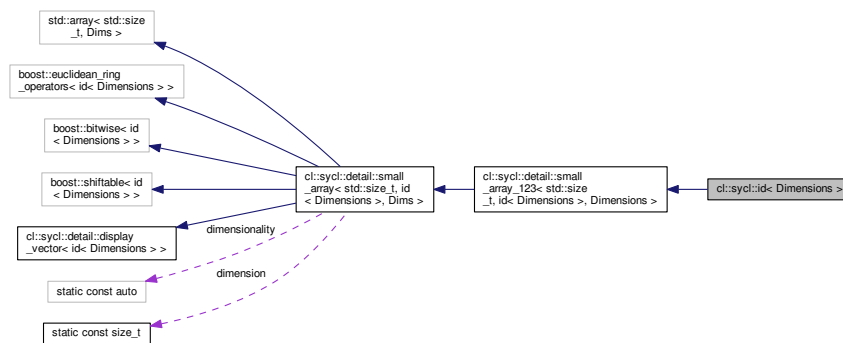
Define a multi-dimensional index, used for example to locate a work item.

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

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



Collaboration diagram for `cl::sycl::id< Dimensions >`:



## Public Member Functions

- `id` (const [range](#)< Dimensions > &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 id()

```

template<int Dimensions = 1>
cl::sycl::id< Dimensions >::id (
    const range< Dimensions > & 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 45 of file [id.hpp](#).

Referenced by `cl::sycl::id< dimensionality >::id()`.



```

00049      : detail::small_array_123<std::size_t, id<Dimensions>, Dimensions>
00050      { range_size }

```

Here is the caller graph for this function:



### 8.9.2.3 class `cl::sycl::item`

```

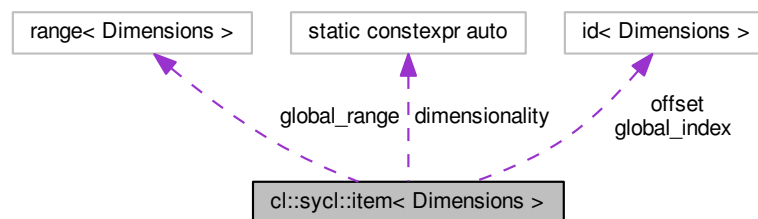
template<int Dimensions = 1>
class cl::sycl::item< Dimensions >

```

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< Dimensions >`:



#### Public Member Functions

- `item` (`range< Dimensions > global_size`, `id< Dimensions > global_index`, `id< Dimensions > 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< Dimensions > 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`< Dimensions > `get_range` () const  
*Returns a range<> representing the dimensions of the range of possible values of the item.*
- `id`< Dimensions > `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< Dimensions > 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` = Dimensions

#### Private Attributes

- `range`< Dimensions > `global_range`
- `id`< Dimensions > `global_index`
- `id`< Dimensions > `offset`

### 8.9.2.3.1 Constructor & Destructor Documentation

#### 8.9.2.3.1.1 `item()` [1/2]

```
template<int Dimensions = 1>
cl::sycl::item< Dimensions >::item (
    range< Dimensions > global_size,
    id< Dimensions > global_index,
    id< Dimensions > 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< Dimensions >::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 `item()` [2/2]

```
template<int Dimensions = 1>
cl::sycl::item< Dimensions >::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< Dimensions >::item()`.

Here is the caller graph for this function:



## 8.9.2.3.2 Member Function Documentation

8.9.2.3.2.1 `display()`

```
template<int Dimensions = 1>
void cl::sycl::item< Dimensions >::display ( ) const [inline]
```

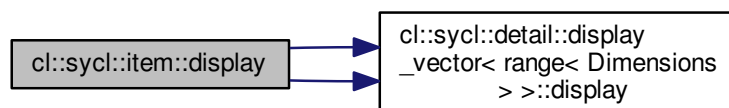
Display the value for debugging and validation purpose.

Definition at line 117 of file `item.hpp`.

References `cl::sycl::detail::display_vector< range< Dimensions > >::display()`, and `cl::sycl::detail::display_vector< id< Dimensions > >::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** `get()` [1/2]

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::item< Dimensions >::get ( ) const [inline]
```

Return the constituent local or global id<> representing the work-item's position in the iteration space.

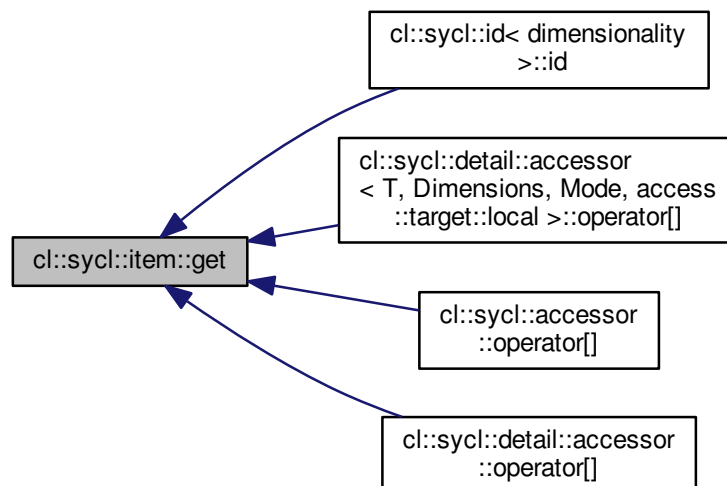
Definition at line 69 of file `item.hpp`.

References `cl::sycl::item< Dimensions >::global_index`.

Referenced by `cl::sycl::id< dimensionality >::id()`, `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[]()`, `cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[]()`, and `cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[]()`.

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

Here is the caller graph for this function:

**8.9.2.3.2.3** `get()` [2/2]

```
template<int Dimensions = 1>
size_t cl::sycl::item< Dimensions >::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 `get_linear_id()`

```
template<int Dimensions = 1>
size_t cl::sycl::item< Dimensions >::get_linear_id ( ) const [inline]
```

Return the linearized ID in the item's range.

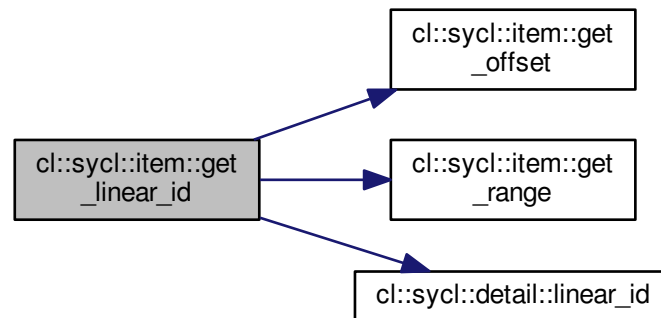
Computed as the flattened ID after the offset is subtracted.

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

References [cl::sycl::item< Dimensions >::get\\_offset\(\)](#), [cl::sycl::item< Dimensions >::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 `get_offset()`

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::item< Dimensions >::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.

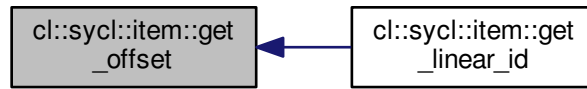
Definition at line 97 of file [item.hpp](#).

References [cl::sycl::item< Dimensions >::offset](#).

Referenced by [cl::sycl::item< Dimensions >::get\\_linear\\_id\(\)](#).

```
00097 { return offset; }
```

Here is the caller graph for this function:



#### 8.9.2.3.2.6 get\_range()

```
template<int Dimensions = 1>
range<Dimensions> cl::sycl::item< Dimensions >::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< Dimensions >::global\\_range](#).

Referenced by [cl::sycl::item< Dimensions >::get\\_linear\\_id\(\)](#).

```
00087 { return global_range; }
```

Here is the caller graph for this function:



## 8.9.2.3.2.7 operator[]()

```
template<int Dimensions = 1>
auto& cl::sycl::item< Dimensions >::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 set()

```
template<int Dimensions = 1>
void cl::sycl::item< Dimensions >::set (
    id< Dimensions > 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 dimensionality

```
template<int Dimensions = 1>
constexpr auto cl::sycl::item< Dimensions >::dimensionality = Dimensions [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 global\_index

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::item< Dimensions >::global_index [private]
```

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

Referenced by [cl::sycl::item< Dimensions >::get\(\)](#).

#### 8.9.2.3.3.3 global\_range

```
template<int Dimensions = 1>
range<Dimensions> cl::sycl::item< Dimensions >::global_range [private]
```

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

Referenced by [cl::sycl::item< Dimensions >::get\\_range\(\)](#).

#### 8.9.2.3.3.4 offset

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::item< Dimensions >::offset [private]
```

Definition at line 41 of file [item.hpp](#).

Referenced by [cl::sycl::item< Dimensions >::get\\_offset\(\)](#).

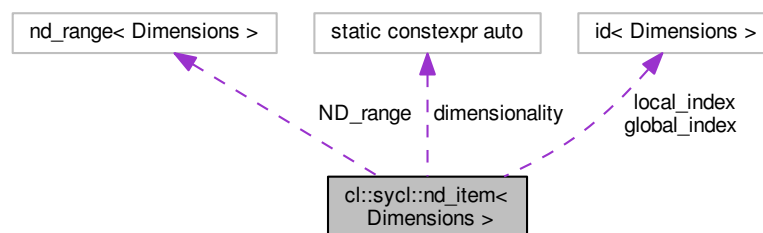
### 8.9.2.4 struct cl::sycl::nd\_item

```
template<int Dimensions = 1>
struct cl::sycl::nd_item< Dimensions >
```

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< Dimensions >`:





## Public Member Functions

- `nd_item (nd_range< Dimensions > ndr)`  
*Create an empty `nd_item<>` from an `nd_range<>`*
- `nd_item (id< Dimensions > global_index, nd_range< Dimensions > ndr)`  
*Create a full `nd_item`.*
- `nd_item ()=default`  
*To be able to copy and assign `nd_item`, use default constructors too.*
- `id< Dimensions > 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< Dimensions > 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< Dimensions > 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< Dimensions > 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< Dimensions > get_global_range () const`  
*Return a `range<>` representing the dimensions of the `nd_range<>`*
- `range< Dimensions > get_local_range () const`  
*Return a `range<>` representing the dimensions of the current work-group.*
- `id< Dimensions > 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< Dimensions > get_nd_range () const`  
*Return the `nd_range<>` of the current execution.*
- `item< Dimensions > 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< Dimensions > Index)`
- `void set_global (id< Dimensions > Index)`

## Static Public Attributes

- static constexpr auto `dimensionality` = Dimensions

## Private Attributes

- [id](#)< Dimensions > [global\\_index](#)
- [id](#)< Dimensions > [local\\_index](#)
- [nd\\_range](#)< Dimensions > [ND\\_range](#)

### 8.9.2.4.1 Constructor & Destructor Documentation

#### 8.9.2.4.1.1 [nd\\_item\(\)](#) [1/3]

```
template<int Dimensions = 1>
cl::sycl::nd_item< Dimensions >::nd_item (
    nd_range< Dimensions > 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 [nd\\_item\(\)](#) [2/3]

```
template<int Dimensions = 1>
cl::sycl::nd_item< Dimensions >::nd_item (
    id< Dimensions > global_index,
    nd_range< Dimensions > ndr ) [inline]
```

Create a full [nd\\_item](#).

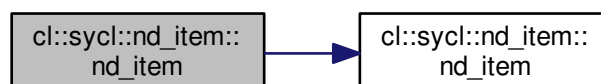
**Todo** This is for validation purpose. Hide this to the programmer somehow

Definition at line 62 of file [nd\\_item.hpp](#).

References [cl::sycl::nd\\_item< Dimensions >::nd\\_item\(\)](#).

```
00063 :
00064     global_index { global_index },
00065     // Compute the local index using the offset and the group size
00066     local_index
00067     { (global_index - ndr.get_offset())%id<Dimensions> { ndr.get_local() } },
00068     ND_range { ndr }
00069 {}
```

Here is the call graph for this function:



8.9.2.4.1.3 `nd_item()` [3/3]

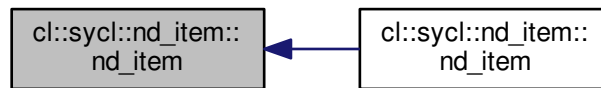
```
template<int Dimensions = 1>
cl::sycl::nd_item< Dimensions >::nd_item ( ) [default]
```

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< Dimensions >::nd_item()`.

Here is the caller graph for this function:



## 8.9.2.4.2 Member Function Documentation

8.9.2.4.2.1 `barrier()`

```
template<int Dimensions = 1>
void cl::sycl::nd_item< Dimensions >::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.

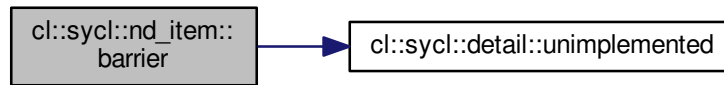
In addition, the barrier performs a fence operation ensuring that all memory accesses in the specified address space issued before the barrier complete before those issued after the barrier

Definition at line 199 of file `nd_item.hpp`.

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

```
00200                                     {
00201 #if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
00202     /* Use OpenMP barrier in the implementation with 1 OpenMP thread per
00203        work-item of the work-group */
00204 #pragma omp barrier
00205 #else
00206     // \todo To be implemented efficiently otherwise
00207     detail::unimplemented();
00208 #endif
00209 }
```

Here is the call graph for this function:



#### 8.9.2.4.2.2 get\_global() [1/2]

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_item< Dimensions >::get_global ( ) const [inline]
```

Return the constituent global id representing the work-item's position in the global iteration space.

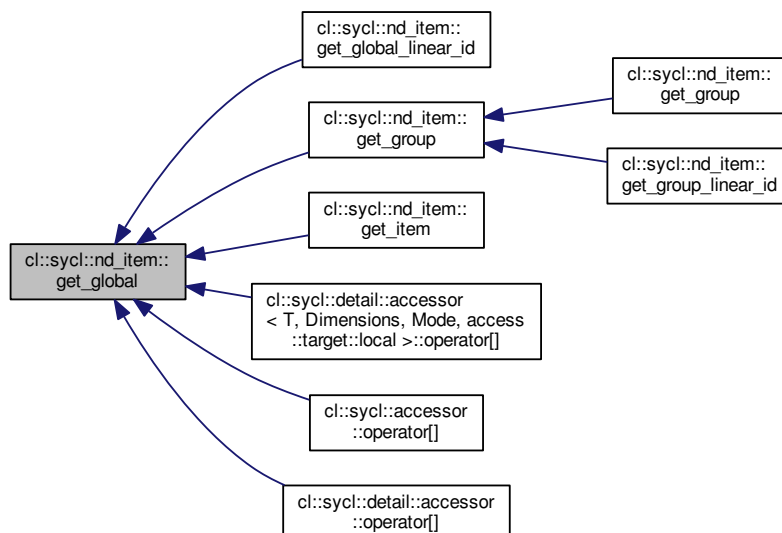
Definition at line 82 of file [nd\\_item.hpp](#).

References [cl::sycl::nd\\_item< Dimensions >::global\\_index](#).

Referenced by [cl::sycl::nd\\_item< Dimensions >::get\\_global\\_linear\\_id\(\)](#), [cl::sycl::nd\\_item< Dimensions >::get\\_group\(\)](#), [cl::sycl::nd\\_item< Dimensions >::get\\_item\(\)](#), [cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator\[\]\(\)](#), [cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator\[\]\(\)](#), and [cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator\[\]\(\)](#).

```
00082 { return global_index; }
```

Here is the caller graph for this function:



#### 8.9.2.4.2.3 `get_global()` [2/2]

```
template<int Dimensions = 1>
size_t cl::sycl::nd_item< Dimensions >::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 89 of file `nd_item.hpp`.

References `cl::sycl::nd_item< Dimensions >::get_global()`.

Referenced by `cl::sycl::nd_item< Dimensions >::get_global()`.

```
00089 { 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 `get_global_linear_id()`

```
template<int Dimensions = 1>
size_t cl::sycl::nd_item< Dimensions >::get_global_linear_id ( ) const [inline]
```

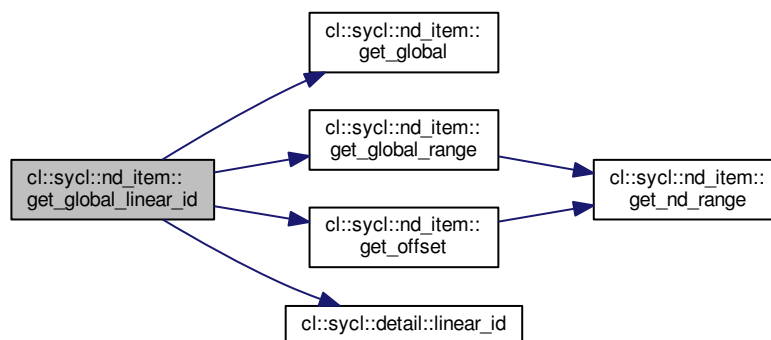
Return the flattened id of the current work-item after subtracting the offset.

Definition at line 95 of file `nd_item.hpp`.

References `cl::sycl::nd_item< Dimensions >::get_global()`, `cl::sycl::nd_item< Dimensions >::get_global_range()`, `cl::sycl::nd_item< Dimensions >::get_offset()`, and `cl::sycl::detail::linear_id()`.

```
00095     {
00096     return detail::linear_id(get_global_range(),
00097                             get_global(), get_offset());
00097     }
```

Here is the call graph for this function:



#### 8.9.2.4.2.5 `get_global_range()`

```
template<int Dimensions = 1>
range<Dimensions> cl::sycl::nd_item< Dimensions >::get_global_range ( ) const [inline]
```

Return a `range<>` representing the dimensions of the `nd_range<>`

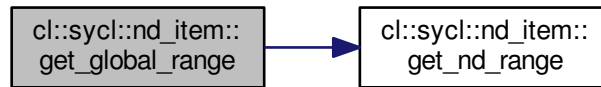
Definition at line 158 of file `nd_item.hpp`.

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

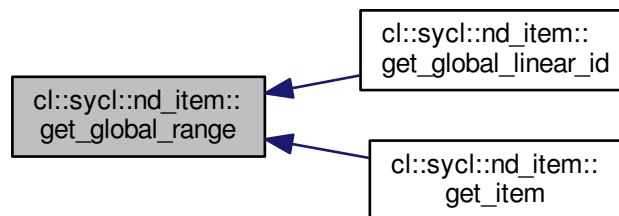
Referenced by `cl::sycl::nd_item< Dimensions >::get_global_linear_id()`, and `cl::sycl::nd_item< Dimensions >::get_item()`.

```
00158     {
00159     return get_nd_range().get_global();
00160     }
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.9.2.4.2.6 `get_group()` [1/2]

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_item< Dimensions >::get_group ( ) const [inline]
```

Return the constituent group group representing the work-group's position within the overall `nd_range`.

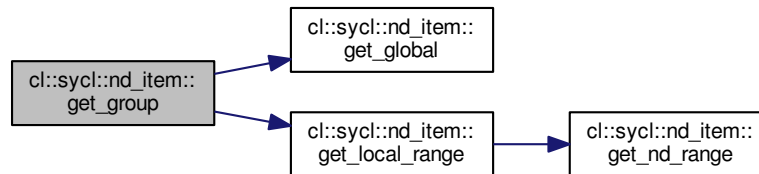
Definition at line 124 of file `nd_item.hpp`.

References `cl::sycl::nd_item< Dimensions >::get_global()`, and `cl::sycl::nd_item< Dimensions >::get_local_range()`.

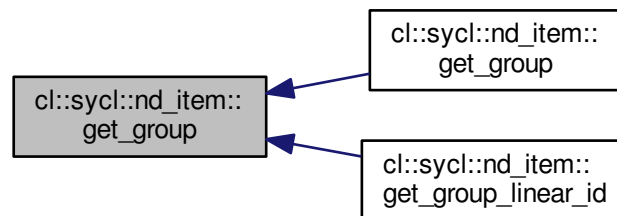
Referenced by `cl::sycl::nd_item< Dimensions >::get_group()`, and `cl::sycl::nd_item< Dimensions >::get_group_linear_id()`.

```
00124         {
00125     /* Convert get_local_range() to an id<> to remove ambiguity into using
00126         implicit conversion either from range<> to id<> or the opposite */
00127     return get_global()/id<Dimensions> { get_local_range() };
00128 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.9.2.4.2.7 `get_group()` [2/2]

```

template<int Dimensions = 1>
size_t cl::sycl::nd_item< Dimensions >::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 135 of file `nd_item.hpp`.

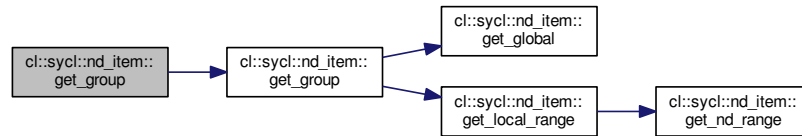
References `cl::sycl::nd_item< Dimensions >::get_group()`.

```

00135                                     {
00136     return get_group()[dimension];
00137 }
  
```



Here is the call graph for this function:



#### 8.9.2.4.2.8 `get_group_linear_id()`

```

template<int Dimensions = 1>
size_t cl::sycl::nd_item< Dimensions >::get_group_linear_id ( ) const [inline]

```

Return the flattened id of the current work-group.

Definition at line 141 of file `nd_item.hpp`.

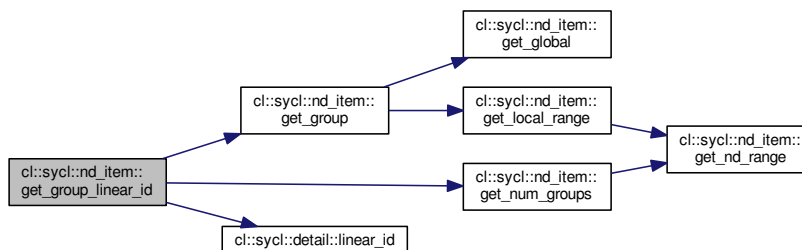
References `cl::sycl::nd_item< Dimensions >::get_group()`, `cl::sycl::nd_item< Dimensions >::get_num_groups()`, and `cl::sycl::detail::linear_id()`.

```

00141         {
00142     return detail::linear_id(get_num_groups(),
00143         get_group());
00143     }

```

Here is the call graph for this function:



#### 8.9.2.4.2.9 `get_item()`

```
template<int Dimensions = 1>
item<Dimensions> cl::sycl::nd_item< Dimensions >::get_item ( ) const [inline]
```

Allows projection down to an item.

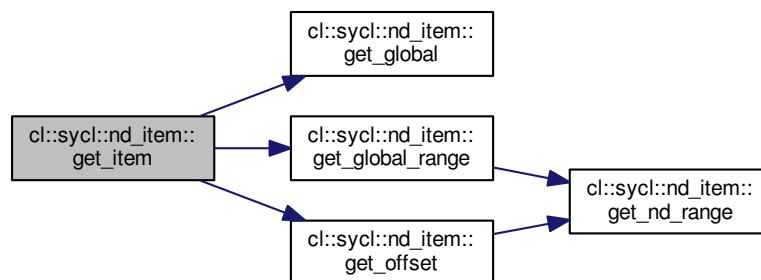
**Todo** Add to the specification

Definition at line 184 of file `nd_item.hpp`.

References `cl::sycl::nd_item< Dimensions >::get_global()`, `cl::sycl::nd_item< Dimensions >::get_global_range()`, and `cl::sycl::nd_item< Dimensions >::get_offset()`.

```
00184         {
00185     return { get_global_range(), get_global(),
00186           get_offset() };
00186     }
```

Here is the call graph for this function:



#### 8.9.2.4.2.10 `get_local()` [1/2]

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_item< Dimensions >::get_local ( ) const [inline]
```

Return the constituent local id representing the work-item's position within the current work-group.

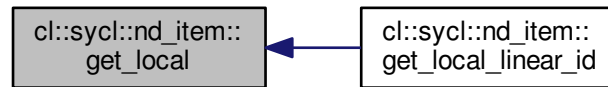
Definition at line 103 of file `nd_item.hpp`.

References `cl::sycl::nd_item< Dimensions >::local_index`.

Referenced by `cl::sycl::nd_item< Dimensions >::get_local_linear_id()`.

```
00103 { return local_index; }
```

Here is the caller graph for this function:



#### 8.9.2.4.2.11 get\_local() [2/2]

```
template<int Dimensions = 1>
size_t cl::sycl::nd_item< Dimensions >::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.

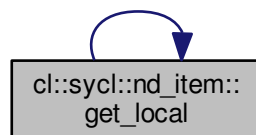
Definition at line 110 of file [nd\\_item.hpp](#).

References [cl::sycl::nd\\_item< Dimensions >::get\\_local\(\)](#).

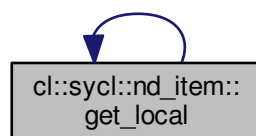
Referenced by [cl::sycl::nd\\_item< Dimensions >::get\\_local\(\)](#).

```
00110 { 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 `get_local_linear_id()`

```
template<int Dimensions = 1>
size_t cl::sycl::nd_item< Dimensions >::get_local_linear_id ( ) const [inline]
```

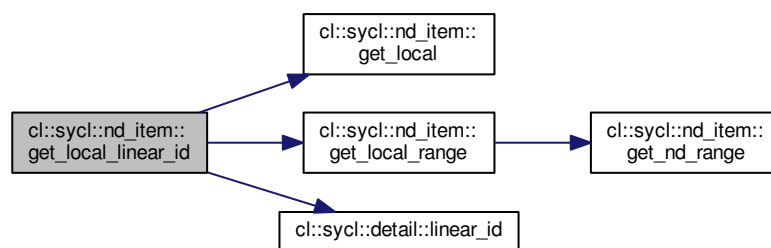
Return the flattened id of the current work-item within the current work-group.

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

References `cl::sycl::nd_item< Dimensions >::get_local()`, `cl::sycl::nd_item< Dimensions >::get_local_range()`, and `cl::sycl::detail::linear_id()`.

```
00116         {
00117     return detail::linear_id(get_local_range(),
00118                             get_local());
00118 }
```

Here is the call graph for this function:



#### 8.9.2.4.2.13 `get_local_range()`

```
template<int Dimensions = 1>
range<Dimensions> cl::sycl::nd_item< Dimensions >::get_local_range ( ) const [inline]
```

Return a `range<>` representing the dimensions of the current work-group.

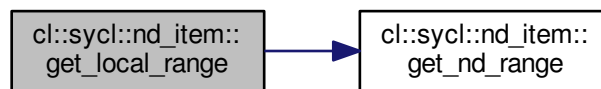
Definition at line 164 of file `nd_item.hpp`.

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

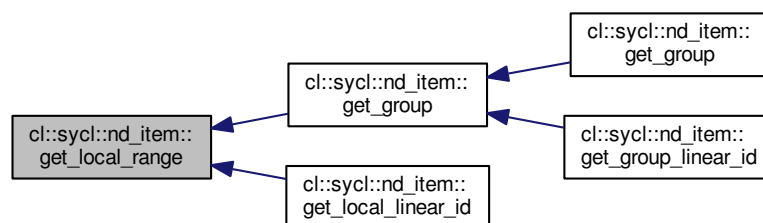
Referenced by `cl::sycl::nd_item< Dimensions >::get_group()`, and `cl::sycl::nd_item< Dimensions >::get_local_linear_id()`.

```
00164     {
00165     return get_nd_range().get_local();
00166 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.9.2.4.2.14 get\_nd\_range()

```
template<int Dimensions = 1>
nd_range<Dimensions> cl::sycl::nd_item< Dimensions >::get_nd_range ( ) const [inline]
```

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

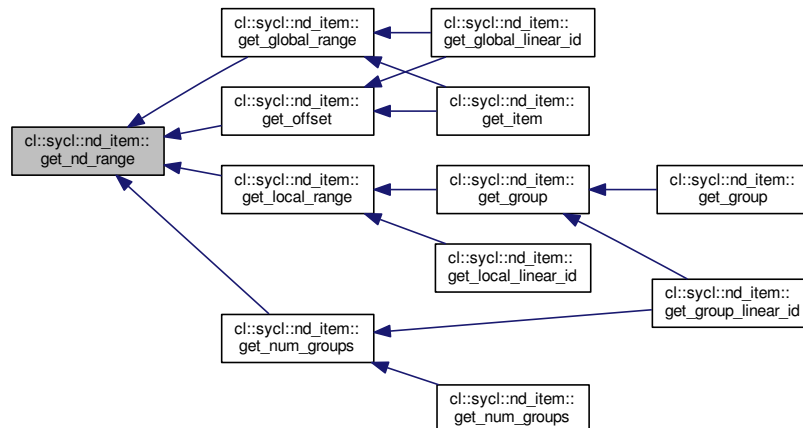
Definition at line 177 of file [nd\\_item.hpp](#).

References [cl::sycl::nd\\_item< Dimensions >::ND\\_range](#).

Referenced by [cl::sycl::nd\\_item< Dimensions >::get\\_global\\_range\(\)](#), [cl::sycl::nd\\_item< Dimensions >::get\\_local\\_range\(\)](#), [cl::sycl::nd\\_item< Dimensions >::get\\_num\\_groups\(\)](#), and [cl::sycl::nd\\_item< Dimensions >::get\\_offset\(\)](#).

```
00177 { return ND_range; }
```

Here is the caller graph for this function:



#### 8.9.2.4.2.15 `get_num_groups()` [1/2]

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_item< Dimensions >::get_num_groups ( ) const [inline]
```

Return the number of groups in the `nd_range`.

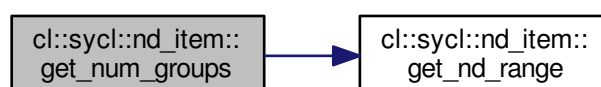
Definition at line 147 of file `nd_item.hpp`.

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

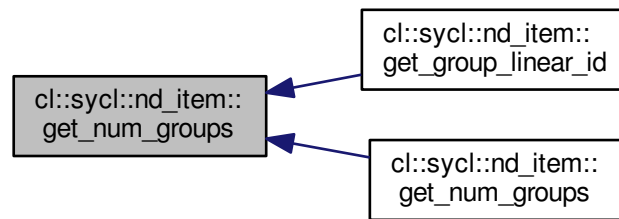
Referenced by `cl::sycl::nd_item< Dimensions >::get_group_linear_id()`, and `cl::sycl::nd_item< Dimensions >::get_num_groups()`.

```
00147 {
00148     return get_nd_range().get_group();
00149 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.9.2.4.2.16 `get_num_groups()` [2/2]

```
template<int Dimensions = 1>
size_t cl::sycl::nd_item< Dimensions >::get_num_groups (
    int dimension ) const [inline]
```

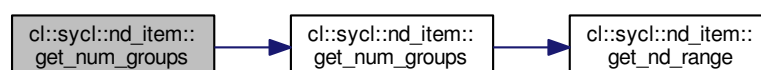
Return the number of groups for dimension in the `nd_range`.

Definition at line 152 of file `nd_item.hpp`.

References `cl::sycl::nd_item< Dimensions >::get_num_groups()`.

```
00152                                     {
00153     return get_num_groups() [dimension];
00154 }
```

Here is the call graph for this function:



#### 8.9.2.4.2.17 get\_offset()

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_item< Dimensions >::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.

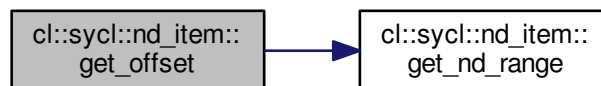
Definition at line 173 of file `nd_item.hpp`.

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

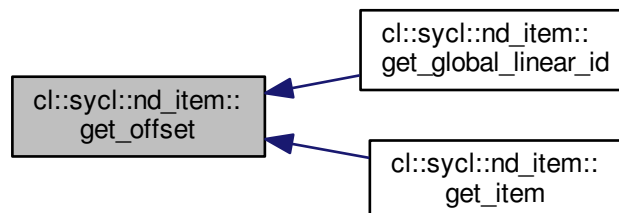
Referenced by `cl::sycl::nd_item< Dimensions >::get_global_linear_id()`, and `cl::sycl::nd_item< Dimensions >::get_item()`.

```
00173 { 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 set\_global()

```
template<int Dimensions = 1>
void cl::sycl::nd_item< Dimensions >::set_global (
    id< Dimensions > Index ) [inline]
```

Definition at line 217 of file `nd_item.hpp`.

```
00217 { global_index = Index; }
```



**8.9.2.4.2.19 set\_local()**

```
template<int Dimensions = 1>
void cl::sycl::nd_item< Dimensions >::set_local (
    id< Dimensions > Index ) [inline]
```

Definition at line 213 of file [nd\\_item.hpp](#).

```
00213 { local_index = Index; }
```

**8.9.2.4.3 Member Data Documentation****8.9.2.4.3.1 dimensionality**

```
template<int Dimensions = 1>
constexpr auto cl::sycl::nd_item< Dimensions >::dimensionality = Dimensions [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 global\_index**

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_item< Dimensions >::global_index [private]
```

Definition at line 40 of file [nd\\_item.hpp](#).

Referenced by [cl::sycl::nd\\_item< Dimensions >::get\\_global\(\)](#).

**8.9.2.4.3.3 local\_index**

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_item< Dimensions >::local_index [private]
```

Definition at line 43 of file [nd\\_item.hpp](#).

Referenced by [cl::sycl::nd\\_item< Dimensions >::get\\_local\(\)](#).

**8.9.2.4.3.4 ND\_range**

```
template<int Dimensions = 1>
nd_range<Dimensions> cl::sycl::nd_item< Dimensions >::ND_range [private]
```

Definition at line 44 of file [nd\\_item.hpp](#).

Referenced by [cl::sycl::nd\\_item< Dimensions >::get\\_nd\\_range\(\)](#).



## Private Attributes

- [range< dimensionality > global\\_range](#)
- [range< dimensionality > local\\_range](#)
- [id< dimensionality > offset](#)

## 8.9.2.5.1 Constructor &amp; Destructor Documentation

## 8.9.2.5.1.1 nd\_range()

```
template<int Dimensions = 1>
cl::sycl::nd_range< Dimensions >::nd_range (
    range< Dimensions > global_size,
    range< Dimensions > local_size,
    id< Dimensions > 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 }
00054     { }
```

## 8.9.2.5.2 Member Function Documentation

## 8.9.2.5.2.1 display()

```
template<int Dimensions = 1>
void cl::sycl::nd_range< Dimensions >::display ( ) const [inline]
```

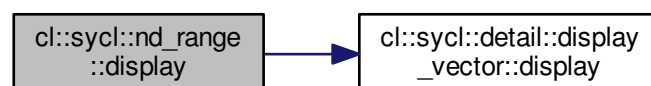
Display the value for debugging and validation purpose.

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 `get_global()`

```
template<int Dimensions = 1>
range<Dimensions> cl::sycl::nd_range< Dimensions >::get_global ( ) const [inline]
```

Get the global iteration space range.

Definition at line 58 of file `nd_range.hpp`.

References `cl::sycl::nd_range< Dimensions >::global_range`.

```
00058 { return global_range; }
```

#### 8.9.2.5.2.3 `get_group()`

```
template<int Dimensions = 1>
auto cl::sycl::nd_range< Dimensions >::get_group ( ) const [inline]
```

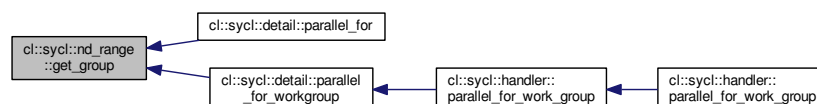
Get the range of work-groups needed to run this ND-range.

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 range 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<Dimensions>{ 1 })/
        local_range;
00072 }
```

Here is the caller graph for this function:



8.9.2.5.2.4 `get_local()`

```
template<int Dimensions = 1>
range<Dimensions> cl::sycl::nd_range< Dimensions >::get_local ( ) const [inline]
```

Get the local part of the iteration space range.

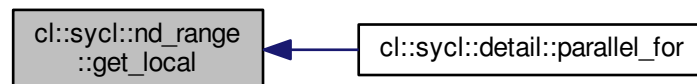
Definition at line 62 of file `nd_range.hpp`.

References `cl::sycl::nd_range< Dimensions >::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 `get_offset()`

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_range< Dimensions >::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< Dimensions >::offset`.

```
00076 { return offset; }
```

## 8.9.2.5.3 Member Data Documentation

#### 8.9.2.5.3.1 dimensionality

```
template<int Dimensions = 1>
constexpr auto cl::sycl::nd_range< Dimensions >::dimensionality = Dimensions [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 global\_range

```
template<int Dimensions = 1>
range<dimensionality> cl::sycl::nd_range< Dimensions >::global_range [private]
```

Definition at line 40 of file [nd\\_range.hpp](#).

Referenced by [cl::sycl::nd\\_range< Dimensions >::get\\_global\(\)](#).

#### 8.9.2.5.3.3 local\_range

```
template<int Dimensions = 1>
range<dimensionality> cl::sycl::nd_range< Dimensions >::local_range [private]
```

Definition at line 41 of file [nd\\_range.hpp](#).

Referenced by [cl::sycl::nd\\_range< Dimensions >::get\\_local\(\)](#).

#### 8.9.2.5.3.4 offset

```
template<int Dimensions = 1>
id<dimensionality> cl::sycl::nd_range< Dimensions >::offset [private]
```

Definition at line 42 of file [nd\\_range.hpp](#).

Referenced by [cl::sycl::nd\\_range< Dimensions >::get\\_offset\(\)](#).

#### 8.9.2.6 struct cl::sycl::detail::parallel\_for\_iterate

```
template<std::size_t level, typename Range, typename ParallelForFunc, typename Id>
struct cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFunc, 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 50 of file [parallelism.hpp](#).

## Public Member Functions

- [parallel\\_for\\_iterate](#) (Range r, ParallelForFuncor &f, Id &index)

## 8.9.2.6.1 Constructor &amp; Destructor Documentation

## 8.9.2.6.1.1 parallel\_for\_iterate()

```
template<std::size_t level, typename Range , typename ParallelForFuncor , typename Id >
cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFuncor, Id >::parallel_for←
_iterate (
    Range r,
    ParallelForFuncor & f,
    Id & index ) [inline]
```

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

```
00051                                     {
00052     for (boost::multi_array_types::index _sycl_index = 0,
00053         _sycl_end = r[Range::dimensionality - level];
00054         _sycl_index < _sycl_end;
00055         _sycl_index++) {
00056         // Set the current value of the index for this dimension
00057         index[Range::dimensionality - level] = _sycl_index;
00058         // Iterate further on lower dimensions
00059         parallel_for_iterate<level - 1,
00060                               Range,
00061                               ParallelForFuncor,
00062                               Id> { r, f, index };
00063     }
00064 }
```

## 8.9.2.7 struct cl::sycl::detail::parallel\_OpenMP\_for\_iterate

```
template<std::size_t level, typename Range, typename ParallelForFuncor, typename Id>
struct cl::sycl::detail::parallel_OpenMP_for_iterate< level, Range, ParallelForFuncor, 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 77 of file [parallelism.hpp](#).

## Public Member Functions

- [parallel\\_OpenMP\\_for\\_iterate](#) (Range r, ParallelForFuncor &f)

## 8.9.2.7.1 Constructor &amp; Destructor Documentation

### 8.9.2.7.1.1 parallel\_OpenMP\_for\_iterate()

```
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 78 of file [parallelism.hpp](#).

```
00078                                     {
00079     // Create the OpenMP threads before the for-loop to avoid creating an
00080     // index in each iteration
00081     #pragma omp parallel
00082     {
00083         // Allocate an OpenMP thread-local index
00084         Id index;
00085         // Make a simple loop end condition for OpenMP
00086         boost::multi_array_types::index _sycl_end =
00087             r[Range::dimensionality - level];
00088         /* Distribute the iterations on the OpenMP threads. Some OpenMP
00089         "collapse" could be useful for small iteration space, but it
00090         would need some template specialization to have real contiguous
00091         loop nests */
00092         #pragma omp for
00093         for (boost::multi_array_types::index _sycl_index = 0;
00094             _sycl_index < _sycl_end;
00095             _sycl_index++) {
00096             // Set the current value of the index for this dimension
00097             index[Range::dimensionality - level] = _sycl_index;
00098             // Iterate further on lower dimensions
00099             parallel_for_iterate<level - 1,
00100                                     Range,
00101                                     ParallelForFunctor,
00102                                     Id> { r, f, index };
00103         }
00104     }
00105 }
```

### 8.9.2.8 struct cl::sycl::detail::parallel\_for\_iterate< 0, Range, ParallelForFunctor, Id >

```
template<typename Range, typename ParallelForFunctor, typename Id>
struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >
```

Stop the recursion when level reaches 0 by simply calling the kernel functor with the constructed id.

Definition at line 112 of file [parallelism.hpp](#).

#### Public Member Functions

- [parallel\\_for\\_iterate](#) (Range r, ParallelForFunctor &f, Id &index)

### 8.9.2.8.1 Constructor & Destructor Documentation



8.9.2.8.1.1 `parallel_for_iterate()`

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

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

```
00113                                     {
00114     f(index);
00115 }
```

8.9.2.9 class `cl::sycl::range`

```
template<int Dimensions = 1>
class cl::sycl::range< Dimensions >
```

A SYCL range defines a multi-dimensional index range that can be used to define launch parallel computation extent or buffer sizes.

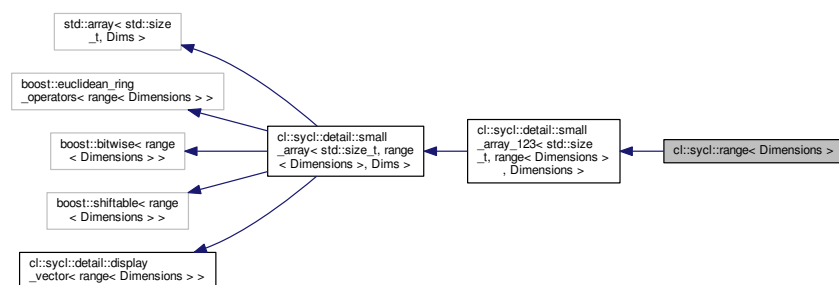
**Todo** use `std::size_t` `Dimensions` instead of `int` `Dimensions` in the specification?

**Todo** add to the specification this default parameter value?

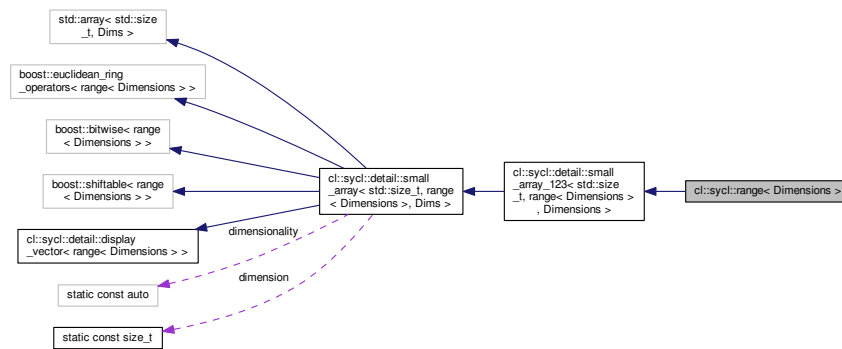
**Todo** add to the specification some way to specify an offset?

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

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



Collaboration diagram for `cl::sycl::range< Dimensions >`:



## Public Member Functions

- `size_t get_count () const`  
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 get\_count()

```
template<int Dimensions = 1>
size_t cl::sycl::range< Dimensions >::get_count ( ) const [inline]
```

Return the number of elements in the range.

**Todo** Give back `size()` its real meaning in the specification

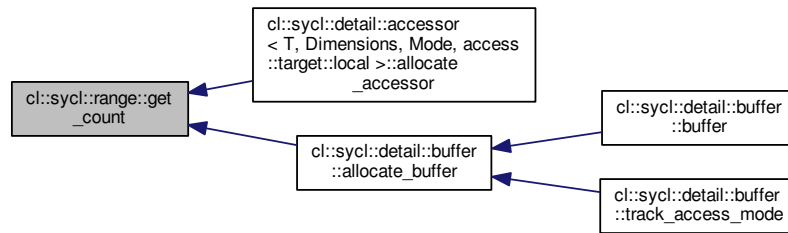
**Todo** add this method to the specification

Definition at line 53 of file [range.hpp](#).

Referenced by `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::allocate_accessor()`, and `cl::sycl::detail::buffer< T, Dimensions >::allocate_buffer()`.

```
00053     {
00054         // Return the product of the sizes in each dimension
00055         return std::accumulate(this->cbegin(),
00056                                this->cend(),
00057                                1,
00058                                std::multiplies<size_t> {});
00059     }
```

Here is the caller graph for this function:



### 8.9.3 Function Documentation

#### 8.9.3.1 make\_id() [1/4]

```
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 71 of file [id.hpp](#).

```
00071 { return i; }
```

#### 8.9.3.2 make\_id() [2/4]

```
auto cl::sycl::make_id (
    id< 2 > i ) [inline]
```

```
#include <include/CL/sycl/id.hpp>
```

Definition at line 72 of file [id.hpp](#).

```
00072 { return i; }
```

**8.9.3.3 make\_id()** [3/4]

```
auto cl::sycl::make_id (
    id< 3 > i ) [inline]

#include <include/CL/sycl/id.hpp>
```

Definition at line 73 of file [id.hpp](#).

```
00073 { return i; }
```

**8.9.3.4 make\_id()** [4/4]

```
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 79 of file [id.hpp](#).

```
00079                                     {
00080     // Call constructor directly to allow narrowing
00081     return id<sizeof...(Args)>(Args...);
00082 }
```

**8.9.3.5 make\_range()** [1/4]

```
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 69 of file [range.hpp](#).

```
00069 { return r; }
```

**8.9.3.6** `make_range()` [2/4]

```
auto cl::sycl::make_range (
    range< 2 > r ) [inline]

#include <include/CL/sycl/range.hpp>
```

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

```
00070 { return r; }
```

**8.9.3.7** `make_range()` [3/4]

```
auto cl::sycl::make_range (
    range< 3 > r ) [inline]

#include <include/CL/sycl/range.hpp>
```

Definition at line 71 of file [range.hpp](#).

```
00071 { return r; }
```

**8.9.3.8** `make_range()` [4/4]

```
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 78 of file [range.hpp](#).

```
00078 {
00079     // Call constructor directly to allow narrowing
00080     return range<sizeof...(Args)>(Args...);
00081 }
```

### 8.9.3.9 parallel\_for() [1/4]

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

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

Implementation of a data parallel computation with parallelism specified at launch time by a range<>.

Kernel index is id or int.

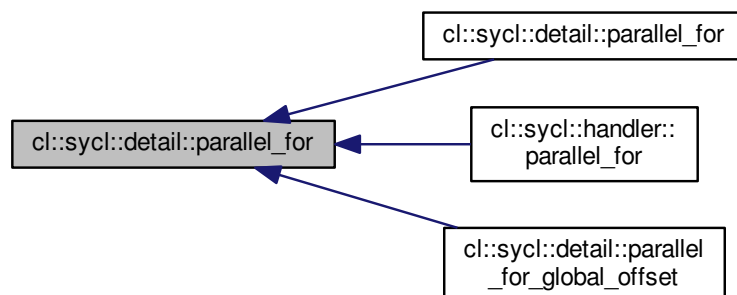
This implementation use OpenMP 3 if compiled with the right flag.

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

Referenced by [cl::sycl::detail::parallel\\_for\(\)](#), [cl::sycl::handler::parallel\\_for\(\)](#), and [cl::sycl::detail::parallel\\_for\\_global\\_offset\(\)](#).

```
00127         {
00128     #ifdef _OPENMP
00129         // Use OpenMP for the top loop level
00130         parallel_OpenMP_for_iterate<Dimensions,
00131                                     range<Dimensions>,
00132                                     ParallelForFunctor,
00133                                     id<Dimensions>> { r, f };
00134     #else
00135         // In a sequential execution there is only one index processed at a time
00136         id<Dimensions> index;
00137         parallel_for_iterate<Dimensions,
00138                             range<Dimensions>,
00139                             ParallelForFunctor,
00140                             id<Dimensions>> { r, f, index };
00141     #endif
00142 }
```

Here is the caller graph for this function:



8.9.3.10 `parallel_for()` [2/4]

```
template<int 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 151 of file `parallelism.hpp`.

```
00153     {
00154     auto reconstruct_item = [&] (id<Dimensions> l) {
00155         // Reconstruct the global item
00156         item<Dimensions> index { r, l };
00157         // Call the user kernel with the item<> instead of the id<>
00158         f(index);
00159     };
00160 #ifdef _OPENMP
00161     // Use OpenMP for the top loop level
00162     parallel_OpenMP_for_iterate<Dimensions,
00163                             range<Dimensions>,
00164                             decltype(reconstruct_item),
00165                             id<Dimensions>> { r, reconstruct_item } ;
00166 #else
00167     // In a sequential execution there is only one index processed at a time
00168     id<Dimensions> index;
00169     parallel_for_iterate<Dimensions,
00170                       range<Dimensions>,
00171                       decltype(reconstruct_item),
00172                       id<Dimensions>> { r, reconstruct_item, index } ;
00173 #endif
00174 }
```

8.9.3.11 `parallel_for()` [3/4]

```
template<int 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 182 of file `parallelism.hpp`.

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

```
00182     {
00183     using mf_t = decltype(std::mem_fn(&ParallelForFuncor::operator())) ;
00184     using arg_t = typename mf_t::second_argument_type;
00185     parallel_for(r, f, arg_t{});
00186 }
```

Here is the call graph for this function:



### 8.9.3.12 `parallel_for()` [4/4]

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

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

Implement a variation of `parallel_for` to take into account a `nd_range<>`

**Todo** Add an OpenMP implementation

**Todo** Deal with incomplete work-groups

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

Definition at line 217 of file `parallelism.hpp`.

References `cl::sycl::nd_range< Dimensions >::get_group()`, `cl::sycl::nd_range< Dimensions >::get_local()`, and `cl::sycl::detail::parallel_for_workitem()`.

```
00218                                     {
00219     // To iterate on the work-group
00220     id<Dimensions> group;
00221     range<Dimensions> group_range = r.get_group();
00222
00223 #ifdef _OPENMP
00224
00225     auto iterate_in_work_group = [&] (id<Dimensions> g) {
00226         //group.display();
00227
00228         // Then iterate on the local work-groups
00229         cl::sycl::group<Dimensions> wg {g, r};
00230         parallel_for_workitem<Dimensions>
00231             decltype(f)>(wg, f);
00232     };
00233
00234 #else
00235
00236     // In a sequential execution there is only one index processed at a time
00237     nd_item<Dimensions> index { r };
00238
00239     // To iterate on the local work-item
00240     id<Dimensions> local;
00241     range<Dimensions> local_range = r.get_local();
00242
```

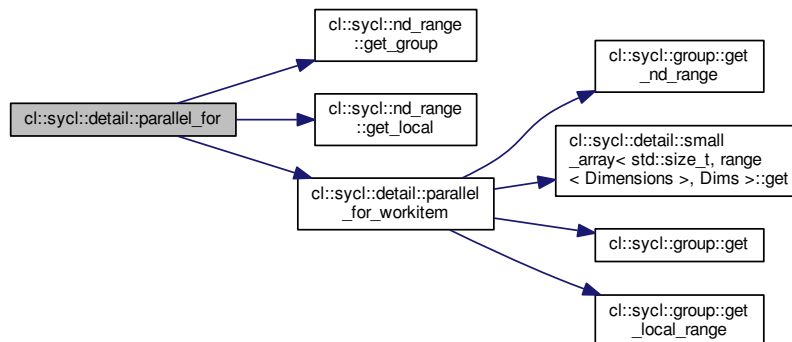


```

00243 // Reconstruct the nd_item from its group and local id
00244 auto reconstruct_item = [&] (id<Dimensions> l) {
00245     //local.display();
00246     // Reconstruct the global nd_item
00247     index.set_local(local);
00248     // Upgrade local_range to an id<> so that we can * with the group (an id<>)
00249     index.set_global(local + id<Dimensions>(local_range)*group);
00250     // Call the user kernel at last
00251     f(index);
00252 };
00253
00254 /* To recycle the parallel_for on range<>, wrap the ParallelForFuncion f
00255    into another functor that iterates inside the work-group and then
00256    calls f */
00257 auto iterate_in_work_group = [&] (id<Dimensions> g) {
00258     //group.display();
00259     // Then iterate on the local work-groups
00260     parallel_for_iterate<Dimensions,
00261         range<Dimensions>,
00262         decltype(reconstruct_item),
00263         id<Dimensions>>> { local_range,
00264             reconstruct_item,
00265             local };
00266 };
00267
00268 #endif
00269
00270 // First iterate on all the work-groups
00271 parallel_for_iterate<Dimensions,
00272     range<Dimensions>,
00273     decltype(iterate_in_work_group),
00274     id<Dimensions>>> { group_range,
00275         iterate_in_work_group,
00276         group };
00277 }

```

Here is the call graph for this function:



### 8.9.3.13 parallel\_for\_global\_offset()

```

template<int Dimensions = 1, typename ParallelForFuncion >
void cl::sycl::detail::parallel_for_global_offset (
    range< Dimensions > global_size,
    id< Dimensions > offset,
    ParallelForFuncion f )

```

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

Implementation of `parallel_for` with a `range<>` and an offset.

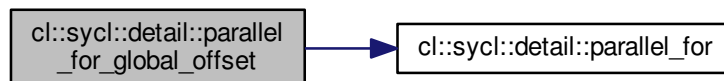
Definition at line 191 of file `parallelism.hpp`.

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

```

00193
00194 // Reconstruct the item from its id<> and its offset
00195 auto reconstruct_item = [&] (id<Dimensions> l) {
00196     // Reconstruct the global item
00197     item<Dimensions> index { global_size, l + offset, offset };
00198     // Call the user kernel with the item<> instead of the id<>
00199     f(index);
00200 };
00201
00202 // First iterate on all the work-groups
00203 parallel_for(global_size, reconstruct_item);
00204 }
```

Here is the call graph for this function:



#### 8.9.3.14 `parallel_for_work_item()`

```

template<int Dimensions = 1, typename ParallelForFuncor >
void cl::sycl::parallel_for_work_item (
    const group< Dimensions > & g,
    ParallelForFuncor 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 39 of file `parallelism.hpp`.

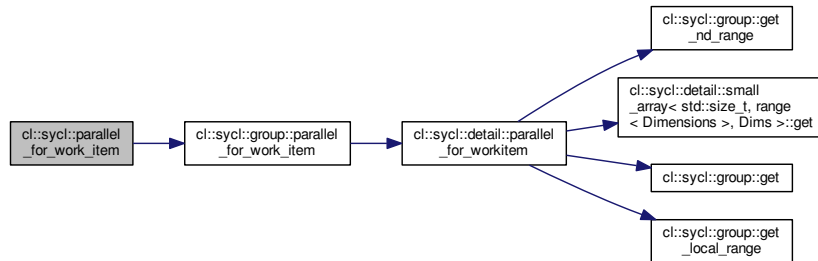
References `cl::sycl::group< Dimensions >::parallel_for_work_item()`.

```

00040                                     {
00041     g.parallel_for_work_item(f);
00042 }

```

Here is the call graph for this function:



### 8.9.3.15 parallel\_for\_workgroup()

```

template<int Dimensions = 1, typename ParallelForFuncor >
void cl::sycl::detail::parallel_for_workgroup (
    nd_range< Dimensions > r,
    ParallelForFuncor f )

```

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

Implement the loop on the work-groups.

Definition at line 282 of file [parallelism.hpp](#).

References [cl::sycl::nd\\_range< Dimensions >::get\\_group\(\)](#).

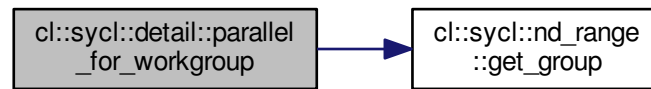
Referenced by [cl::sycl::handler::parallel\\_for\\_work\\_group\(\)](#).

```

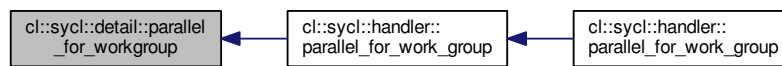
00283                                     {
00284     // In a sequential execution there is only one index processed at a time
00285     group<Dimensions> g { r };
00286
00287     // First iterate on all the work-groups
00288     parallel_for_iterate<Dimensions,
00289                         range<Dimensions>,
00290                         ParallelForFuncor,
00291                         group<Dimensions>> {
00292         r.get_group(),
00293         f,
00294         g };
00295 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.9.3.16 parallel\_for\_workitem()

```

template<int 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 303 of file `parallelism.hpp`.

References `cl::sycl::group< Dimensions >::get()`, `cl::sycl::detail::small_array< std::size_t, range< Dimensions >, Dims >::get()`, `cl::sycl::group< Dimensions >::get_local_range()`, and `cl::sycl::group< Dimensions >::get_nd_range()`.

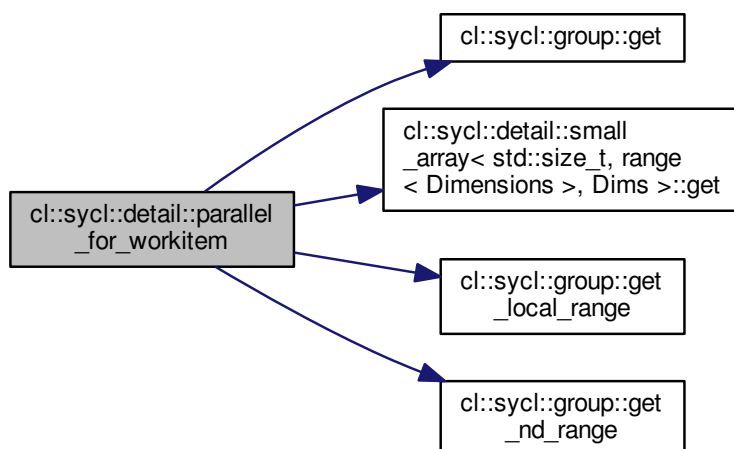
Referenced by `cl::sycl::detail::parallel_for()`, and `cl::sycl::group< Dimensions >::parallel_for_work_item()`.

```

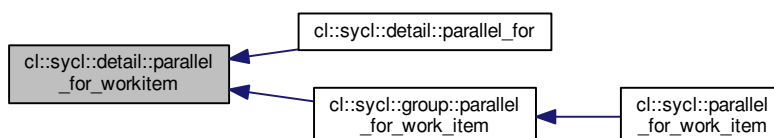
00304 {
00305 #if defined(_OPENMP) && (!defined(TRISYCL_NO_BARRIER) && !defined(_MSC_VER))
00306 /* To implement barriers With OpenMP, one thread is created for each
00307    work-item in the group and thus an OpenMP barrier has the same effect
00308    of an OpenCL barrier executed by the work-items in a workgroup
00309
00310    The issue is that the parallel_for_workitem() execution is slow even
00311    when nd_item::barrier() is not used
00312
00313    \todo Simplify by just using omp parallel for collapse
00314 */
00315
00316 range<Dimensions> l_r = g.get_nd_range().get_local();
00317 auto tot = l_r.get(0);
00318 for (int i = 1; i < (int) Dimensions; ++i){
00319     tot *= l_r.get(i);
00320 }
00321 #pragma omp parallel num_threads(tot)
00322 {
00323     nd_item<Dimensions> index { g.get_nd_range() };
00324     id<Dimensions> local; // to initialize correctly
00325 #pragma omp for nowait
00326     for (std::size_t th_id = 0; th_id < tot; ++th_id) {
00327         if (Dimensions == 1) {
00328             local[0] = th_id;
00329         } else if (Dimensions == 2) {
00330             local[0] = th_id / l_r.get(1);
00331             local[1] = th_id % l_r.get(1);
00332         } else if (Dimensions == 3) {
00333             local[0] = th_id / (l_r.get(1)*l_r.get(2));
00334             local[1] = (th_id / l_r.get(2)) % l_r.get(1);
00335             local[2] = th_id % l_r.get(2);
00336         }
00337         index.set_local(local);
00338         index.set_global(local + id<Dimensions>(l_r)*g.get());
00339         f(index);
00340     }
00341 }
00342 #else
00343 // In a sequential execution there is only one index processed at a time
00344 nd_item<Dimensions> index { g.get_nd_range() };
00345 // To iterate on the local work-item
00346 id<Dimensions> local;
00347
00348 // Reconstruct the nd_item from its group and local id
00349 auto reconstruct_item = [&] (id<Dimensions> l) {
00350     //local.display();
00351     //l.display();
00352     // Reconstruct the global nd_item
00353     index.set_local(local);
00354     // \todo Some strength reduction here
00355     index.set_global(local + id<Dimensions>(g.get_local_range())*g.get());
00356     // Call the user kernel at last
00357     f(index);
00358 };
00359
00360 // Then iterate on all the work-items of the work-group
00361 parallel_for_iterate<Dimensions,
00362     range<Dimensions>,
00363     decltype(reconstruct_item),
00364     id<Dimensions>> {
00365     g.get_local_range(),
00366     reconstruct_item,
00367     local };
00368 #endif
00369 }

```

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 `actual_type=unsigned char`, `uchar4` is equivalent to `vec<unsigned char, 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

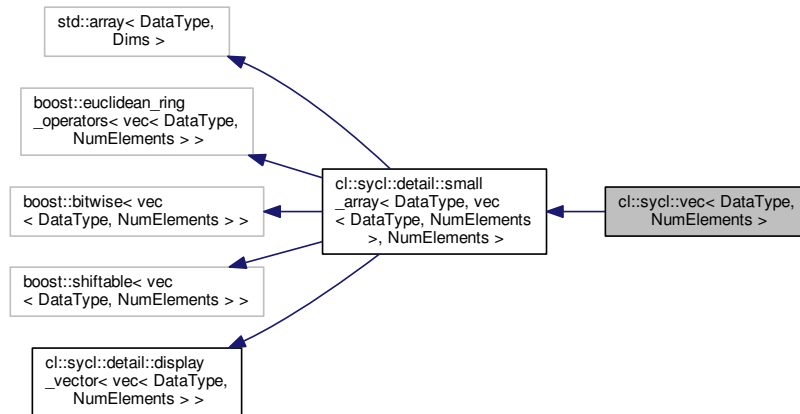
**Todo** move the implementation elsewhere

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

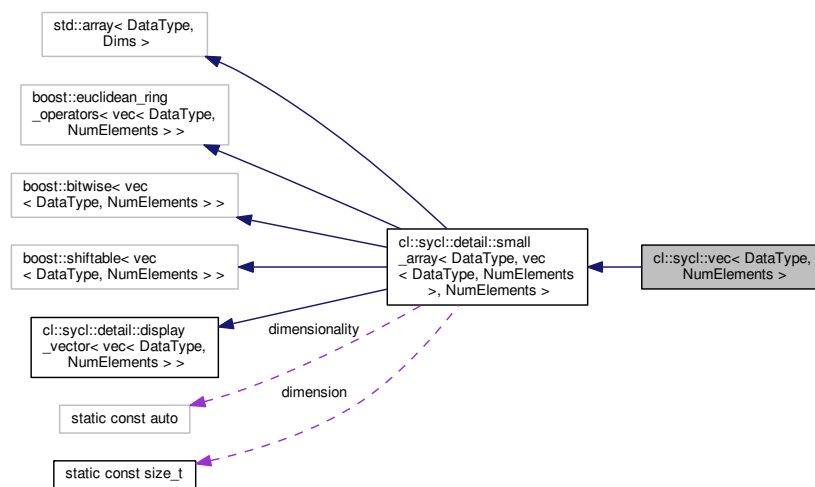
**Todo** rename in the specification `element_type` to `value_type`

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 `basic_type`

```
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 `vec()` [1/2]

```
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 `vec()` [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 `flatten()` [1/2]

```
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** `flatten()` [2/2]

```
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** `flatten_to_tuple()`

```
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 TRISYCL\_DEFINE\_VEC\_TYPE

```
#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 163 of file [vec.hpp](#).

### 8.10.3.2 TRISYCL\_DEFINE\_VEC\_TYPE\_SIZE

```
#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 actual\_type=unsigned char, uchar4 is equivalent to vec<unsigned char, 4>

Definition at line 159 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](#)  
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\\_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...](#)*
- struct [is\\_wrapper](#)
- 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 constant_ptr = constant< T * >`  
*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 global_ptr = global< T * >`  
*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 local_ptr = local< T * >`  
*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 T >`  
`using private_ptr = priv< T * >`  
*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 allocator objects give the programmer some control on how the memory is allocated inside SYCL.*
- `template<typename T >`  
`using image_allocator = std::allocator< T >`  
*The allocator used for the image inside SYCL.*
- `template<typename T >`  
`using map_allocator = std::allocator< T >`  
*The allocator used to map the memory at the same place.*
- `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_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 `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 `host_selector` = `device_type_name_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.*
- template<>  
    auto `device::get_info`< `info::device::max_work_group_size` > () const
- template<>  
    auto `device::get_info`< `info::device::max_compute_units` > () const
- template<>  
    auto `device::get_info`< `info::device::device_type` > () const
- template<>  
    auto `device::get_info`< `info::device::local_mem_size` > () const
- template<>  
    auto `device::get_info`< `info::device::max_mem_alloc_size` > () const
- template<>  
    auto `device::get_info`< `info::device::vendor` > () const
- template<>  
    auto `device::get_info`< `info::device::name` > () const
- template<>  
    auto `device::get_info`< `info::device::profile` > () const
- 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)`*
- `TRISYCL_MATH_WRAP` (abs) `TRISYCL_MATH_WRAP`(atan) `TRISYCL_MATH_WRAP2s`(fmax) `TRISYCL_MATH_WRAP2s`(fmin) `TRISYCL_MATH_WRAP2s`(frexp) template< typename T > T max(T x

- `template<typename T >`  
`T min (T x, T y, T z)`
- `TRISYCL_MATH_WRAP2s` (modf) `TRISYCL_MATH_WRAP3s`(remquo) `TRISYCL_MATH_WRAP2`(rotate)  
`namespace native`
- `template<int Dimensions = 1, typename ParallelForFuncor >`  
`void parallel_for_work_item (const group< Dimensions > &g, ParallelForFuncor 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)*

## Variables

- `T y`
- `T T z`

## 9.2.1 Typedef Documentation

### 9.2.1.1 function\_class

```
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 mutex\_class

```
using cl::sycl::mutex_class = typedef std::mutex
```

Definition at line 69 of file [default\\_classes.hpp](#).

### 9.2.1.3 shared\_ptr\_class

```
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 string\_class

```
using cl::sycl::string_class = typedef std::string
```

Definition at line 40 of file [default\\_classes.hpp](#).

#### 9.2.1.5 unique\_ptr\_class

```
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 vector\_class

```
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 weak\_ptr\_class

```
template<class T >  
using cl::sycl::weak_ptr_class = typedef std::weak_ptr<T>
```

Definition at line 114 of file [default\\_classes.hpp](#).

## 9.2.2 Function Documentation

#### 9.2.2.1 min()

```
template<typename T >  
T cl::sycl::min (  
    T x,  
    T y,  
    T z )
```

Definition at line 120 of file [math.hpp](#).

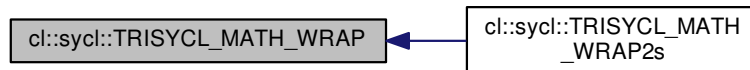
```
00120     {  
00121     return std::min(x, std::min(y, z));  
00122 }
```

### 9.2.2.2 TRISYCL\_MATH\_WRAP()

```
cl::sycl::TRISYCL_MATH_WRAP (
    abs )
```

Referenced by [TRISYCL\\_MATH\\_WRAP2s\(\)](#).

Here is the caller graph for this function:



### 9.2.2.3 TRISYCL\_MATH\_WRAP2s()

```
cl::sycl::TRISYCL_MATH_WRAP2s (
    modf )
```

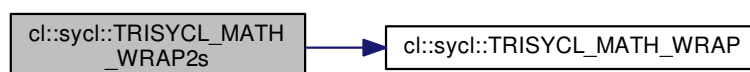
Definition at line 128 of file [math.hpp](#).

References [TRISYCL\\_MATH\\_WRAP\(\)](#).

```

00166         {
00167     TRISYCL_MATH_WRAP(cos)
00168     /*TRISYCL_MATH_WRAP2(divide)
00169     TRISYCL_MATH_WRAP(exp)
00170     TRISYCL_MATH_WRAP(exp2)
00171     /*TRISYCL_MATH_WRAP(exp10)
00172     TRISYCL_MATH_WRAP(log)
00173     TRISYCL_MATH_WRAP(log2)
00174     TRISYCL_MATH_WRAP(log10)
00175     /*TRISYCL_MATH_WRAP(powr)
00176     /*TRISYCL_MATH_WRAP( recip)
00177     /*TRISYCL_MATH_WRAP( rsqrt)
00178     TRISYCL_MATH_WRAP(sin)
00179     TRISYCL_MATH_WRAP(sqrt)
00180     TRISYCL_MATH_WRAP(tan)
00181 }
```

Here is the call graph for this function:



### 9.2.3 Variable Documentation

#### 9.2.3.1 `y`

`T cl::sycl::y`

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

#### 9.2.3.2 `z`

`T T cl::sycl::z`

**Initial value:**

```
{
    return std::max(x, std::max(y, z))
}
```

Definition at line 109 of file [math.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 `fence_space`

```
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 mode

```
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 <i>not</i> 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 target

```
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...](#)*
- class [accessor< T, Dimensions, Mode, access::target::local >](#)  
*The local accessor specialization abstracts the way local memory is allocated to a kernel to be shared between work-items of the same work-group. [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. [More...](#)*

- 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 [container\\_element\\_aspect](#)  
A mix-in to add some container element aspects. [More...](#)
- class [context](#)
- 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\\_context](#)
- 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...](#)
- struct [ocl\\_type](#)  
Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device. [More...](#)
- struct [ocl\\_type< T, constant\\_address\\_space >](#)  
Add an attribute for `__constant` address space. [More...](#)
- struct [ocl\\_type< T, generic\\_address\\_space >](#)  
Add an attribute for `__generic` address space. [More...](#)
- struct [ocl\\_type< T, global\\_address\\_space >](#)  
Add an attribute for `__global` address space. [More...](#)
- struct [ocl\\_type< T, local\\_address\\_space >](#)  
Add an attribute for `__local` address space. [More...](#)
- struct [ocl\\_type< T, private\\_address\\_space >](#)  
Add an attribute for `__private` address space. [More...](#)
- class [opengl\\_context](#)  
SYCL OpenGL context.
- class [opengl\\_device](#)  
SYCL OpenGL device.
- class [opengl\\_kernel](#)  
An abstraction of the OpenCL kernel.
- class [opengl\\_platform](#)  
SYCL OpenGL platform. [More...](#)
- class [opengl\\_queue](#)  
Some implementation details about the SYCL queue.
- 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 Dimensions = 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

*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 , int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_const_t<T>>>>`  
`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<int 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.*
- `template<int 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<int 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<int 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<int Dimensions = 1, typename ParallelForFuncor >`  
`void parallel\_for\_global\_offset (range< Dimensions > global_size, id< Dimensions > offset, ParallelFor_↵`  
`Funcor f)`  
*Implementation of [parallel\\_for](#) with a range<> and an offset.*
- `template<int 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<int Dimensions = 1, typename ParallelForFuncor >`  
`void parallel\_for\_workgroup (nd\_range< Dimensions > r, ParallelForFuncor f)`  
*Implement the loop on the work-groups.*

## Variables

- `TRISYCL_WEAK_ATTRIB_PREFIX detail::cache< cl_context, detail::opengl\_context > opengl\_context\_↵`  
`::cache TRISYCL_WEAK_ATTRIB_SUFFIX`

### 9.4.1 Function Documentation

#### 9.4.1.1 add\_buffer\_to\_task()

```
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 438 of file [handler.hpp](#).

References [cl::sycl::handler::task](#).

```
00440                                     {
00441     command_group_handler->task->add_buffer(b, is_write_mode);
00442     return command_group_handler->task;
00443 }
```

## 9.5 cl::sycl::info Namespace Reference

### Classes

- struct [param\\_traits](#)

*Implement a meta-function from  $(T, \text{value})$  to  $T'$  to express the return type value of an OpenCL function of kind  $(T, \text{value})$*

### 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::devices`, `context::gl_interop` }  
*Context information descriptors.*
- enum `device_type` : unsigned int {  
`device_type::cpu`, `device_type::gpu`, `device_type::accelerator`, `device_type::custom`,  
`device_type::defaults`, `device_type::host`, `device_type::opencl`, `device_type::all` }  
*Type of devices.*
- enum `device` : int {  
`device::device_type`, `device::vendor_id`, `device::max_compute_units`, `device::max_work_item_dimensions`,  
`device::max_work_item_sizes`, `device::max_work_group_size`, `device::preferred_vector_width_char`,  
`device::preferred_vector_width_short`,  
`device::preferred_vector_width_int`, `device::preferred_vector_width_long_long`, `device::preferred_vector_↵`  
`width_float`, `device::preferred_vector_width_double`,  
`device::preferred_vector_width_half`, `device::native_vector_witdth_char`, `device::native_vector_witdth_short`,  
`device::native_vector_witdth_int`,  
`device::native_vector_witdth_long_long`, `device::native_vector_witdth_float`, `device::native_vector_witdth_↵`  
`double`, `device::native_vector_witdth_half`,  
`device::max_clock_frequency`, `device::address_bits`, `device::max_mem_alloc_size`, `device::image_support`,  
`device::max_read_image_args`, `device::max_write_image_args`, `device::image2d_max_height`, `device_↵`  
`::image2d_max_width`,  
`device::image3d_max_height`, `device::image3d_max_widht`, `device::image3d_mas_depth`, `device::image_↵`  
`max_buffer_size`,  
`device::image_max_array_size`, `device::max_samplers`, `device::max_parameter_size`, `device::mem_base_↵`  
`_addr_align`,  
`device::single_fp_config`, `device::double_fp_config`, `device::global_mem_cache_type`, `device::global_mem_↵`  
`_cache_line_size`,  
`device::global_mem_cache_size`, `device::global_mem_size`, `device::max_constant_buffer_size`, `device_↵`  
`::max_constant_args`,  
`device::local_mem_type`, `device::local_mem_size`, `device::error_correction_support`, `device::host_unified_↵`  
`memory`,  
`device::profiling_timer_resolution`, `device::endian_little`, `device::is_available`, `device::is_compiler_available`,  
`device::is_linker_available`, `device::execution_capabilities`, `device::queue_properties`, `device::built_in_↵`  
`kernels`,  
`device::platform`, `device::name`, `device::vendor`, `device::driver_version`,  
`device::profile`, `device::device_version`, `device::opencl_version`, `device::extensions`,  
`device::printf_buffer_size`, `device::preferred_interop_user_sync`, `device::parent_device`, `device::partition_↵`  
`max_sub_devices`,  
`device::partition_properties`, `device::partition_affinity_domain`, `device::partition_type`, `device::reference_↵`  
`count` }  
*Device information descriptors.*
- enum `device_partition_property` : int {  
`device_partition_property::unsupported`, `device_partition_property::partition_equally`, `device_partition_↵`  
`property::partition_by_counts`, `device_partition_property::partition_by_affinity_domain`,  
`device_partition_property::partition_affinity_domain_next_partitionable` }
- enum `device_affinity_domain` : int {  
`device_affinity_domain::unsupported`, `device_affinity_domain::numa`, `device_affinity_domain::L4_cache`,  
`device_affinity_domain::L3_cache`,  
`device_affinity_domain::L2_cache`, `device_affinity_domain::next_partitionable` }
- enum `device_partition_type` : int {  
`device_partition_type::no_partition`, `device_partition_type::numa`, `device_partition_type::L4_cache`, `device_↵`  
`_partition_type::L3_cache`,  
`device_partition_type::L2_cache`, `device_partition_type::L1_cache` }
- enum `local_mem_type` : int { `local_mem_type::none`, `local_mem_type::local`, `local_mem_type::global` }
- enum `fp_config` : int {  
`fp_config::denorm`, `fp_config::inf_nan`, `fp_config::round_to_nearest`, `fp_config::round_to_zero`,  
`fp_config::round_to_inf`, `fp_config::fma`, `fp_config::correctly_rounded_divide_sqrt`, `fp_config::soft_float` }

- enum `global_mem_cache_type` : int { `global_mem_cache_type::none`, `global_mem_cache_type::read_only`, `global_mem_cache_type::write_only` }
  - enum `device_execution_capabilities` : unsigned int { `device_execution_capabilities::exec_kernel`, `device_execution_capabilities::exec_native_kernel` }
  - enum `platform` : unsigned int { `platform::TRISYCL_SKIP_OPENCL` `!=(= CL_PLATFORM_PROFILE)`, `platform::TRISYCL_SKIP_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 queue\_profiling

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 queue

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::context >](#)
- struct [hash< cl::sycl::device >](#)
- struct [hash< cl::sycl::kernel >](#)
- struct [hash< cl::sycl::platform >](#)
- struct [hash< cl::sycl::queue >](#)

## Chapter 10

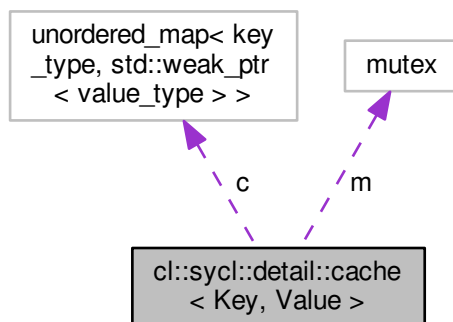
# Class Documentation

### 10.1 `cl::sycl::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.*

### 10.1.1 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](#).

### 10.1.2 Member Typedef Documentation

#### 10.1.2.1 key\_type

```
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](#).

#### 10.1.2.2 value\_type

```
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](#).



### 10.1.3 Member Function Documentation

#### 10.1.3.1 `get_or_register()`

```
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 <code>shared_ptr</code> |

##### Returns

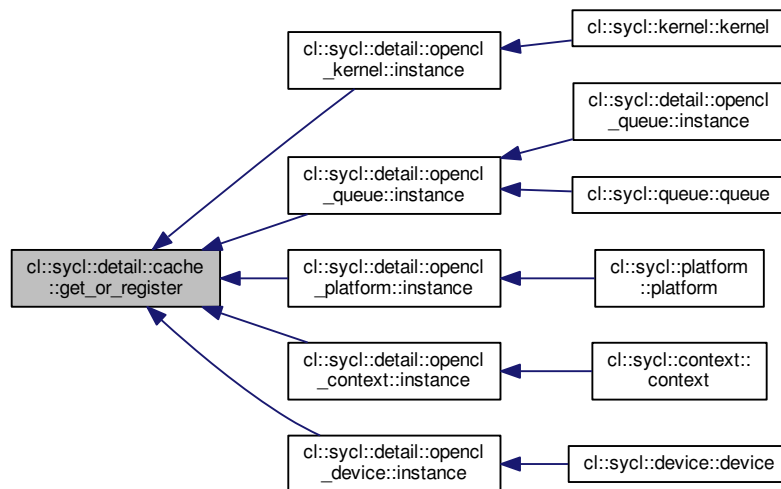
a `shared_ptr` to the value retrieved or inserted

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

Referenced by [cl::sycl::detail::opencl\\_kernel::instance\(\)](#), [cl::sycl::detail::opencl\\_queue::instance\(\)](#), [cl::sycl::detail::opencl\\_platform::instance\(\)](#), [cl::sycl::detail::opencl\\_context::instance\(\)](#), and [cl::sycl::detail::opencl\\_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:



### 10.1.3.2 remove()

```

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 |
|----|----------|---|

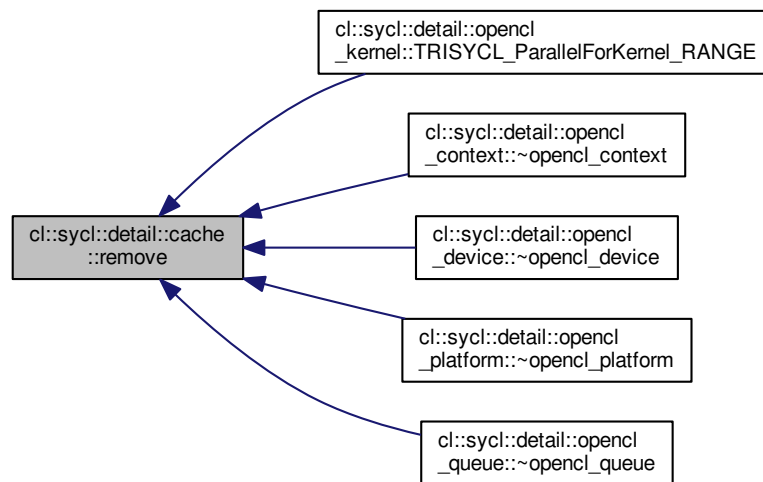
Definition at line 83 of file [cache.hpp](#).

Referenced by [cl::sycl::detail::opencl\\_kernel::TRISYCL\\_ParallelForKernel\\_RANGE\(\)](#), [cl::sycl::detail::opencl\\_context::~~opencl\\_context\(\)](#), [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:



## 10.1.4 Member Data Documentation

### 10.1.4.1 `c`

```
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](#).

### 10.1.4.2 `m`

```
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](#).

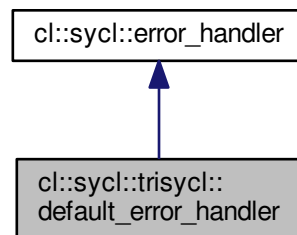
The documentation for this class was generated from the following file:

- [include/CL/sycl/detail/cache.hpp](#)

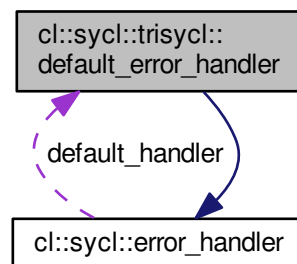
## 10.2 cl::sycl::trisycl::default\_error\_handler Struct Reference

```
#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

#### 10.2.1 Detailed Description

Definition at line 52 of file [error\\_handler.hpp](#).

## 10.2.2 Member Function Documentation

### 10.2.2.1 report\_error()

```
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 54 of file [error\\_handler.hpp](#).

```
00054                                     {
00055     }
```

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

- include/CL/sycl/[error\\_handler.hpp](#)

## 10.3 cl::sycl::event Class Reference

```
#include <event.hpp>
```

### Public Member Functions

- [event](#) ()=default

### 10.3.1 Detailed Description

Definition at line 14 of file [event.hpp](#).

### 10.3.2 Constructor & Destructor Documentation

### 10.3.2.1 event()

```
cl::sycl::event::event ( ) [default]
```

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

- [include/CL/sycl/event.hpp](#)

## 10.4 handler\_event Class Reference

Handler event.

```
#include <handler_event.hpp>
```

### 10.4.1 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](#)

## 10.5 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

### 10.5.1 Detailed Description

```
template<typename T, int Dimensions, typename Allocator>
struct std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >
```

Definition at line 542 of file [buffer.hpp](#).

## 10.5.2 Member Function Documentation

### 10.5.2.1 operator()()

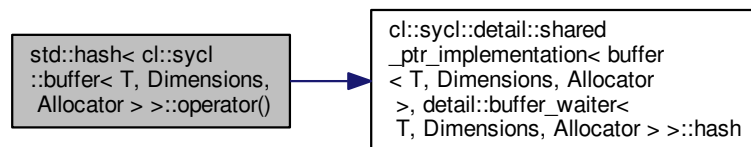
```
template<typename T , int Dimensions, typename Allocator >
auto std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >::operator() (
    const cl::sycl::buffer< T, Dimensions, Allocator > & b ) const [inline]
```

Definition at line 544 of file [buffer.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< buffer< T, Dimensions, Allocator >, detail::buffer\\_waiter< T, Dimensions, Allocator > >::hash\(\)](#).

```
00544                                     {
00545     // Forward the hashing to the implementation
00546     return b.hash();
00547 }
```

Here is the call graph for this function:



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

- [include/CL/sycl/buffer.hpp](#)

## 10.6 std::hash< cl::sycl::context > Struct Template Reference

```
#include <context.hpp>
```

### Public Member Functions

- auto [operator\(\)](#) (const [cl::sycl::context](#) &c) const

### 10.6.1 Detailed Description

```
template<>
struct std::hash< cl::sycl::context >
```

Definition at line 227 of file [context.hpp](#).

### 10.6.2 Member Function Documentation

#### 10.6.2.1 operator>()

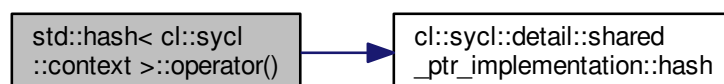
```
auto std::hash< cl::sycl::context >::operator() (
    const cl::sycl::context & c ) const [inline]
```

Definition at line 228 of file [context.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< Parent, Implementation >::hash\(\)](#).

```
00228                                     {
00229     return c.hash();
00230 }
```

Here is the call graph for this function:



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

- [include/CL/sycl/context.hpp](#)

## 10.7 std::hash< cl::sycl::device > Struct Template Reference

```
#include <device.hpp>
```

### Public Member Functions

- auto [operator\(\)](#) (const [cl::sycl::device](#) &d) const



### 10.7.1 Detailed Description

```
template<>
struct std::hash< cl::sycl::device >
```

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

### 10.7.2 Member Function Documentation

#### 10.7.2.1 `operator()()`

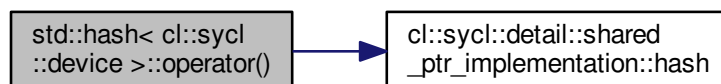
```
auto std::hash< cl::sycl::device >::operator() (
    const cl::sycl::device & d ) const [inline]
```

Definition at line 314 of file [device.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< Parent, Implementation >::hash\(\)](#).

```
00314                                     {
00315     // Forward the hashing to the implementation
00316     return d.hash();
00317 }
```

Here is the call graph for this function:



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

- [include/CL/sycl/device.hpp](#)

## 10.8 `std::hash< cl::sycl::kernel >` Struct Template Reference

```
#include <kernel.hpp>
```

### Public Member Functions

- auto [operator\(\)](#) (const [cl::sycl::kernel](#) &k) const

### 10.8.1 Detailed Description

```
template<>
struct std::hash< cl::sycl::kernel >
```

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

### 10.8.2 Member Function Documentation

#### 10.8.2.1 operator>()

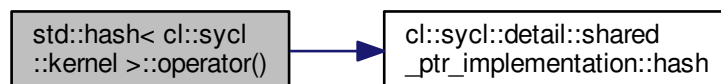
```
auto std::hash< cl::sycl::kernel >::operator() (
    const cl::sycl::kernel & k ) const [inline]
```

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

References [cl::sycl::detail::shared\\_ptr\\_implementation< Parent, Implementation >::hash\(\)](#).

```
00127
00128     // Forward the hashing to the implementation
00129     return k.hash();
00130 }
```

Here is the call graph for this function:



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

- [include/CL/sycl/kernel.hpp](#)

## 10.9 std::hash< cl::sycl::platform > Struct Template Reference

```
#include <platform.hpp>
```

### Public Member Functions

- auto [operator\(\)](#) (const [cl::sycl::platform](#) &p) const

### 10.9.1 Detailed Description

```
template<>
struct std::hash< cl::sycl::platform >
```

Definition at line 199 of file [platform.hpp](#).

### 10.9.2 Member Function Documentation

#### 10.9.2.1 operator>()

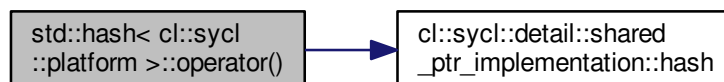
```
auto std::hash< cl::sycl::platform >::operator() (
    const cl::sycl::platform & p ) const [inline]
```

Definition at line 201 of file [platform.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< Parent, Implementation >::hash\(\)](#).

```
00201     {
00202         // Forward the hashing to the implementation
00203         return p.hash();
00204     }
```

Here is the call graph for this function:



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

- [include/CL/sycl/platform.hpp](#)

## 10.10 std::hash< cl::sycl::queue > Struct Template Reference

```
#include <queue.hpp>
```

### Public Member Functions

- auto [operator\(\)](#) (const [cl::sycl::queue](#) &q) const

### 10.10.1 Detailed Description

```
template<>
struct std::hash< cl::sycl::queue >
```

Definition at line 389 of file [queue.hpp](#).

### 10.10.2 Member Function Documentation

#### 10.10.2.1 operator()

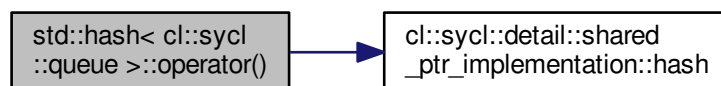
```
auto std::hash< cl::sycl::queue >::operator() (
    const cl::sycl::queue & q ) const [inline]
```

Definition at line 391 of file [queue.hpp](#).

References [cl::sycl::detail::shared\\_ptr\\_implementation< Parent, Implementation >::hash\(\)](#).

```
00391                                     {
00392     // Forward the hashing to the implementation
00393     return q.hash();
00394 }
```

Here is the call graph for this function:



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

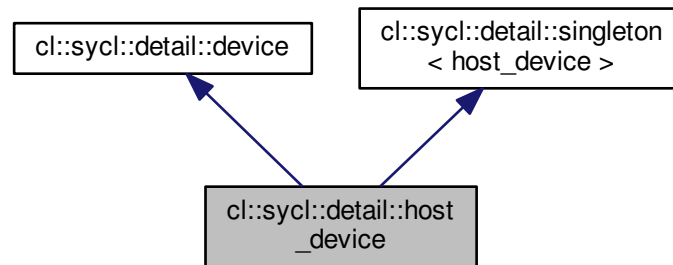
- [include/CL/sycl/queue.hpp](#)

## 10.11 cl::sycl::detail::host\_device Class Reference

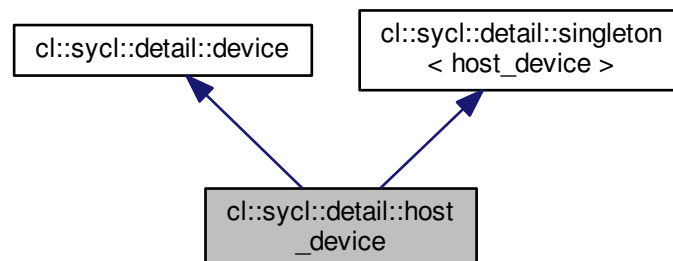
SYCL host device.

```
#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.*
- `boost::compute::device` & [get\\_boost\\_compute](#) () override  
*Return the underlying Boost.Compute device.*
- `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

### 10.11.1 Detailed Description

SYCL host device.

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

Definition at line 31 of file [host\\_device.hpp](#).

### 10.11.2 Member Function Documentation

#### 10.11.2.1 get()

```
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 }
```

## 10.11.2.2 get\_boost\_compute()

```
boost::compute::device& cl::sycl::detail::host_device::get_boost_compute ( ) [inline], [override],
[virtual]
```

Return the underlying Boost.Compute device.

This throws an error since there is no OpenCL device associated to the host device.

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

Definition at line 52 of file [host\\_device.hpp](#).

```
00052                                     {
00053     throw non_cl_error("The host device has no underlying OpenCL device");
00054 }
```

## 10.11.2.3 get\_platform()

```
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

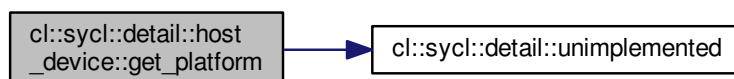
Implements [cl::sycl::detail::device](#).

Definition at line 88 of file [host\\_device.hpp](#).

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

```
00088                                     {
00089     detail::unimplemented();
00090     return {};
00091 }
```

Here is the call graph for this function:



#### 10.11.2.4 has\_extension()

```
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

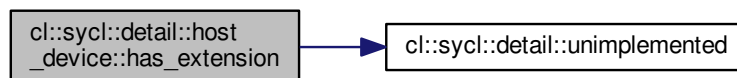
Implements [cl::sycl::detail::device](#).

Definition at line 112 of file [host\\_device.hpp](#).

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

```
00112                                     {
00113     detail::unimplemented();
00114     return {};
00115 }
```

Here is the call graph for this function:



#### 10.11.2.5 is\_accelerator()

```
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 77 of file [host\\_device.hpp](#).

```
00077                                     {
00078     return false;
00079 }
```



### 10.11.2.6 is\_cpu()

```
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 65 of file [host\\_device.hpp](#).

```
00065                                     {  
00066     return false;  
00067 }
```

### 10.11.2.7 is\_gpu()

```
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 71 of file [host\\_device.hpp](#).

```
00071                                     {  
00072     return false;  
00073 }
```

### 10.11.2.8 is\_host()

```
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 59 of file [host\\_device.hpp](#).

```
00059                                     {  
00060     return true;  
00061 }
```

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

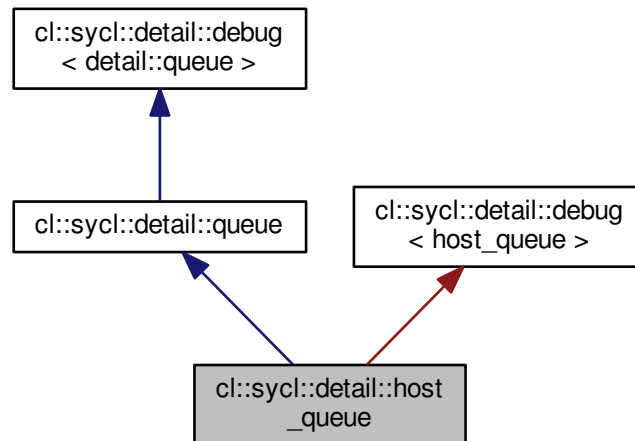
- [include/CL/sycl/device/detail/host\\_device.hpp](#)

## 10.12 cl::sycl::detail::host\_queue Class Reference

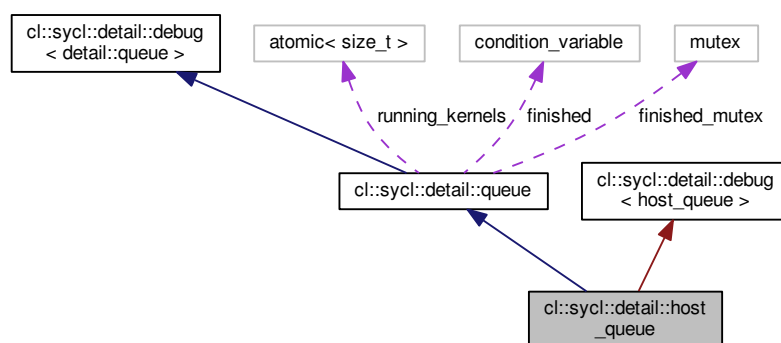
Some implementation details about the SYCL queue.

```
#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

### 10.12.1 Detailed Description

Some implementation details about the SYCL queue.

Note that a host queue is not a singleton, compared to host device or host platform, for example.

Definition at line 30 of file [host\\_queue.hpp](#).

### 10.12.2 Member Function Documentation

#### 10.12.2.1 get()

```
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 39 of file [host\\_queue.hpp](#).

```
00039                                     {
00040     throw non_cl_error("The host queue has no OpenCL command queue");
00041 }
```

### 10.12.2.2 `get_boost_compute()`

```
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 49 of file [host\\_queue.hpp](#).

```
00049                                     {  
00050     throw non_cl_error("The host queue has no OpenCL command queue");  
00051 }
```

### 10.12.2.3 `get_context()`

```
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 56 of file [host\\_queue.hpp](#).

```
00056                                     {  
00057     // Return the default context which is the host context  
00058     return {};  
00059 }
```

### 10.12.2.4 `get_device()`

```
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 63 of file [host\\_queue.hpp](#).

```
00063                                     {  
00064     // Return the default device which is the host device  
00065     return {};  
00066 }
```

## 10.12.2.5 is\_host()

```
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 70 of file [host\\_queue.hpp](#).

```
00070                                     {  
00071     return true;  
00072 }
```

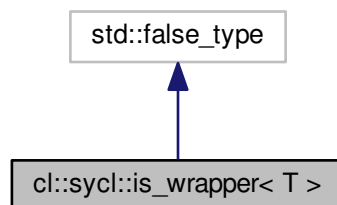
The documentation for this class was generated from the following file:

- [include/CL/sycl/queue/detail/host\\_queue.hpp](#)

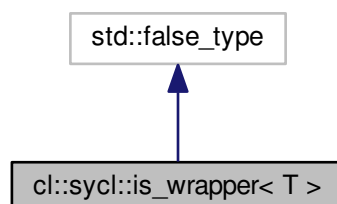
## 10.13 cl::sycl::is\_wrapper&lt; T &gt; Struct Template Reference

```
#include <opencl_types.hpp>
```

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



Collaboration diagram for `cl::sycl::is_wrapper< T >`:



### 10.13.1 Detailed Description

```
template<class T>
struct cl::sycl::is_wrapper< T >
```

Definition at line 81 of file [opencl\\_types.hpp](#).

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

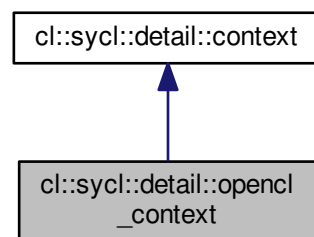
- [include/CL/sycl/opencl\\_types.hpp](#)

## 10.14 cl::sycl::detail::opencl\_context Class Reference

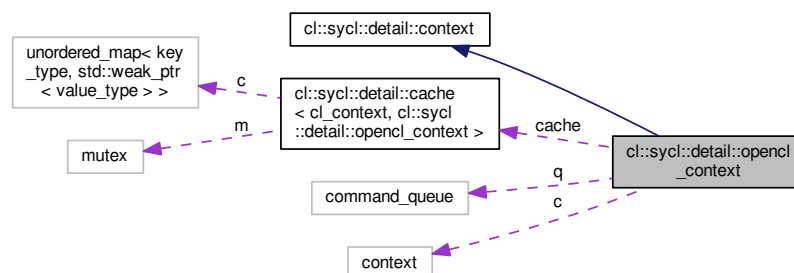
SYCL OpenCL context.

```
#include <opencl_context.hpp>
```

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



Collaboration diagram for `cl::sycl::detail::opencl_context`:



## Public Member Functions

- `cl_context get ()` const override  
*Return the underlying `cl_context` of the `cl::sycl::context`.*
- `boost::compute::context & get_boost_compute ()` override  
*Return the underlying `boost::compute::context` of the `cl::sycl::context`.*
- `boost::compute::command_queue & get_boost_queue ()` override  
*Return the queue that is associated to the context.*
- `bool is_host ()` const override  
*Return false because the context is not a SYCL host context.*
- `cl::sycl::platform get_platform ()` const override  
*Return the platform of the context.*
- `vector_class< cl::sycl::device > get_devices ()` const override  
*Returns the set of devices that are part of this context.*
- `~opencl_context ()` override  
*Unregister from the cache on destruction.*

## Static Public Member Functions

- static `std::shared_ptr< opencl_context > instance (const boost::compute::context &c)`  
*Get a singleton instance of the `opencl_context`.*

## Private Member Functions

- `opencl_context (const boost::compute::context &c)`  
*Only the instance factory can build it.*

## Private Attributes

- `boost::compute::context c`  
*User the Boost Compute abstraction of the OpenCL context.*
- `boost::compute::command_queue q`  
*A boost `command_queue` associated to an OpenCL context for when we need to transfer data but no queue is given (eg.*

## Static Private Attributes

- static `detail::cache< cl_context, detail::opencl_context > cache`  
*A cache to always return the same alive context for a given OpenCL context.*

### 10.14.1 Detailed Description

SYCL OpenCL context.

Definition at line 30 of file `opencl_context.hpp`.

## 10.14.2 Constructor & Destructor Documentation

### 10.14.2.1 `opengl_context()`

```
cl::sycl::detail::opengl_context::opengl_context (
    const boost::compute::context & c ) [inline], [private]
```

Only the instance factory can build it.

Definition at line 122 of file [opengl\\_context.hpp](#).

References [c](#).

```
00122                                     :
00123     c { c },
00124     q { boost::compute::command_queue { c, c.get_device() } } {}
```

### 10.14.2.2 `~opengl_context()`

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

Unregister from the cache on destruction.

Definition at line 129 of file [opengl\\_context.hpp](#).

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

```
00129                                     {
00130     cache.remove(c.get());
00131 }
```

Here is the call graph for this function:



## 10.14.3 Member Function Documentation



### 10.14.3.1 `get()`

```
cl_context cl::sycl::detail::opencl_context::get ( ) const [inline], [override], [virtual]
```

Return the underlying `cl_context` of the `cl::sycl::context`.

Implements `cl::sycl::detail::context`.

Definition at line 51 of file `opencl_context.hpp`.

```
00051                                     {
00052     return c.get();
00053 }
```

### 10.14.3.2 `get_boost_compute()`

```
boost::compute::context& cl::sycl::detail::opencl_context::get_boost_compute ( ) [inline],
[override], [virtual]
```

Return the underlying `boost::compute::context` of the `cl::sycl::context`.

Implements `cl::sycl::detail::context`.

Definition at line 59 of file `opencl_context.hpp`.

References `c`.

```
00059                                     {
00060     return c;
00061 }
```

### 10.14.3.3 `get_boost_queue()`

```
boost::compute::command_queue& cl::sycl::detail::opencl_context::get_boost_queue ( ) [inline],
[override], [virtual]
```

Return the queue that is associated to the context.

Implements `cl::sycl::detail::context`.

Definition at line 65 of file `opencl_context.hpp`.

References `q`.

```
00065                                     {
00066     return q;
00067 }
```

#### 10.14.3.4 `get_devices()`

```
vector_class<cl::sycl::device> cl::sycl::detail::opengl_context::get_devices ( ) const [inline],
[override], [virtual]
```

Returns the set of devices that are part of this context.

**Todo** To be implemented

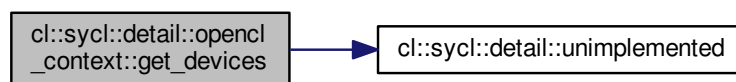
Implements [cl::sycl::detail::context](#).

Definition at line 106 of file [opengl\\_context.hpp](#).

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

```
00106                                     {
00107     detail::unimplemented();
00108     return {};
00109 }
```

Here is the call graph for this function:



#### 10.14.3.5 `get_platform()`

```
cl::sycl::platform cl::sycl::detail::opengl_context::get_platform ( ) const [inline], [override],
[virtual]
```

Return the platform of the context.

Return synchronous errors via the SYCL exception class.

**Todo** To be implemented

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

Definition at line 96 of file [opengl\\_context.hpp](#).

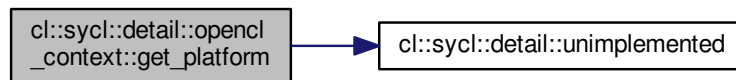
References [cl::sycl::detail::unimplemented\(\)](#).

```

00096                                     {
00097     detail::unimplemented();
00098     return {};
00099 }

```

Here is the call graph for this function:



#### 10.14.3.6 instance()

```

static std::shared_ptr<opengl_context> cl::sycl::detail::opengl_context::instance (
    const boost::compute::context & c ) [inline], [static]

```

Get a singleton instance of the `opengl_context`.

Definition at line 113 of file `opengl_context.hpp`.

References `cl::sycl::detail::cache< Key, Value >::get_or_register()`.

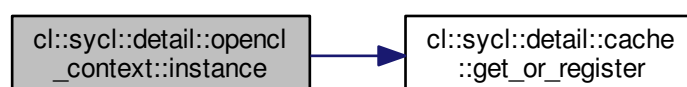
Referenced by `cl::sycl::context::context()`.

```

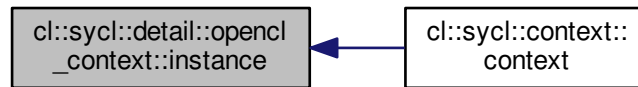
00113                                     {
00114     return cache.get_or_register(c.get(),
00115                                [&] { return new opengl_context {
00116         c }; });
00116 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.14.3.7 `is_host()`

```
bool cl::sycl::detail::openccl_context::is_host ( ) const [inline], [override], [virtual]
```

Return false because the context is not a SYCL host context.

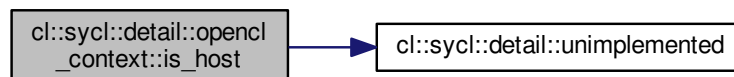
Implements [cl::sycl::detail::context](#).

Definition at line 71 of file [openccl\\_context.hpp](#).

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

```
00071                                     {
00072     return false;
00073 }
```

Here is the call graph for this function:



#### 10.14.4 Member Data Documentation

#### 10.14.4.1 `c`

```
boost::compute::context cl::sycl::detail::opencl_context::c [private]
```

User the Boost Compute abstraction of the OpenCL context.

Definition at line 33 of file [opencl\\_context.hpp](#).

Referenced by [get\\_boost\\_compute\(\)](#), and [opencl\\_context\(\)](#).

#### 10.14.4.2 `cache`

```
detail::cache<cl_context, detail::opencl_context> cl::sycl::detail::opencl_context::cache  
[static], [private]
```

A cache to always return the same alive context for a given OpenCL context.

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

Definition at line 46 of file [opencl\\_context.hpp](#).

Referenced by [~opencl\\_context\(\)](#).

#### 10.14.4.3 `q`

```
boost::compute::command_queue cl::sycl::detail::opencl_context::q [private]
```

A boost `command_queue` associated to an OpenCL context for when we need to transfer data but no queue is given (eg.

When an buffer accessor is created)

Definition at line 39 of file [opencl\\_context.hpp](#).

Referenced by [get\\_boost\\_queue\(\)](#).

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

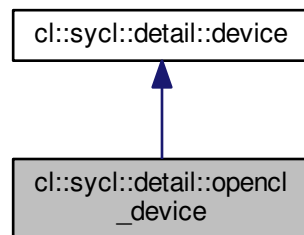
- [include/CL/sycl/context/detail/opencl\\_context.hpp](#)

## 10.15 cl::sycl::detail::opencil\_device Class Reference

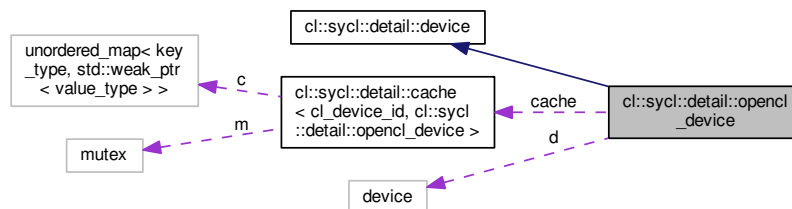
SYCL OpenCL device.

```
#include <opencil_device.hpp>
```

Inheritance diagram for cl::sycl::detail::opencil\_device:



Collaboration diagram for cl::sycl::detail::opencil\_device:



### Public Member Functions

- `cl_device_id` [get](#) () const override  
*Return the cl\_device\_id of the underlying OpenCL device.*
- `boost::compute::device` & [get\\_boost\\_compute](#) () override  
*Return the underlying Boost.Compute 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 build 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.*

### 10.15.1 Detailed Description

SYCL OpenCL device.

Definition at line 30 of file `opengl_device.hpp`.

### 10.15.2 Constructor & Destructor Documentation

#### 10.15.2.1 `opengl_device()`

```
cl::sycl::detail::opengl_device::opengl_device (
    const boost::compute::device & d ) [inline], [private]
```

Only the instance factory can build it.

Definition at line 126 of file `opengl_device.hpp`.

```
00126 : d { d } {}
```

### 10.15.2.2 ~opengl\_device()

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

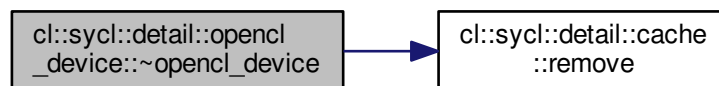
Unregister from the cache on destruction.

Definition at line 131 of file [opengl\\_device.hpp](#).

References [cache](#), [cl::sycl::detail::cache< Key, Value >::remove\(\)](#), [TRISYCL\\_WEAK\\_ATTRIB\\_PREFIX](#), and [cl::sycl::detail::TRISYCL\\_WEAK\\_ATTRIB\\_SUFFIX](#).

```
00131                                     {
00132     cache.remove(d.id());
00133 }
```

Here is the call graph for this function:



## 10.15.3 Member Function Documentation

### 10.15.3.1 get()

```
cl_device_id cl::sycl::detail::opengl_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 [opengl\\_device.hpp](#).

```
00045                                     {
00046     return d.id();
00047 }
```



### 10.15.3.2 get\_boost\_compute()

```
boost::compute::device& cl::sycl::detail::opencl_device::get_boost_compute ( ) [inline],  
[override], [virtual]
```

Return the underlying Boost.Compute device.

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

Definition at line 51 of file [opencl\\_device.hpp](#).

References [d](#).

```
00051                                     {  
00052     return d;  
00053 }
```

### 10.15.3.3 get\_platform()

```
cl::sycl::platform cl::sycl::detail::opencl_device::get_platform ( ) const [inline], [override],  
[virtual]
```

Return the platform of device.

Return synchronous errors via the SYCL exception class.

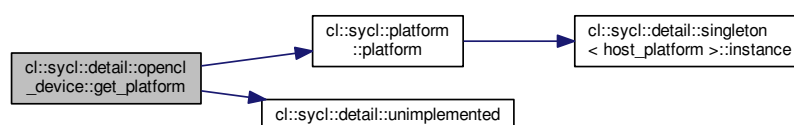
Implements [cl::sycl::detail::device](#).

Definition at line 87 of file [opencl\\_device.hpp](#).

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

```
00087                                     {  
00088     return d.platform();  
00089 }
```

Here is the call graph for this function:



#### 10.15.3.4 has\_extension()

```
bool cl::sycl::detail::opencl_device::has_extension (
    const string_class & extension ) const [inline], [override], [virtual]
```

Specify whether a specific extension is supported on the device.

**Todo** To be implemented

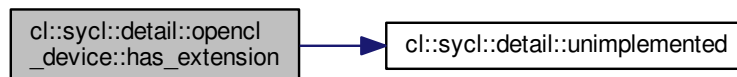
Implements [cl::sycl::detail::device](#).

Definition at line 110 of file [opencl\\_device.hpp](#).

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

```
00110                                     {
00111     detail::unimplemented();
00112     return {};
00113 }
```

Here is the call graph for this function:



#### 10.15.3.5 instance()

```
static std::shared_ptr<opencl_device> cl::sycl::detail::opencl_device::instance (
    const boost::compute::device & d ) [inline], [static]
```

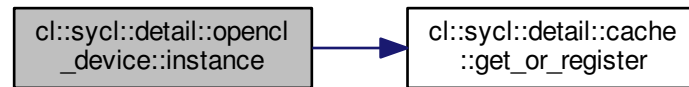
Definition at line 118 of file [opencl\\_device.hpp](#).

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

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

```
00118                                     {
00119     return cache.get_or_register(d.id(),
00120                                [&] { return new opencl_device { d }; });
00121 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.15.3.6 is\_accelerator()

```
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 77 of file [opengl\\_device.hpp](#).

```
00077                                     {
00078     // Even in Boost.Compute the type is a bit-field, so use & instead of ==
00079     return d.type() & boost::compute::device::accelerator;
00080 }
```

### 10.15.3.7 is\_cpu()

```
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 63 of file [opengl\\_device.hpp](#).

```
00063                                     {
00064     // Even in Boost.Compute the type is a bit-field, so use & instead of ==
00065     return d.type() & boost::compute::device::cpu;
00066 }
```

### 10.15.3.8 is\_gpu()

```
bool cl::sycl::detail::opencl_device::is_gpu ( ) const [inline], [override], [virtual]
```

Test if the OpenCL is a GPU device.

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

Definition at line 70 of file [opencl\\_device.hpp](#).

```
00070                                     {
00071     // Even in Boost.Compute the type is a bit-field, so use & instead of ==
00072     return d.type() & boost::compute::device::gpu;
00073 }
```

### 10.15.3.9 is\_host()

```
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 57 of file [opencl\\_device.hpp](#).

```
00057                                     {
00058     return false;
00059 }
```

## 10.15.4 Member Data Documentation

### 10.15.4.1 cache

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

## 10.15.4.2 d

```
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](#).

Referenced by [get\\_boost\\_compute\(\)](#).

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

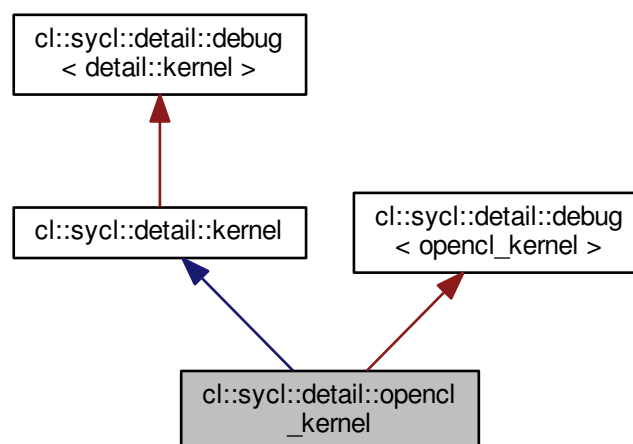
- include/CL/sycl/device/detail/[opencl\\_device.hpp](#)

## 10.16 cl::sycl::detail::opencl\_kernel Class Reference

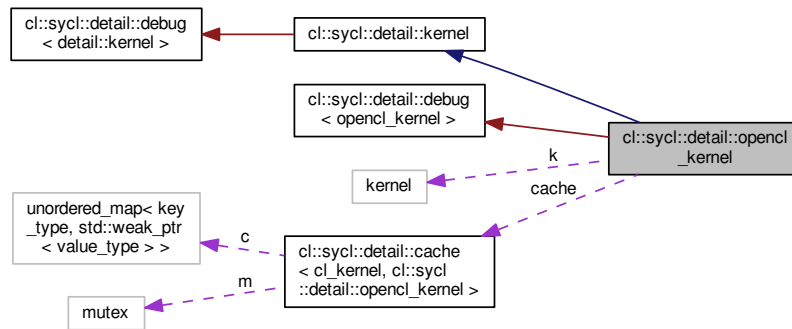
An abstraction of the OpenCL kernel.

```
#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.*
- void [single\\_task](#) (std::shared\_ptr< [detail::task](#) > [task](#), std::shared\_ptr< [detail::queue](#) > q) override  
*Launch a single task of the OpenCL 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< [opencl\\_kernel](#) > [instance](#) (const boost::compute::kernel &k)

## Private Member Functions

- [opencl\\_kernel](#) (const boost::compute::kernel &k)

## Private Attributes

- boost::compute::kernel [k](#)  
*Use the Boost Compute abstraction of the OpenCL kernel.*

## Static Private Attributes

- static [detail::cache](#)< cl\_kernel, [detail::opencl\\_kernel](#) > [cache](#)  
*A cache to always return the same alive kernel for a given OpenCL kernel.*

### 10.16.1 Detailed Description

An abstraction of the OpenCL kernel.

Definition at line 29 of file [opengl\\_kernel.hpp](#).

### 10.16.2 Constructor & Destructor Documentation

#### 10.16.2.1 opengl\_kernel()

```
cl::sycl::detail::opengl_kernel::opengl_kernel (
    const boost::compute::kernel & k ) [inline], [private]
```

Definition at line 42 of file [opengl\\_kernel.hpp](#).

```
00042 : k { k } {}
```

### 10.16.3 Member Function Documentation

#### 10.16.3.1 get()

```
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 58 of file [opengl\\_kernel.hpp](#).

```
00058                                     {
00059     /// \todo Test error and throw. Externalize this feature in Boost.Compute?
00060     clRetainKernel(k);
00061     return k.get();
00062 }
```

### 10.16.3.2 `get_boost_compute()`

```
boost::compute::kernel cl::sycl::detail::opencl_kernel::get_boost_compute ( ) const [inline],
[override], [virtual]
```

Return the Boost.Compute OpenCL kernel object for this kernel.

This is an extension.

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

Definition at line 69 of file [opencl\\_kernel.hpp](#).

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

```
00069                                     {
00070         return k;
00071     }
```

Here is the call graph for this function:



### 10.16.3.3 `instance()`

```
static std::shared_ptr<opencl_kernel> cl::sycl::detail::opencl_kernel::instance (
    const boost::compute::kernel & k ) [inline], [static]
```

Definition at line 48 of file [opencl\\_kernel.hpp](#).

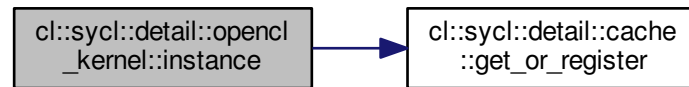
References [cl::sycl::detail::cache< Key, Value >::get\\_or\\_register\(\)](#).

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

```
00048                                     {
00049         return cache.get_or_register(k.get(),
00050                                     [&] { return new opencl_kernel { k }; });
00051     }
```



Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.16.3.4 single\_task()

```

void cl::sycl::detail::opengl_kernel::single_task (
    std::shared_ptr< detail::task > task,
    std::shared_ptr< detail::queue > q ) [inline], [override], [virtual]
  
```

Launch a single task of the OpenCL kernel.

**Todo** Remove either task or q

Implements `cl::sycl::detail::kernel`.

Definition at line 91 of file `opengl_kernel.hpp`.

```

00092                                     {
00093     q->get_boost_compute().enqueue_task(k);
00094     /* For now use a crude synchronization mechanism to map directly a
00095        host task to an accelerator task */
00096     q->get_boost_compute().finish();
00097 }
  
```

### 10.16.3.5 TRISYCL\_ParallelForKernel\_RANGE()

```
cl::sycl::detail::opengl_kernel::TRISYCL_ParallelForKernel_RANGE (
    1 ) [inline], [override]
```

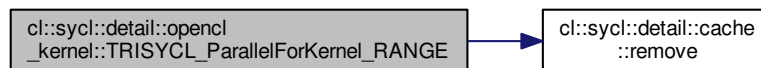
Unregister from the cache on destruction.

Definition at line 126 of file [opengl\\_kernel.hpp](#).

References [cache](#), [cl::sycl::detail::cache< Key, Value >::remove\(\)](#), [TRISYCL\\_WEAK\\_ATTRIB\\_PREFIX](#), and [cl::sycl::detail::TRISYCL\\_WEAK\\_ATTRIB\\_SUFFIX](#).

```
00133         {
00134         cache.remove(k.get());
00135     }
```

Here is the call graph for this function:



## 10.16.4 Member Data Documentation

### 10.16.4.1 cache

```
detail::cache<cl_kernel, detail::opengl\_kernel> cl::sycl::detail::opengl_kernel::cache [static],
[private]
```

A cache to always return the same alive kernel for a given OpenCL kernel.

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

Definition at line 40 of file [opengl\\_kernel.hpp](#).

Referenced by [TRISYCL\\_ParallelForKernel\\_RANGE\(\)](#).

## 10.16.4.2 k

```
boost::compute::kernel cl::sycl::detail::opencl_kernel::k [private]
```

Use the Boost Compute abstraction of the OpenCL kernel.

Definition at line 33 of file [opencl\\_kernel.hpp](#).

Referenced by [get\\_boost\\_compute\(\)](#).

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

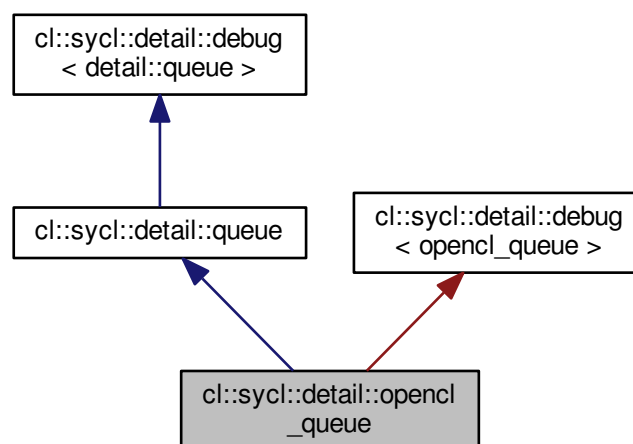
- include/CL/sycl/kernel/detail/[opencl\\_kernel.hpp](#)

## 10.17 cl::sycl::detail::opencl\_queue Class Reference

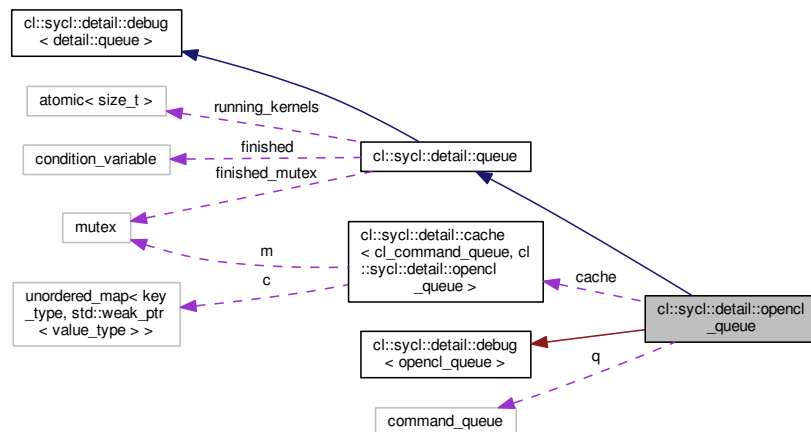
Some implementation details about the SYCL queue.

```
#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)`  
*Get a singleton instance of the `opencl_queue`.*
- static `std::shared_ptr< detail::queue > instance (const cl::sycl::device &d)`  
*Create a new queue associated to this device.*

## 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::opengl\\_queue](#) > [cache](#)  
A cache to always return the same alive queue for a given OpenCL command queue.

## Additional Inherited Members

### 10.17.1 Detailed Description

Some implementation details about the SYCL queue.

Definition at line 23 of file [opengl\\_queue.hpp](#).

### 10.17.2 Constructor & Destructor Documentation

#### 10.17.2.1 opengl\_queue()

```
cl::sycl::detail::opengl_queue::opengl_queue (
    const boost::compute::command_queue & q ) [inline], [private]
```

Only the instance factory can built it.

Definition at line 67 of file [opengl\\_queue.hpp](#).

```
00067 : q { q } {}
```

#### 10.17.2.2 ~opengl\_queue()

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

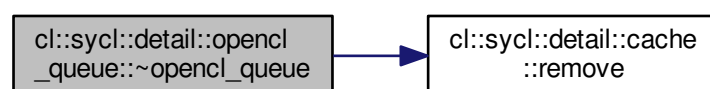
Unregister from the cache on destruction.

Definition at line 96 of file [opengl\\_queue.hpp](#).

References [cache](#), [cl::sycl::detail::cache< Key, Value >::remove\(\)](#), [TRISYCL\\_WEAK\\_ATTRIB\\_PREFIX](#), and [cl::sycl::detail::TRISYCL\\_WEAK\\_ATTRIB\\_SUFFIX](#).

```
00096 {
00097     cache.remove(q.get());
00098 }
```

Here is the call graph for this function:



### 10.17.3 Member Function Documentation

#### 10.17.3.1 get()

```
cl_command_queue cl::sycl::detail::opencl_queue::get ( ) const [inline], [override], [private],  
[virtual]
```

Return the `cl_command_queue` of the underlying OpenCL queue.

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

Definition at line 36 of file [opencl\\_queue.hpp](#).

```
00036                                     {  
00037     return q.get();  
00038 }
```

#### 10.17.3.2 get\_boost\_compute()

```
boost::compute::command_queue& cl::sycl::detail::opencl_queue::get_boost_compute ( ) [inline],  
[override], [private], [virtual]
```

Return the underlying Boost.Compute command queue.

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

Definition at line 42 of file [opencl\\_queue.hpp](#).

References [q](#).

```
00042                                     {  
00043     return q;  
00044 }
```

#### 10.17.3.3 get\_context()

```
cl::sycl::context cl::sycl::detail::opencl_queue::get_context ( ) const [inline], [override],  
[private], [virtual]
```

Return the SYCL context associated to the queue.

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

Definition at line 48 of file [opencl\\_queue.hpp](#).

```
00048                                     {  
00049     return q.get_context();  
00050 }
```

## 10.17.3.4 get\_device()

```
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 54 of file [opencil\\_queue.hpp](#).

```
00054                                     {
00055     return q.get_device();
00056 }
```

## 10.17.3.5 instance() [1/2]

```
static std::shared_ptr<opencil_queue> cl::sycl::detail::opencil_queue::instance (
    const boost::compute::command_queue & q ) [inline], [static]
```

Get a singleton instance of the [opencil\\_queue](#).

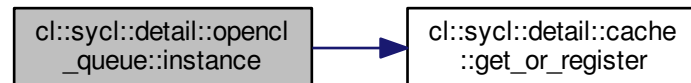
Definition at line 73 of file [opencil\\_queue.hpp](#).

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

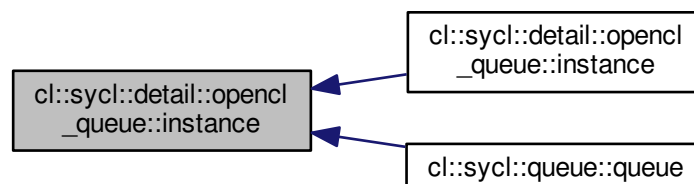
Referenced by [instance\(\)](#), and [cl::sycl::queue::queue\(\)](#).

```
00073                                     {
00074     return cache.get_or_register(q.get(),
00075                                [&] { return new opencil_queue { q }; });
00076 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.17.3.6 instance() [2/2]

```
static std::shared_ptr<detail::queue> cl::sycl::detail::openccl_queue::instance (
    const cl::sycl::device & d ) [inline], [static]
```

Create a new queue associated to this device.

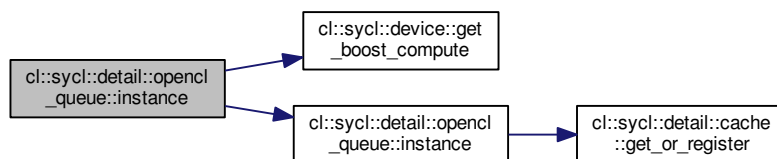
**Todo** Check with SYCL committee what is the expected behaviour here about the context. Is this a new context everytime, or always the same for a given device?

Definition at line 86 of file [openccl\\_queue.hpp](#).

References [cl::sycl::device::get\\_boost\\_compute\(\)](#), and [instance\(\)](#).

```
00086         {
00087         return instance (boost::compute::command_queue {
00088             // For now, create a new context every time
00089             boost::compute::context { d.get_boost_compute() },
00090             d.get_boost_compute()
00091         });
00092     }
```

Here is the call graph for this function:



### 10.17.3.7 is\_host()

```
bool cl::sycl::detail::openccl_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 60 of file [openccl\\_queue.hpp](#).

```
00060         {
00061         return false;
00062     }
```

## 10.17.4 Member Data Documentation



10.17.4.1 `cache`

```
detail::cache<cl_command_queue, detail::opencl_queue> cl::sycl::detail::opencl_queue::cache
[static], [private]
```

A cache to always return the same alive queue for a given OpenCL command queue.

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

Definition at line 33 of file [opencl\\_queue.hpp](#).

Referenced by [~opencl\\_queue\(\)](#).

10.17.4.2 `q`

```
boost::compute::command_queue cl::sycl::detail::opencl_queue::q [private]
```

Use the Boost Compute abstraction of the OpenCL command queue.

Definition at line 26 of file [opencl\\_queue.hpp](#).

Referenced by [get\\_boost\\_compute\(\)](#).

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

- [include/CL/sycl/queue/detail/opencl\\_queue.hpp](#)

10.18 `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>
```

## 10.18.1 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:

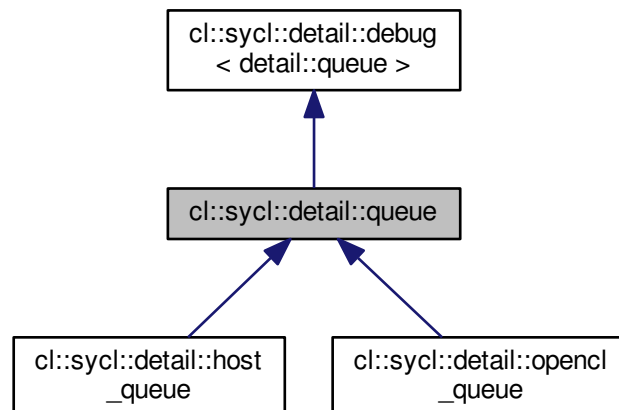
- [include/CL/sycl/info/param\\_traits.hpp](#)

## 10.19 cl::sycl::detail::queue Struct Reference

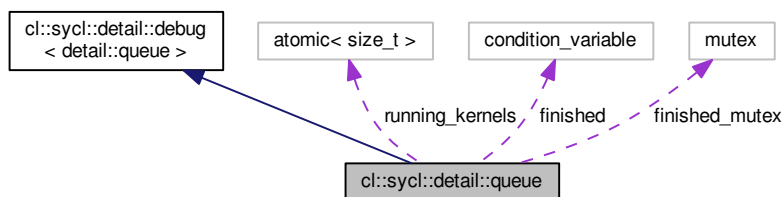
Some implementation details about the SYCL queue.

```
#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.*

### 10.19.1 Detailed Description

Some implementation details about the SYCL queue.

Definition at line 30 of file [queue.hpp](#).

### 10.19.2 Constructor & Destructor Documentation

#### 10.19.2.1 queue()

```
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     }
```

### 10.19.2.2 ~queue()

```
virtual cl::sycl::detail::queue::~~queue ( ) [inline], [virtual]
```

Wait for all kernel completion before the queue destruction.

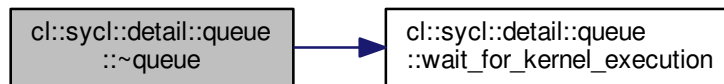
**Todo** Update according spec since queue destruction is non blocking

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

References [wait\\_for\\_kernel\\_execution\(\)](#).

```
00119         {
00120     wait_for_kernel_execution();
00121     }
```

Here is the call graph for this function:



## 10.19.3 Member Function Documentation

### 10.19.3.1 get()

```
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](#).

## 10.19.3.2 get\_boost\_compute()

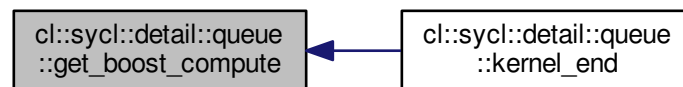
```
virtual boost::compute::command_queue& cl::sycl::detail::queue::get_boost_compute ( ) [pure virtual]
```

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:



## 10.19.3.3 get\_context()

```
virtual cl::sycl::context cl::sycl::detail::queue::get_context ( ) const [pure virtual]
```

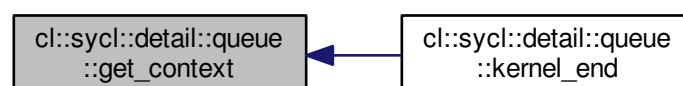
Return the SYCL queue's context.

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:



#### 10.19.3.4 `get_device()`

```
virtual cl::sycl::device cl::sycl::detail::queue::get_device ( ) const [pure virtual]
```

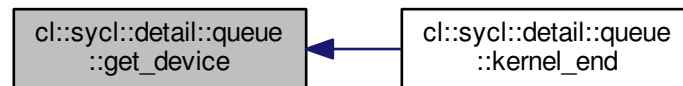
Return the SYCL device the queue is associated with.

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:



#### 10.19.3.5 `is_host()`

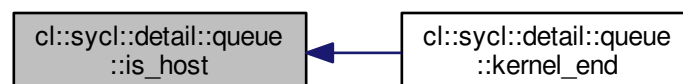
```
virtual bool cl::sycl::detail::queue::is_host ( ) const [pure virtual]
```

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:



## 10.19.3.6 kernel\_end()

```
void cl::sycl::detail::queue::kernel_end ( ) [inline]
```

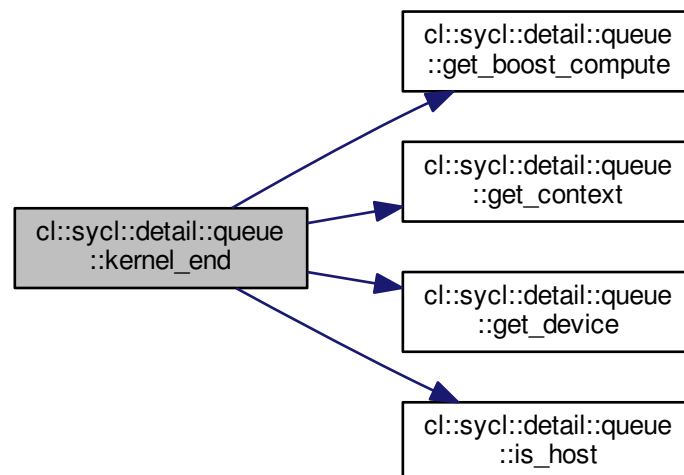
Signal that a new kernel finished on this queue.

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
00072            In some cases several threads might want to wait for the
00073            same queue, because of this \c notify_one is not be enough
00074            and a \c notify_all is needed
00075         */
00076         finished.notify_all();
00077     }
00078 }
```

Here is the call graph for this function:



## 10.19.3.7 kernel\_start()

```
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 }
```

### 10.19.3.8 wait\_for\_kernel\_execution()

```
void cl::sycl::detail::queue::wait_for_kernel_execution ( ) [inline]
```

Wait for all kernel completion.

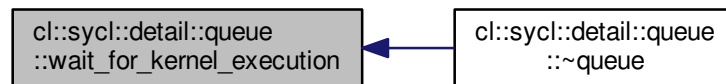
Definition at line 47 of file [queue.hpp](#).

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:



## 10.19.4 Member Data Documentation

### 10.19.4.1 finished

```
std::condition_variable cl::sycl::detail::queue::finished
```

To signal when all the kernels have completed.

Definition at line 35 of file [queue.hpp](#).

### 10.19.4.2 finished\_mutex

```
std::mutex cl::sycl::detail::queue::finished_mutex
```

To protect the access to the condition variable.

Definition at line 37 of file [queue.hpp](#).



10.19.4.3 `running_kernels`

```
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:

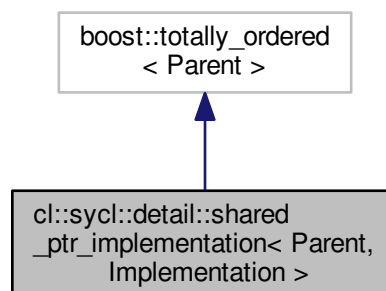
- [include/CL/sycl/queue/detail/queue.hpp](#)

## 10.20 `cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >` Struct Template Reference

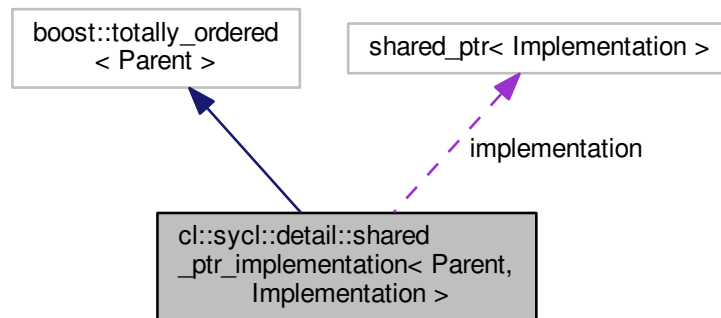
Provide an implementation as `shared_ptr` with total ordering and hashing to be used with algorithms and in (un)ordered containers.

```
#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.*

#### 10.20.1 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](#).

## 10.20.2 Constructor & Destructor Documentation

### 10.20.2.1 shared\_ptr\_implementation() [1/3]

```
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 46 of file [shared\\_ptr\\_implementation.hpp](#).

```
00047 : implementation { i } {}
```

### 10.20.2.2 shared\_ptr\_implementation() [2/3]

```
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 51 of file [shared\\_ptr\\_implementation.hpp](#).

```
00051 : implementation { i } {}
```

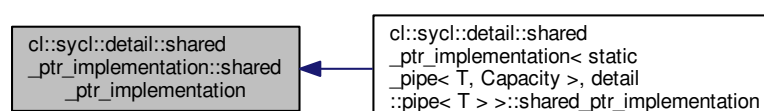
### 10.20.2.3 shared\_ptr\_implementation() [3/3]

```
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:



### 10.20.3 Member Function Documentation

#### 10.20.3.1 hash()

```
template<typename Parent, typename Implementation>
auto cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash ( ) const
[inline]
```

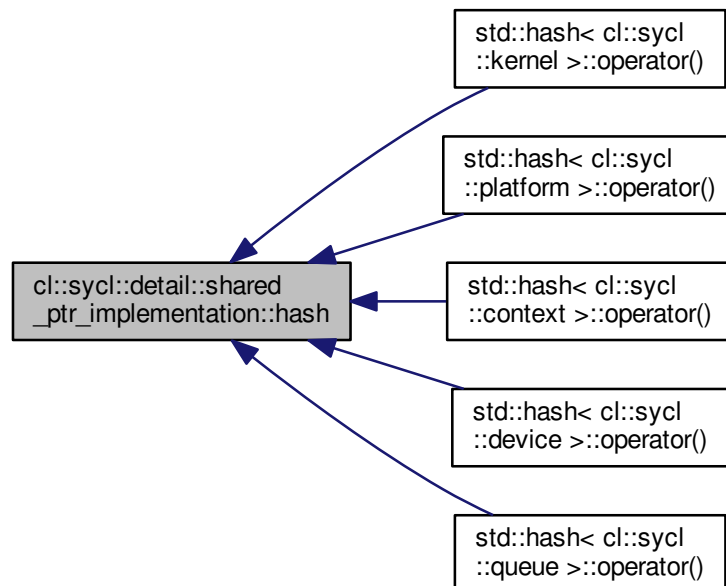
Forward the hashing for unordered containers to the implementation.

Definition at line 83 of file [shared\\_ptr\\_implementation.hpp](#).

Referenced by [std::hash< cl::sycl::kernel >::operator\(\)\(\)](#), [std::hash< cl::sycl::platform >::operator\(\)\(\)](#), [std::hash< cl::sycl::context >::operator\(\)\(\)](#), [std::hash< cl::sycl::device >::operator\(\)\(\)](#), and [std::hash< cl::sycl::queue >::operator\(\)\(\)](#).

```
00083         {
00084     return std::hash<decltype(implementation)>{}(implementation);
00085     }
```

Here is the caller graph for this function:



### 10.20.3.2 `operator<()`

```
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 77 of file `shared_ptr_implementation.hpp`.

```
00077                                     {
00078     return implementation < other.implementation;
00079 }
```

### 10.20.3.3 `operator==()`

```
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 64 of file `shared_ptr_implementation.hpp`.

```
00064                                     {
00065     return implementation == other.implementation;
00066 }
```

## 10.20.4 Member Data Documentation

### 10.20.4.1 `implementation`

```
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()`, and `cl::sycl::handler::single_task()`.

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

- `include/CL/sycl/detail/shared_ptr_implementation.hpp`

## 10.21 `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.*

#### 10.21.1 Detailed Description

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

Provide a singleton factory.

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

#### 10.21.2 Member Function Documentation

##### 10.21.2.1 `instance()`

```
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:

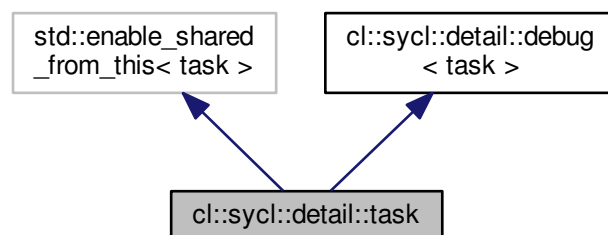
- `include/CL/sycl/detail/singleton.hpp`

## 10.22 cl::sycl::detail::task Struct Reference

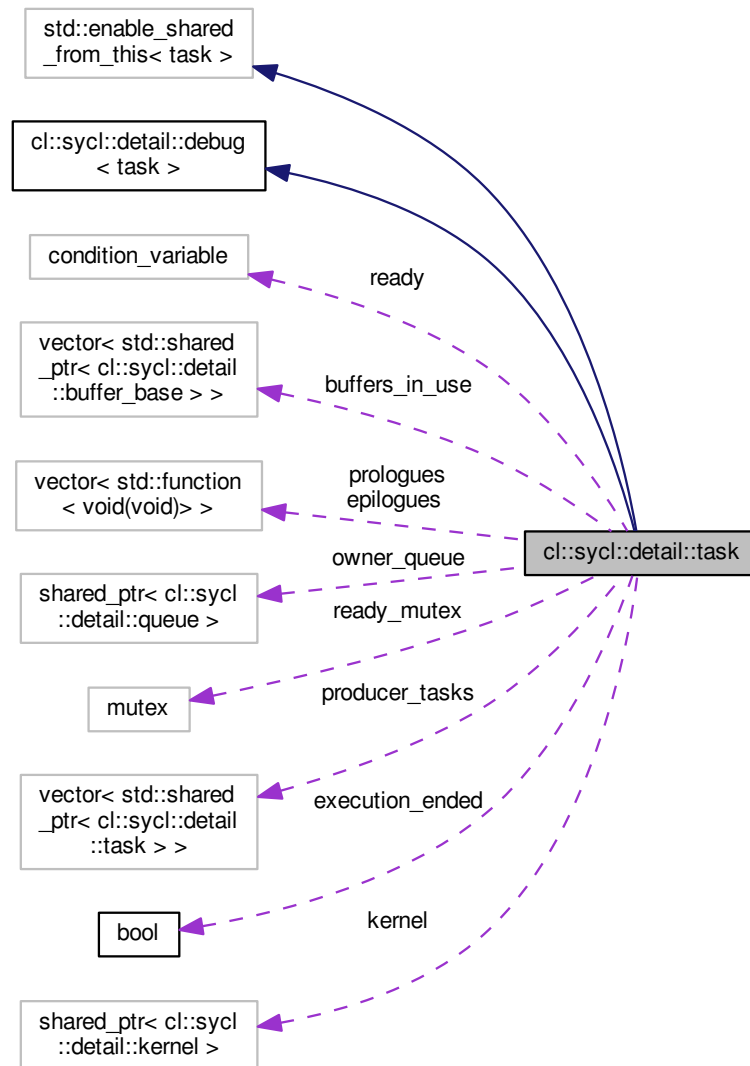
The abstraction to represent SYCL tasks executing inside command\_group.

```
#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`

### 10.22.1 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`.

## 10.22.2 Constructor & Destructor Documentation

### 10.22.2.1 task()

```
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 } {}
```

## 10.22.3 Member Function Documentation

### 10.22.3.1 add\_buffer()

```
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 170 of file [task.hpp](#).

References [TRISYCL\\_DUMP\\_T](#).

```
00171      {
00172          TRISYCL_DUMP_T("Add buffer " << buf << " in task " << this);
00173          /* Keep track of the use of the buffer to notify its release at
00174             the end of the execution */
00175          buffers_in_use.push_back(buf);
00176          // To be sure the buffer does not disappear before the kernel can run
00177          buf->use();
00178
00179          std::shared_ptr<detail::task> latest_producer;
00180          if (is_write_mode) {
00181              /* Set this task as the latest producer of the buffer so that
00182                 another kernel may wait on this task */
00183              latest_producer = buf->set_latest_producer(shared_from_this());
00184          }
00185          else
00186              latest_producer = buf->get_latest_producer();
00187
00188          /* If the buffer is to be produced by a task, add the task in the
00189             producer list to wait on it before running the task core
00190
00191             If a buffer is accessed first in write mode and then in read mode,
00192             the task will add itself as a producer and will wait for itself
00193             when calling \c wait_for_producers, we avoid this by checking that
00194             \c latest_producer is not \c this
00195          */
00196          if (latest_producer && latest_producer != shared_from_this())
00197              producer_tasks.push_back(latest_producer);
00198      }
```

### 10.22.3.2 add\_postlude()

```
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 228 of file [task.hpp](#).

```
00228                                     {
00229     epilogues.push_back(f);
00230 }
```

### 10.22.3.3 add\_prelude()

```
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 222 of file [task.hpp](#).

```
00222                                     {
00223     prologues.push_back(f);
00224 }
```

### 10.22.3.4 get\_kernel()

```
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 249 of file [task.hpp](#).

References [kernel](#).

```
00249                                     {
00250     if (!kernel)
00251         throw non_cl_error("Cannot use an OpenCL kernel in this context");
00252     return *kernel;
00253 }
```

### 10.22.3.5 get\_queue()

```
auto cl::sycl::detail::task::get_queue ( ) [inline]
```

Get the queue behind the task to run a kernel on.

Definition at line 234 of file [task.hpp](#).

References [owner\\_queue](#).

```
00234         {
00235         return owner_queue;
00236     }
```

### 10.22.3.6 notify\_consumers()

```
void cl::sycl::detail::task::notify_consumers ( ) [inline]
```

Notify the waiting tasks that we are done.

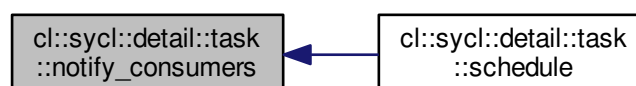
Definition at line 143 of file [task.hpp](#).

References [TRISYCL\\_DUMP\\_T](#).

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

```
00143         {
00144         TRISYCL_DUMP_T("Notify all the task waiting for this task " << this);
00145         {
00146             std::unique_lock<std::mutex> ul { ready_mutex };
00147             execution_ended = true;
00148         }
00149         /* \todo Verify that the memory model with the notify does not
00150            require some fence or atomic */
00151         ready.notify_all();
00152     }
```

Here is the caller graph for this function:



### 10.22.3.7 postlude()

```
void cl::sycl::detail::task::postlude ( ) [inline]
```

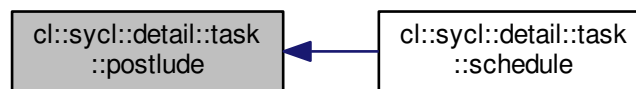
Execute the epilogues.

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

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

```
00212     {  
00213     for (const auto &p : epilogues)  
00214         p();  
00215     /* Free the functors that may own an accessor owning a buffer  
00216        preventing the command group to complete */  
00217     epilogues.clear();  
00218 }
```

Here is the caller graph for this function:



### 10.22.3.8 prelude()

```
void cl::sycl::detail::task::prelude ( ) [inline]
```

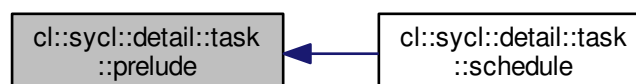
Execute the prologues.

Definition at line 202 of file [task.hpp](#).

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

```
00202     {  
00203     for (const auto &p : prologues)  
00204         p();  
00205     /* Free the functors that may own an accessor owning a buffer  
00206        preventing the command group to complete */  
00207     prologues.clear();  
00208 }
```

Here is the caller graph for this function:



### 10.22.3.9 `release_buffers()`

```
void cl::sycl::detail::task::release_buffers ( ) [inline]
```

Release the buffers that have been used by this task.

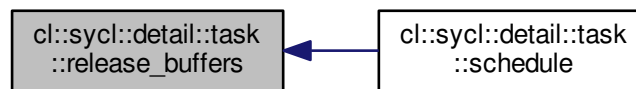
Definition at line 134 of file [task.hpp](#).

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:



### 10.22.3.10 `schedule()`

```
void cl::sycl::detail::task::schedule (
    std::function< void(void)> f ) [inline]
```

Add a new task to the task graph and schedule for execution.

Detach the thread since it will synchronize by its own means

**Todo** This is an issue if there is an exception in the kernel

Definition at line 75 of file [task.hpp](#).

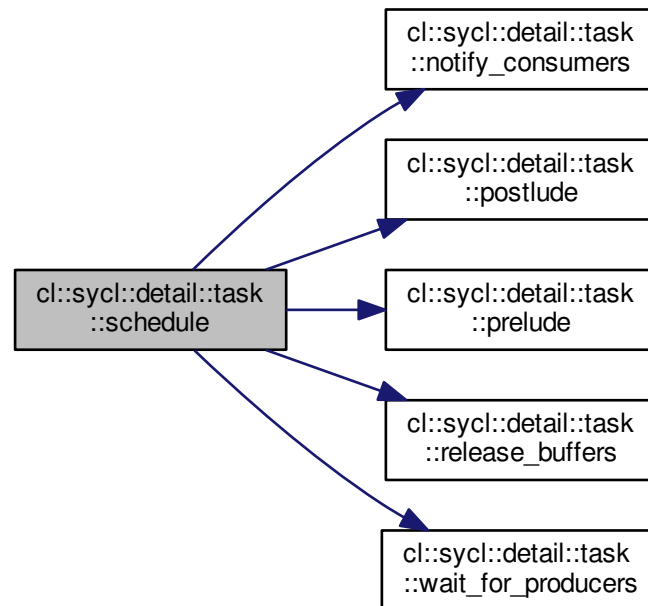
References [notify\\_consumers\(\)](#), [postlude\(\)](#), [prelude\(\)](#), [release\\_buffers\(\)](#), [TRISYCL\\_DUMP\\_T](#), and [wait\\_for\\_↵producers\(\)](#).

```

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         #ifndef TRISYCL_NO_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             \todo This is an issue if there is an exception in the kernel
00113             */
00114             thread.detach();
00115         #else
00116             // Just a synchronous execution otherwise
00117             execution();
00118         #endif
00119     }
00120 }

```

Here is the call graph for this function:



#### 10.22.3.11 set\_kernel()

```
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 240 of file [task.hpp](#).

```
00240                                     {
00241     kernel = k;
00242 }
```

#### 10.22.3.12 wait()

```
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 159 of file [task.hpp](#).

References [execution\\_ended](#), and [TRISYCL\\_DUMP\\_T](#).



```

00159         {
00160             TRISYCL_DUMP_T("The task wait for task " << this << " to end");
00161             std::unique_lock<std::mutex> ul { ready_mutex };
00162             ready.wait(ul, [&] { return execution_ended; });
00163         }

```

### 10.22.3.13 wait\_for\_producers()

```
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](#).

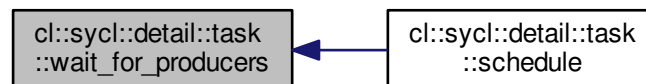
Referenced by [schedule\(\)](#).

```

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:



## 10.22.4 Member Data Documentation

### 10.22.4.1 buffers\_in\_use

```
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](#).

#### 10.22.4.2 epilogues

```
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](#).

#### 10.22.4.3 execution\_ended

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

#### 10.22.4.4 kernel

```
std::shared_ptr<cl::sycl::detail::kernel> cl::sycl::detail::task::kernel
```

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

Referenced by [get\\_kernel\(\)](#).

#### 10.22.4.5 owner\_queue

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

#### 10.22.4.6 producer\_tasks

```
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](#).

#### 10.22.4.7 prologues

```
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](#).

#### 10.22.4.8 ready

```
std::condition_variable cl::sycl::detail::task::ready
```

To signal when this task is ready.

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

#### 10.22.4.9 ready\_mutex

```
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](#)



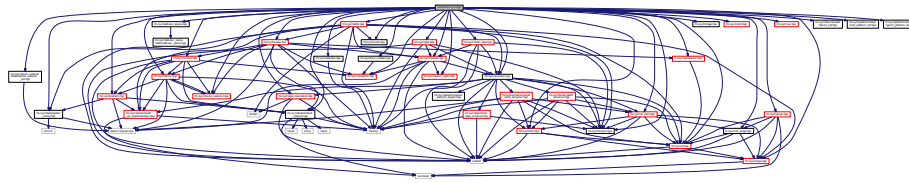
## Chapter 11

# 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/allocator.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/math.hpp"
#include "CL/sycl/nd_item.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/opencv_types.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 "CL/sycl/platform/detail/host_platform_tail.hpp"
#include "CL/sycl/platform/detail/opencv_platform_tail.hpp"
```

Include dependency graph for `sycl.hpp`:



## 11.2 `sycl.hpp`

```

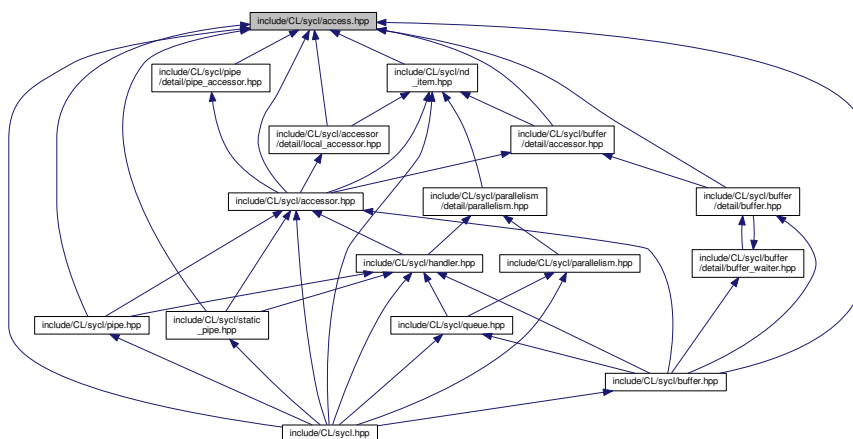
00001 /** \file
00002
00003     \mainpage
00004
00005     This is the main OpenCL SYCL C++ header file to experiment with
00006     the OpenCL CL provisional specification.
00007
00008     For more information about OpenCL SYCL:
00009     http://www.khronos.org/sycl/
00010
00011     For more information on this project and to access to the source of
00012     this file, look at https://github.com/triSYCL/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--2017 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/allocator.hpp"
00038 #include "CL/sycl/allocator.hpp"
00039 #include "CL/sycl/address_space.hpp"
00040 #include "CL/sycl/buffer.hpp"
00041 #include "CL/sycl/context.hpp"
00042 #include "CL/sycl/device.hpp"
00043 #include "CL/sycl/device_selector.hpp"
00044 #include "CL/sycl/error_handler.hpp"
00045 #include "CL/sycl/event.hpp"
00046 #include "CL/sycl/exception.hpp"
00047 #include "CL/sycl/group.hpp"
00048 #include "CL/sycl/handler.hpp"
00049 #include "CL/sycl/id.hpp"
00050 #include "CL/sycl/image.hpp"
00051 #include "CL/sycl/item.hpp"
00052 #include "CL/sycl/math.hpp"
00053 #include "CL/sycl/nd_item.hpp"
00054 #include "CL/sycl/nd_range.hpp"
00055 #include "CL/sycl/opencl_types.hpp"
00056 #include "CL/sycl/parallelism.hpp"
00057 #include "CL/sycl/pipe.hpp"
00058 #include "CL/sycl/pipe_reservation.hpp"
00059 #include "CL/sycl/platform.hpp"
00060 #include "CL/sycl/queue.hpp"
00061 #include "CL/sycl/range.hpp"
00062 #include "CL/sycl/static_pipe.hpp"
00063 #include "CL/sycl/vec.hpp"
00064
00065 // Some includes at the end to break some dependencies

```

```
00066 #include "CL/sycl/device_selector/detail/device_selector_tail.hpp"
00067 "
00067 #include "CL/sycl/device/detail/device_tail.hpp"
00068 #include "CL/sycl/platform/detail/host_platform_tail.hpp"
00069 #ifdef TRISYCL_OPENCL
00070 #include "CL/sycl/platform/detail/opencl_platform_tail.hpp"
00071 #endif
00072
00073 /*
00074      # Some Emacs stuff:
00075      ### Local Variables:
00076      ###   ispell-local-dictionary: "american"
00077      ###   eval: (flyspell-prog-mode)
00078      ### End:
00079 */
```

### 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 /*
00077  # Some Emacs stuff:
00078  ### Local Variables:
00079  ### ispell-local-dictionary: "american"
00080  ### eval: (flyspell-prog-mode)
00081  ### End:
00082 */
00083
00084 #endif // TRISYCL_SYCL_ACCESS_HPP

```

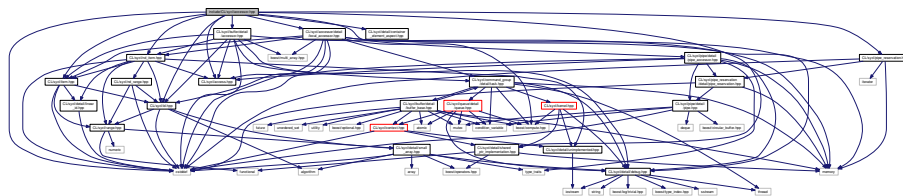
## 11.5 include/CL/sycl/accessor.hpp File Reference

```

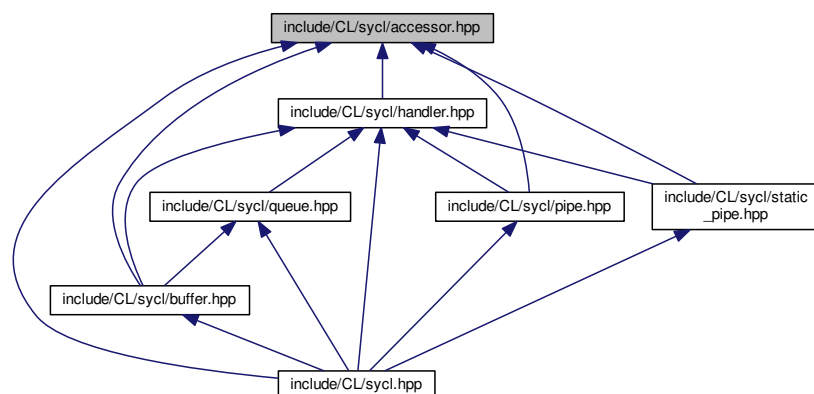
#include <cstddef>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor/detail/local_accessor.hpp"
#include "CL/sycl/buffer/detail/accessor.hpp"
#include "CL/sycl/detail/container_element_aspect.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#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 >`  
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::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/accessor/detail/local_accessor.hpp"
00016 #include "CL/sycl/buffer/detail/accessor.hpp"
00017 #include "CL/sycl/detail/container_element_aspect.hpp"
00018 #include "CL/sycl/detail/shared_ptr_implementation.hpp"
00019 #include "CL/sycl/id.hpp"
00020 #include "CL/sycl/item.hpp"
00021 #include "CL/sycl/nd_item.hpp"
00022 #include "CL/sycl/pipe_reservation.hpp"
00023 #include "CL/sycl/pipe/detail/pipe_accessor.hpp"
00024
00025 namespace cl {
00026 namespace sycl {
00027
00028 template <typename T, int Dimensions, typename Allocator>
00029 class buffer;
00030 template <typename T>
00031 class pipe;
00032 class handler;
00033

```

```

00034 /** \addtogroup data Data access and storage in SYCL
00035     @{
00036 */
00037
00038 /** The accessor abstracts the way buffer or pipe data are accessed
00039     inside a kernel in a multidimensional variable length array way.
00040
00041     \todo Implement it for images according so section 3.3.4.5
00042 */
00043 template <typename DataType,
00044           int Dimensions,
00045           access::mode AccessMode,
00046           access::target Target = access::target::global_buffer>
00047 class accessor :
00048     public detail::shared_ptr_implementation<accessor<DataType,
00049                                           Dimensions,
00050                                           AccessMode,
00051                                           Target>,
00052                                           detail::accessor<DataType,
00053                                           Dimensions,
00054                                           AccessMode,
00055                                           Target>>,
00056     public detail::container_element_aspect<DataType> {
00057 public:
00058
00059     /// \todo in the specification: store the dimension for user request
00061     static constexpr auto dimensionality = Dimensions;
00062 private:
00063
00064     using accessor_detail = typename detail::accessor<DataType,
00065                                                       Dimensions,
00066                                                       AccessMode,
00067                                                       Target>;
00069
00070     // The type encapsulating the implementation
00071     using implementation_t = typename
00072     accessor::shared_ptr_implementation;
00073
00074     // Allows the comparison operation to access the implementation
00075     friend implementation_t;
00076 public:
00077
00078     // Make the implementation member directly accessible in this class
00079     using implementation_t::implementation;
00080
00081     /** Construct a buffer accessor from a buffer using a command group
00082         handler object from the command group scope
00083
00084         Constructor only available for global_buffer or constant_buffer
00085         target.
00086
00087         access_target defines the form of access being obtained.
00088
00089         \todo Add template allocator type in all the accessor
00090         constructors in the specification or just use a more opaque
00091         Buffer type?
00092
00093         \todo fix specification where access mode should be target
00094         instead
00095     */
00096     template <typename Allocator>
00097     accessor(buffer<DataType, Dimensions, Allocator> &
00098             target_buffer,
00099             handler &command_group_handler) : implementation_t {
00100         new detail::accessor<DataType, Dimensions, AccessMode, Target>
00101         {
00102             target_buffer.implementation->implementation, command_group_handler }
00103     } {
00104         static_assert(Target == access::target::global_buffer
00105             || Target == access::target::constant_buffer,
00106             "access target should be global_buffer or constant_buffer "
00107             "when a handler is used");
00108         // Now the implementation is created, register it
00109         implementation->register_accessor();
00110     }
00111
00112     /** Construct a buffer accessor from a buffer
00113
00114         Constructor only available for host_buffer target.
00115
00116         access_target defines the form of access being obtained.
00117     */
00118     template <typename Allocator>

```

```

00118     accessor(buffer<DataType, Dimensions, Allocator> &
target_buffer)
00119     : implementation_t {
00120     new detail::accessor<DataType, Dimensions, AccessMode, Target>
{
00121         target_buffer.implementation->implementation }
00122     } {
00123         static_assert(Target == access::target::host_buffer,
00124             "without a handler, access target should be host_buffer");
00125     }
00126
00127
00128     /** Construct a buffer accessor from a buffer given a specific range for
00129     access permissions and an offset that provides the starting point
00130     for the access range using a command group handler object from the
00131     command group scope
00132
00133     This accessor limits the processing of the buffer to the [offset,
00134     offset+range[ for every dimension. Any other parts of the buffer
00135     will be unaffected.
00136
00137     Constructor only available for access modes global_buffer, and
00138     constant_buffer (see Table "Buffer accessor constructors").
00139     access_target defines the form of access being obtained.
00140
00141     This accessor is recommended for discard-write and discard read
00142     write access modes, when the unaffected parts of the processing
00143     should be retained.
00144     */
00145     template <typename Allocator>
00146     accessor(buffer<DataType, Dimensions, Allocator> &
target_buffer,
00147         handler &command_group_handler,
00148         const range<Dimensions> &offset,
00149         const range<Dimensions> &range) {
00150         detail::unimplemented();
00151     }
00152
00153
00154     /** Construct an accessor of dimension Dimensions with elements of type
00155     DataType using the passed range to specify the size in each
00156     dimension
00157
00158     It needs as a parameter a command group handler object from the
00159     command group scope. Constructor only available if AccessMode is
00160     local, see Table 3.25.
00161     */
00162     accessor(const range<Dimensions> &allocation_size,
00163         handler &command_group_handler)
00164         : implementation_t { new detail::accessor<DataType,
00165             Dimensions,
00166             AccessMode,
00167             access::target::local> {
00168             allocation_size, command_group_handler
00169         }
00170     }
00171     {
00172         static_assert(Target == access::target::local,
00173             "This accessor constructor requires "
00174             "access target be local");
00175     }
00176
00177
00178     /** Return a range object representing the size of the buffer in
00179     terms of number of elements in each dimension as passed to the
00180     constructor
00181
00182     \todo Move on
00183     https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
00184     https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
00185     */
00186     auto get_range() const {
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 implementation->get_range();
00195         */
00196     }
00197
00198     /** Returns the total number of elements behind the accessor
00199
00200     Equal to get_range()[0] * ... * get_range()[Dimensions-1].
00201

```

```

00202     \todo Move on
00203     https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
00204     https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
00205 */
00206 auto get_count() const {
00207     return implementation->get_count();
00208 }
00209
00210
00211 /** Returns the size of the underlying buffer storage in bytes
00212
00213     \todo It is incompatible with buffer get_size() in the spec
00214
00215     \todo Move on
00216     https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
00217     https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
00218 */
00219 auto get_size() const {
00220     return implementation->get_size();
00221 }
00222
00223
00224 /** Use the accessor with integers à la [][][]
00225
00226     Use array_view_type::reference instead of auto& because it does not
00227     work in some dimensions.
00228 */
00229 typename accessor_detail::reference operator[](std::size_t index) {
00230     return (*implementation)[index];
00231 }
00232
00233
00234 /** Use the accessor with integers à la [][][]
00235
00236     Use array_view_type::reference instead of auto& because it does not
00237     work in some dimensions.
00238 */
00239 typename accessor_detail::reference operator[](std::size_t index) const {
00240     return (*implementation)[index];
00241 }
00242
00243
00244 /// To use the accessor with [id<>]
00245 auto &operator[](id<dimensionality> index) {
00246     return (*implementation)[index];
00247 }
00248
00249
00250 /// To use the accessor with [id<>]
00251 auto &operator[](id<dimensionality> index) const {
00252     return (*implementation)[index];
00253 }
00254
00255
00256 /// To use an accessor with [item<>]
00257 auto &operator[](item<dimensionality> index) {
00258     return (*this)[index.get()];
00259 }
00260
00261
00262 /// To use an accessor with [item<>]
00263 auto &operator[](item<dimensionality> index) const {
00264     return (*this)[index.get()];
00265 }
00266
00267
00268 /** To use an accessor with an [nd_item<>]
00269
00270     \todo Add in the specification because used by HPC-GPU slide 22
00271 */
00272 auto &operator[](nd_item<dimensionality> index) {
00273     return (*this)[index.get_global()];
00274 }
00275
00276
00277 /** To use an accessor with an [nd_item<>]
00278
00279     \todo Add in the specification because used by HPC-GPU slide 22
00280 */
00281 auto &operator[](nd_item<dimensionality> index) const {
00282     return (*this)[index.get_global()];
00283 }
00284
00285
00286 /** Get the first element of the accessor
00287
00288     Useful with an accessor on a scalar for example.
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000

```

```

00289     \todo Add in the specification
00290 */
00291 typename accessor_detail::reference operator*() {
00292     return **implementation;
00293 }
00294
00295
00296 /** Get the first element of the accessor
00297
00298     Useful with an accessor on a scalar for example.
00299
00300     \todo Add in the specification?
00301
00302     \todo Add the concept of 0-dim buffer and accessor for scalar
00303     and use an implicit conversion to value_type reference to access
00304     the value with the accessor?
00305 */
00306 typename accessor_detail::reference operator*() const {
00307     return **implementation;
00308 }
00309
00310
00311 /** Get the pointer to the start of the data
00312
00313     \todo Should it be named data() instead? */
00314 auto
00315 get_pointer() const {
00316     return implementation->get_pointer();
00317 }
00318
00319
00320 /** Forward all the iterator functions to the implementation
00321
00322     \todo Add these functions to the specification
00323
00324     \todo The fact that the lambda capture make a const copy of the
00325     accessor is not yet elegantly managed... The issue is that
00326     begin()/end() dispatch is made according to the accessor
00327     constness and not from the array member constness...
00328
00329     \todo try to solve it by using some enable_if on array
00330     constness?
00331
00332     \todo The issue is that the end may not be known if it is
00333     implemented by a raw OpenCL cl_mem... So only provide on the
00334     device the iterators related to the start? Actually the accessor
00335     needs to know a part of the shape to have the multidimensional
00336     addressing. So this only require a size_t more...
00337
00338     \todo Factor out these in a template helper
00339 */
00340
00341 // iterator begin() { return array.begin(); }
00342 typename accessor_detail::iterator begin() const {
00343     return implementation->begin();
00344 }
00345
00346
00347 // iterator end() { return array.end(); }
00348 typename accessor_detail::iterator end() const {
00349     return implementation->end();
00350 }
00351
00352
00353 // const_iterator begin() const { return implementation->begin(); }
00354
00355
00356 // const_iterator end() const { return implementation->end(); }
00357
00358
00359 typename accessor_detail::const_iterator cbegin() const {
00360     return implementation->cbegin();
00361 }
00362
00363
00364
00365 typename accessor_detail::const_iterator cend() const {
00366     return implementation->cend();
00367 }
00368
00369
00370 typename accessor_detail::reverse_iterator rbegin() const {
00371     return implementation->rbegin();
00372 };
00373
00374
00375 typename accessor_detail::reverse_iterator rend() const {

```

```

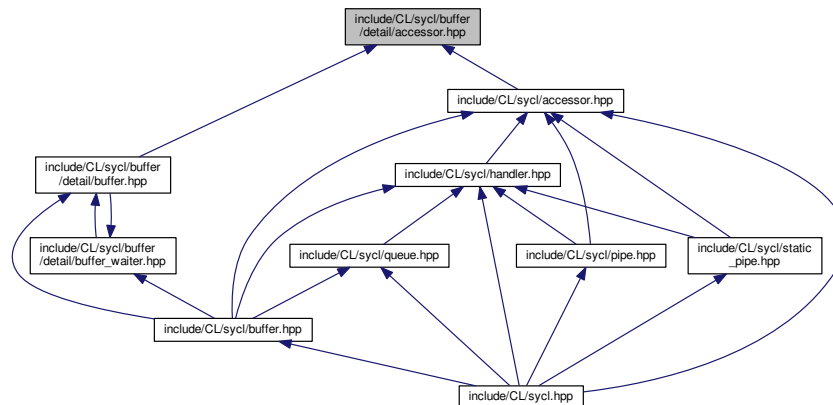
00376     return implementation->rend();
00377 }
00378
00379 // const_reverse_iterator rbegin() const { return array.rbegin(); }
00380
00381 // const_reverse_iterator rend() const { return array.rend(); }
00382
00383 typename accessor_detail::const_reverse_iterator crbegin() const {
00384     return implementation->rbegin();
00385 }
00386
00387 typename accessor_detail::const_reverse_iterator crend() const {
00388     return implementation->rend();
00389 }
00390
00391 };
00392
00393 /** The pipe accessor abstracts the way pipe data are accessed inside
00394     a kernel
00395
00396     A specialization for an non-blocking pipe
00397 */
00398 template <typename DataType,
00399           access::mode AccessMode>
00400 class accessor<DataType, 1, AccessMode, access::target::pipe> :
00401     public detail::pipe_accessor<DataType, AccessMode, access::target::pipe> {
00402 public:
00403     using accessor_detail =
00404         detail::pipe_accessor<DataType, AccessMode, access::target::pipe>
00405 ;
00406 // Inherit of the constructors to have accessor constructor from detail
00407 using accessor_detail::accessor_detail;
00408
00409 /** Construct a pipe accessor from a pipe using a command group
00410     handler object from the command group scope
00411
00412     access_target defines the form of access being obtained.
00413 */
00414 accessor(pipe<DataType> &p, handler &command_group_handler)
00415 : accessor_detail { p.implementation, command_group_handler } { }
00416
00417 /// Make a reservation inside the pipe
00418 pipe_reservation<accessor> reserve(std::size_t size) const {
00419     return accessor_detail::reserve(size);
00420 }
00421
00422 /// Get the underlying pipe implementation
00423 auto &get_pipe_detail() {
00424     return accessor_detail::get_pipe_detail();
00425 }
00426
00427 };
00428
00429 /** The pipe accessor abstracts the way pipe data are accessed inside
00430     a kernel
00431
00432     A specialization for a blocking pipe
00433 */
00434 template <typename DataType,
00435           access::mode AccessMode>
00436 class accessor<DataType, 1, AccessMode, access::target::blocking_pipe> :
00437     public detail::pipe_accessor<DataType, AccessMode, access::target::blocking_pipe>
00438 {
00439 public:
00440     using accessor_detail =
00441         detail::pipe_accessor<DataType, AccessMode, access::target::blocking_pipe>
00442 ;
00443 // Inherit of the constructors to have accessor constructor from detail
00444 using accessor_detail::accessor_detail;
00445
00446 /** Construct a pipe accessor from a pipe using a command group
00447     handler object from the command group scope
00448
00449     access_target defines the form of access being obtained.
00450 */
00451 accessor(pipe<DataType> &p, handler &command_group_handler)
00452 : accessor_detail { p.implementation, command_group_handler } { }
00453
00454 }

```

## 11.7 include/CL/sycl/buffer/detail/accessor.hpp File Reference



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 #include <boost/multi_array.hpp>
00016
00017 #include "CL/sycl/access.hpp"
00018 #include "CL/sycl/command_group/detail/task.hpp"
00019 #include "CL/sycl/detail/debug.hpp"
00020 #include "CL/sycl/id.hpp"
00021 #include "CL/sycl/item.hpp"
00022 #include "CL/sycl/nd_item.hpp"
```

```

00023
00024 namespace cl {
00025 namespace sycl {
00026
00027 class handler;
00028
00029 namespace detail {
00030
00031 // Forward declaration of detail::buffer for use in accessor
00032 template <typename T, int Dimensions> class buffer;
00033
00034 /** \addtogroup data Data access and storage in SYCL
00035     @{
00036     */
00037
00038 /** The buffer accessor abstracts the way buffer data are accessed
00039     inside a kernel in a multidimensional variable length array way.
00040
00041     This implementation relies on boost::multi_array to provide this
00042     nice syntax and behaviour.
00043
00044     Right now the aim of this class is just to access to the buffer in
00045     a read-write mode, even if capturing the multi_array_ref from a
00046     lambda make it const (since in examples we have lambda with [=]
00047     without mutable lambda).
00048
00049     \todo Use the access::mode
00050     */
00051 template <typename T,
00052           int Dimensions,
00053           access::mode Mode,
00054           access::target Target /* = access::global_buffer */>
00055 class accessor :
00056     public std::enable_shared_from_this<accessor<T,
00057                                           Dimensions,
00058                                           Mode,
00059                                           Target>>,
00060     public detail::debug<accessor<T,
00061                               Dimensions,
00062                               Mode,
00063                               Target>> {
00064     /** Keep a reference to the accessed buffer
00065
00066         Beware that it owns the buffer, which means that the accessor
00067         has to be destroyed to release the buffer and potentially
00068         unblock a kernel at the end of its execution
00069     */
00070     std::shared_ptr<detail::buffer<T, Dimensions>> buf;
00071
00072     /// The implementation is a multi_array_ref wrapper
00073     using array_view_type = boost::multi_array_ref<T, Dimensions>;
00074
00075     // The same type but writable
00076     using writable_array_view_type =
00077         typename std::remove_const<array_view_type>::type;
00078
00079     /** The way the buffer is really accessed
00080
00081         Use a mutable member because the accessor needs to be captured
00082         by value in the lambda which is then read-only. This is to avoid
00083         the user to use mutable lambda or have a lot of const_cast as
00084         previously done in this implementation
00085     */
00086     mutable array_view_type array;
00087
00088     /// The task where the accessor is used in
00089     std::shared_ptr<detail::task> task;
00090
00091 public:
00092
00093     /** \todo in the specification: store the dimension for user request
00094
00095         \todo Use another name, such as from C++17 committee discussions.
00096     */
00097     static constexpr auto dimensionality = Dimensions;
00098
00099     /** \todo in the specification: store the types for user request as STL
00100         or C++AMP */
00101     using value_type = T;
00102     using element = T;
00103     using reference = typename array_view_type::reference;
00104     using const_reference = typename array_view_type::const_reference;
00105
00106     /** Inherit the iterator types from the implementation
00107
00108         \todo Add iterators to accessors in the specification
00109     */

```

```

00110     using iterator = typename array_view_type::iterator;
00111     using const_iterator = typename array_view_type::const_iterator;
00112     using reverse_iterator = typename array_view_type::reverse_iterator;
00113     using const_reverse_iterator =
00114         typename array_view_type::const_reverse_iterator;
00115
00116
00117     /** Construct a host accessor from an existing buffer
00118
00119         \todo fix the specification to rename target that shadows
00120         template parm
00121     */
00122     accessor(std::shared_ptr<detail::buffer<T, Dimensions>>
00123         target_buffer) :
00124         buf { target_buffer }, array { target_buffer->access } {
00125         target_buffer->template track_access_mode<Mode>();
00126         TRISYCL_DUMP_T("Create a host accessor write = " <<
00127             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 #ifdef TRISYCL_OPENCL
00135     /** For the host context, we are obligated to update the buffer state
00136         during the accessors creation, otherwise we have no way of knowing
00137         if a buffer was modified on the host. This is only true because
00138         host accessors are blocking
00139     */
00140     cl::sycl::context ctx;
00141     buf->update_buffer_state(ctx, Mode, get_size(), array.data());
00142 #endif
00143 }
00144
00145 /** Construct a device accessor from an existing buffer
00146
00147     \todo fix the specification to rename target that shadows
00148     template parm
00149 */
00150 accessor(std::shared_ptr<detail::buffer<T, Dimensions>>
00151     target_buffer,
00152     handler &command_group_handler) :
00153     buf { target_buffer }, array { target_buffer->access } {
00154     target_buffer->template track_access_mode<Mode>();
00155     TRISYCL_DUMP_T("Create a kernel accessor write = " <<
00156         is_write_access());
00157     static_assert(Target == access::target::global_buffer
00158         || Target == access::target::constant_buffer,
00159         "access target should be global_buffer or constant_buffer "
00160         "when a handler is used");
00161     // Register the buffer to the task dependencies
00162     task = buffer_add_to_task(buf, &command_group_handler,
00163         is_write_access());
00164 }
00165
00166 /** Register the accessor once a \c std::shared_ptr is created on it
00167
00168     This is to be called from outside once the object is created. It
00169     has been tried directly inside the constructor, but calling \c
00170     shared_from_this() from the constructor dead-lock with
00171     libstdc++6
00172
00173     \todo Double-check with the C++ committee on this issue.
00174 */
00175 void register_accessor() {
00176 #ifdef TRISYCL_OPENCL
00177     if (!task->get_queue()->is_host()) {
00178         // To keep alive this accessor in the following lambdas
00179         auto acc = this->shared_from_this();
00180         /* Before running the kernel, make sure the cl_mem behind this
00181             accessor is up-to-date on the device if needed and pass it to
00182             the kernel */
00183         task->add_prelude([=] {
00184             acc->copy_in_cl_buffer();
00185         });
00186         // After running the kernel, deal with some copy-back if needed
00187         task->add_postlude([=] {
00188             /* Even if this function does nothing, it is required to
00189                 have the capture of acc to keep the accessor alive across
00190                 the kernel execution up to the execution postlude */
00191             acc->copy_back_cl_buffer();
00192         });
00193     }
00194 #endif
00195 }

```

```

00192     }
00193
00194
00195     /** Return a range object representing the size of the buffer in
00196         terms of number of elements in each dimension as passed to the
00197         constructor
00198
00199         \todo Move on
00200         https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
00201         https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
00202     */
00203     auto get_range() const {
00204         /** Interpret the shape which is a pointer to the first element as an
00205             array of Dimensions elements so that the range<Dimensions>
00206             constructor is happy with this collection
00207
00208             \todo Add also a constructor in range<> to accept a const
00209             std::size_t *?
00210         */
00211         return range<Dimensions> {
00212             *(const std::size_t *) [Dimensions]] (array.shape())
00213         };
00214     }
00215
00216
00217     /** Returns the total number of elements behind the accessor
00218
00219         Equal to get_range()[0] * ... * get_range()[Dimensions-1].
00220
00221         \todo Move on
00222         https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
00223         https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
00224     */
00225     auto get_count() const {
00226         return array.num_elements();
00227     }
00228
00229
00230     /** Returns the size of the underlying buffer storage in bytes
00231
00232         \todo Move on
00233         https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
00234         https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
00235     */
00236     auto get_size() const {
00237         return get_count()*sizeof(value_type);
00238     }
00239
00240
00241     /** Use the accessor with integers à la [][][]
00242
00243         Use array_view_type::reference instead of auto& because it does not
00244         work in some dimensions.
00245     */
00246     reference operator[](std::size_t index) {
00247         return array[index];
00248     }
00249
00250
00251     /** Use the accessor with integers à la [][][]
00252
00253         Use array_view_type::reference instead of auto& because it does not
00254         work in some dimensions.
00255     */
00256     reference operator[](std::size_t index) const {
00257         return array[index];
00258     }
00259
00260
00261     /// To use the accessor with [id<>]
00262     auto &operator[](id<dimensionality> index) {
00263         return array(index);
00264     }
00265
00266
00267     /// To use the accessor with [id<>]
00268     auto &operator[](id<dimensionality> index) const {
00269         return array(index);
00270     }
00271
00272
00273     /// To use an accessor with [item<>]
00274     auto &operator[](item<dimensionality> index) {
00275         return (*this)[index.get()];
00276     }
00277
00278

```

```

00279  /// To use an accessor with [item<>]
00280  auto &operator[](item<dimensionality> index) const {
00281      return (*this)[index.get()];
00282  }
00283
00284
00285  /** To use an accessor with an [nd_item<>]
00286
00287      \todo Add in the specification because used by HPC-GPU slide 22
00288  */
00289  auto &operator[](nd_item<dimensionality> index) {
00290      return (*this)[index.get_global()];
00291  }
00292
00293  /** To use an accessor with an [nd_item<>]
00294
00295      \todo Add in the specification because used by HPC-GPU slide 22
00296  */
00297  auto &operator[](nd_item<dimensionality> index) const {
00298      return (*this)[index.get_global()];
00299  }
00300
00301
00302  /** Get the first element of the accessor
00303
00304      Useful with an accessor on a scalar for example.
00305
00306      \todo Add in the specification
00307  */
00308  reference operator*() {
00309      return *array.data();
00310  }
00311
00312
00313  /** Get the first element of the accessor
00314
00315      Useful with an accessor on a scalar for example.
00316
00317      \todo Add in the specification?
00318
00319      \todo Add the concept of 0-dim buffer and accessor for scalar
00320      and use an implicit conversion to value_type reference to access
00321      the value with the accessor?
00322  */
00323  reference operator*() const {
00324      return *array.data();
00325  }
00326
00327
00328  /// Get the buffer used to create the accessor
00329  detail::buffer<T, Dimensions> &get_buffer() {
00330      return *buf;
00331  }
00332
00333
00334  /** Test if the accessor has a read access right
00335
00336      \todo Strangely, it is not really constexpr because it is not a
00337      static method...
00338
00339      \todo to move in the access::mode enum class and add to the
00340      specification ?
00341  */
00342  constexpr bool is_read_access() const {
00343      return Mode == access::mode::read
00344          || Mode == access::mode::read_write
00345          || Mode == access::mode::discard_read_write;
00346  }
00347
00348
00349  /** Test if the accessor has a write access right
00350
00351      \todo Strangely, it is not really constexpr because it is not a
00352      static method...
00353
00354      \todo to move in the access::mode enum class and add to the
00355      specification ?
00356  */
00357  constexpr bool is_write_access() const {
00358      return Mode == access::mode::write
00359          || Mode == access::mode::read_write
00360          || Mode == access::mode::discard_write
00361          || Mode == access::mode::discard_read_write;
00362  }
00363
00364
00365  /** Return the pointer to the data

```

```

00366
00367     \todo Implement the various pointer address spaces
00368 */
00369 auto
00370 get_pointer() {
00371     return array.data();
00372 }
00373
00374
00375 /** Forward all the iterator functions to the implementation
00376
00377     \todo Add these functions to the specification
00378
00379     \todo The fact that the lambda capture make a const copy of the
00380     accessor is not yet elegantly managed... The issue is that
00381     begin()/end() dispatch is made according to the accessor
00382     constness and not from the array member constness...
00383
00384     \todo try to solve it by using some enable_if on array
00385     constness?
00386
00387     \todo The issue is that the end may not be known if it is
00388     implemented by a raw OpenCL cl_mem... So only provide on the
00389     device the iterators related to the start? Actually the accessor
00390     needs to know a part of the shape to have the multidimensional
00391     addressing. So this only require a size_t more...
00392
00393     \todo Factor out these in a template helper
00394
00395     \todo Do we need this in detail::accessor too or only in accessor?
00396 */
00397
00398
00399 // iterator begin() { return array.begin(); }
00400 iterator begin() const {
00401     return const_cast<writable_array_view_type &>(array).
begin();
00402 }
00403
00404
00405 // iterator end() { return array.end(); }
00406 iterator end() const {
00407     return const_cast<writable_array_view_type &>(array).
end();
00408 }
00409
00410
00411 // const_iterator begin() const { return array.begin(); }
00412
00413
00414 // const_iterator end() const { return array.end(); }
00415
00416
00417 const_iterator cbegin() const { return array.begin(); }
00418
00419
00420 const_iterator cend() const { return array.end(); }
00421
00422
00423 // reverse_iterator rbegin() { return array.rbegin(); }
00424 reverse_iterator rbegin() const {
00425     return const_cast<writable_array_view_type &>(array).
rbegin();
00426 }
00427
00428
00429 // reverse_iterator rend() { return array.rend(); }
00430 reverse_iterator rend() const {
00431     return const_cast<writable_array_view_type &>(array).
rend();
00432 }
00433
00434
00435 // const_reverse_iterator rbegin() const { return array.rbegin(); }
00436
00437
00438 // const_reverse_iterator rend() const { return array.rend(); }
00439
00440
00441 const_reverse_iterator crbegin() const { return array.rbegin(); }
00442
00443
00444 const_reverse_iterator crend() const { return array.rend(); }
00445
00446 private:
00447
00448 // The following function are used from handler

```

```

00449     friend handler;
00450
00451 #ifndef TRISYCL_OPENCL
00452     /// Get the boost::compute::buffer or throw if unset
00453     auto get_cl_buffer() const {
00454         /// This throws if not set
00455         auto ctx = task->get_queue()->get_context();
00456         return buf->get_cl_buffer(ctx);
00457     }
00458
00459     /** Lazily associate a CL buffer to the SYCL buffer and copy data in it
00460         if required, updates the state of the data in the buffer across contexts
00461     */
00462     void copy_in_cl_buffer() {
00463         /** Create the OpenCL buffer and copy in it the data from the host if
00464             the buffer doesn't already exists or if the data is not up to date
00465         */
00466         auto ctx = task->get_queue()->get_context();
00467         buf->update_buffer_state(ctx, Mode, get_size(), array.data());
00468     }
00469
00470     /// Does nothing
00471     void copy_back_cl_buffer() {
00472         /** The copy back is handled by the host accessor and the buffer destructor.
00473             We don't need to systematically transfer the data after the
00474             kernel execution
00475             \todo Figure out what to do with this function
00476         */
00477     }
00478 #endif
00479 };
00480
00481 // @} End the data Doxygen group
00482
00483 // @} End the data Doxygen group
00484
00485 // @} End the data Doxygen group
00486
00487 // @} End the data Doxygen group
00488
00489 // @} End the data Doxygen group
00490
00491 /*
00492     # Some Emacs stuff:
00493     ### Local Variables:
00494     ### ispell-local-dictionary: "american"
00495     ### eval: (flyspell-prog-mode)
00496     ### End:
00497 */
00498
00499 #endif // TRISYCL_SYCL_ACCESSOR_DETAIL_ACCESSOR_HPP

```

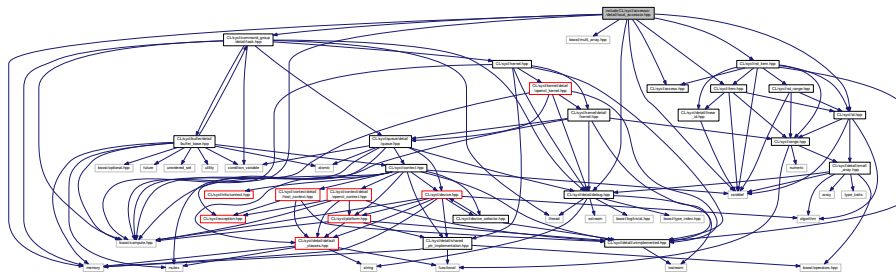
## 11.9 include/CL/sycl/accessor/detail/local\_accessor.hpp File Reference

```

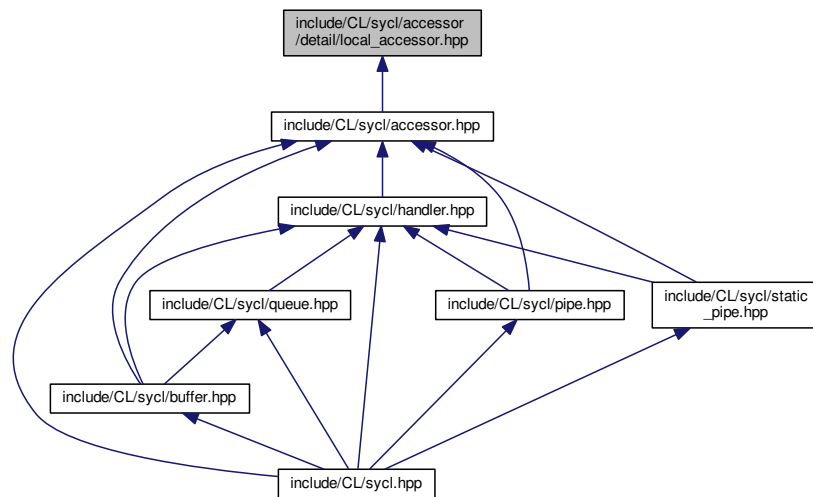
#include <cstddef>
#include <memory>
#include <boost/compute.hpp>
#include <boost/multi_array.hpp>
#include "CL/sycl/access.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"

```

Include dependency graph for local\_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::accessor< T, Dimensions, Mode, access::target::local >`  
The local accessor specialization abstracts the way local memory is allocated to a kernel to be shared between work-items of the same work-group. [More...](#)

## Namespaces

- `cl`  
The vector type to be used as SYCL vector.
- `cl::sycl`
- `cl::sycl::detail`



## 11.10 local\_accessor.hpp

```

00001 #ifndef TRISYCL_SYCL_ACCESSOR_DETAIL_LOCAL_ACCESSOR_HPP
00002 #define TRISYCL_SYCL_ACCESSOR_DETAIL_LOCAL_ACCESSOR_HPP
00003
00004 /** \file The OpenCL SYCL local 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 <cstdlib>
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 class handler;
00030
00031 namespace detail {
00032 // Forward declaration of detail::accessor to declare the specialization
00033 template <typename T,
00034           int Dimensions,
00035           access::mode Mode,
00036           access::target Target>
00037 class accessor;
00038
00039 /** \addtogroup data Data access and storage in SYCL
00040     @{
00041
00042     /** The local accessor specialization abstracts the way local memory
00043         is allocated to a kernel to be shared between work-items of the
00044         same work-group.
00045
00046         \todo Use the access::mode
00047
00048     template <typename T,
00049             int Dimensions,
00050             access::mode Mode>
00051     class accessor<T, Dimensions, Mode, access::target::local> :
00052     public detail::debug<accessor<T,
00053                                 Dimensions,
00054                                 Mode,
00055                                 access::target::local>> {
00056
00057     /// The implementation is a multi_array_ref wrapper
00058     using array_type = boost::multi_array_ref<T, Dimensions>;
00059
00060     // The same type but writable
00061     // \todo Only if T is non const actually
00062     using writable_array_type =
00063     typename std::remove_const<array_type>::type;
00064
00065     /** The way the buffer is really accessed
00066
00067         Use a mutable member because the accessor needs to be captured
00068         by value in the lambda which is then read-only. This is to avoid
00069         the user to use mutable lambda or have a lot of const_cast as
00070         previously done in this implementation
00071
00072     mutable writable_array_type array;
00073
00074     /** The allocation on the host for the local accessor
00075
00076         Note that this is uninitialized memory, as stated in SYCL
00077         specification.
00078
00079     mutable T *allocation = nullptr;
00080
00081     public:

```

```

00085
00086 /** \todo in the specification: store the dimension for user request
00087
00088     \todo Use another name, such as from C++17 committee discussions.
00089 */
00090 static constexpr auto dimensionality = Dimensions;
00091
00092 /** \todo in the specification: store the types for user request as STL
00093     or C++AMP */
00094 using value_type = T;
00095 using element = T;
00096 using reference = typename array_type::reference;
00097 using const_reference = typename array_type::const_reference;
00098
00099 /** Inherit the iterator types from the implementation
00100
00101     \todo Add iterators to accessors in the specification
00102 */
00103 using iterator = typename array_type::iterator;
00104 using const_iterator = typename array_type::const_iterator;
00105 using reverse_iterator = typename array_type::reverse_iterator;
00106 using const_reverse_iterator =
00107     typename array_type::const_reverse_iterator;
00108
00109
00110 /** Construct a device accessor from an existing buffer
00111
00112     \todo fix the specification to rename target that shadows
00113         template param
00114 */
00115 accessor(const range<Dimensions> &allocation_size,
00116         handler &command_group_handler) :
00117     array { allocate_accessor(allocation_size) } {}
00118
00119
00120 // Deallocate the memory
00121 ~accessor() {
00122     deallocate_accessor();
00123 }
00124
00125
00126 /** Return a range object representing the size of the buffer in
00127     terms of number of elements in each dimension as passed to the
00128     constructor
00129
00130     \todo Move on
00131         https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
00132         https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
00133 */
00134 auto get_range() const {
00135     /* Interpret the shape which is a pointer to the first element as an
00136        array of Dimensions elements so that the range<Dimensions>
00137        constructor is happy with this collection
00138
00139        \todo Add also a constructor in range<> to accept a const
00140            std::size_t */
00141     */
00142     return range<Dimensions> {
00143         *(const std::size_t (*)[Dimensions]) (array.shape())
00144     };
00145 }
00146
00147
00148 /** Returns the total number of elements behind the accessor
00149
00150     Equal to get_range()[0] * ... * get_range()[Dimensions-1].
00151
00152     \todo Move on
00153         https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
00154         https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
00155 */
00156 auto get_count() const {
00157     return array.num_elements();
00158 }
00159
00160
00161 /** Returns the size of the underlying buffer storage in bytes
00162
00163     \todo Move on
00164         https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
00165         https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
00166 */
00167 auto get_size() const {
00168     return get_count()*sizeof(value_type);
00169 }
00170
00171

```

```

00172  /** Use the accessor with integers à la [][][]
00173
00174      Use array_view_type::reference instead of auto& because it does not
00175      work in some dimensions.
00176  */
00177  reference operator[](std::size_t index) {
00178      return array[index];
00179  }
00180
00181
00182  /** Use the accessor with integers à la [][][]
00183
00184      Use array_view_type::reference instead of auto& because it does not
00185      work in some dimensions.
00186  */
00187  reference operator[](std::size_t index) const {
00188      return array[index];
00189  }
00190
00191
00192  /// To use the accessor with [id<>]
00193  auto &operator[](id<dimensionality> index) {
00194      return array(index);
00195  }
00196
00197
00198  /// To use the accessor with [id<>]
00199  auto &operator[](id<dimensionality> index) const {
00200      return array(index);
00201  }
00202
00203
00204  /// To use an accessor with [item<>]
00205  auto &operator[](item<dimensionality> index) {
00206      return (*this)[index.get()];
00207  }
00208
00209
00210  /// To use an accessor with [item<>]
00211  auto &operator[](item<dimensionality> index) const {
00212      return (*this)[index.get()];
00213  }
00214
00215
00216  /** To use an accessor with an [nd_item<>]
00217
00218      \todo Add in the specification because used by HPC-GPU slide 22
00219  */
00220  auto &operator[](nd_item<dimensionality> index) {
00221      return (*this)[index.get_global()];
00222  }
00223
00224  /** To use an accessor with an [nd_item<>]
00225
00226      \todo Add in the specification because used by HPC-GPU slide 22
00227  */
00228  auto &operator[](nd_item<dimensionality> index) const {
00229      return (*this)[index.get_global()];
00230  }
00231
00232
00233  /** Get the first element of the accessor
00234
00235      Useful with an accessor on a scalar for example.
00236
00237      \todo Add in the specification
00238  */
00239  reference operator*() {
00240      return *array.data();
00241  }
00242
00243
00244  /** Get the first element of the accessor
00245
00246      Useful with an accessor on a scalar for example.
00247
00248      \todo Add in the specification?
00249
00250      \todo Add the concept of 0-dim buffer and accessor for scalar
00251      and use an implicit conversion to value_type reference to access
00252      the value with the accessor?
00253  */
00254  reference operator*() const {
00255      return *array.data();
00256  }
00257
00258

```

```

00259  /** Test if the accessor has a read access right
00260
00261      \todo Strangely, it is not really constexpr because it is not a
00262      static method...
00263
00264      \todo to move in the access::mode enum class and add to the
00265      specification ?
00266  */
00267  constexpr bool is_read_access() const {
00268      return Mode == access::mode::read
00269      || Mode == access::mode::read_write
00270      || Mode == access::mode::discard_read_write;
00271  }
00272
00273
00274  /** Test if the accessor has a write access right
00275
00276      \todo Strangely, it is not really constexpr because it is not a
00277      static method...
00278
00279      \todo to move in the access::mode enum class and add to the
00280      specification ?
00281  */
00282  constexpr bool is_write_access() const {
00283      return Mode == access::mode::write
00284      || Mode == access::mode::read_write
00285      || Mode == access::mode::discard_write
00286      || Mode == access::mode::discard_read_write;
00287  }
00288
00289
00290  /** Forward all the iterator functions to the implementation
00291
00292      \todo Add these functions to the specification
00293
00294      \todo The fact that the lambda capture make a const copy of the
00295      accessor is not yet elegantly managed... The issue is that
00296      begin()/end() dispatch is made according to the accessor
00297      constness and not from the array member constness...
00298
00299      \todo try to solve it by using some enable_if on array
00300      constness?
00301
00302      \todo The issue is that the end may not be known if it is
00303      implemented by a raw OpenCL cl_mem... So only provide on the
00304      device the iterators related to the start? Actually the accessor
00305      needs to know a part of the shape to have the multidimensional
00306      addressing. So this only require a size_t more...
00307
00308      \todo Factor out these in a template helper
00309
00310      \todo Do we need this in detail::accessor too or only in accessor?
00311  */
00312
00313  // iterator begin() { return array.begin(); }
00314  iterator begin() const {
00315      return const_cast<writable_array_type >(array).
00316      begin();
00317  }
00318
00319  // iterator end() { return array.end(); }
00320  iterator end() const {
00321      return const_cast<writable_array_type >(array).end();
00322  }
00323
00324
00325  // const_iterator begin() const { return array.begin(); }
00326
00327
00328  // const_iterator end() const { return array.end(); }
00329
00330
00331  const_iterator cbegin() const { return array.begin(); }
00332
00333
00334  const_iterator cend() const { return array.end(); }
00335
00336
00337  // reverse_iterator rbegin() { return array.rbegin(); }
00338  reverse_iterator rbegin() const {
00339      return const_cast<writable_array_type >(array).
00340      rbegin();
00341  }
00342
00343

```

```

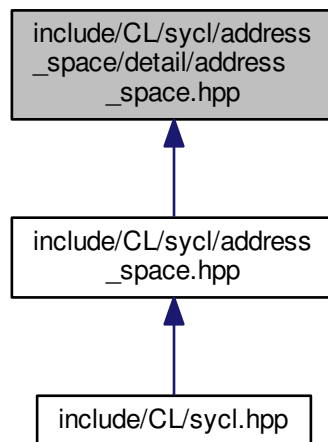
00344 // reverse_iterator rend() { return array.rend(); }
00345 reverse_iterator rend() const {
00346     return const_cast<writable_array_type &>(array).rend();
00347 }
00348
00349
00350 // const_reverse_iterator rbegin() const { return array.rbegin(); }
00351
00352
00353 // const_reverse_iterator rend() const { return array.rend(); }
00354
00355
00356 const_reverse_iterator crbegin() const { return array.rbegin(); }
00357
00358
00359 const_reverse_iterator crend() const { return array.rend(); }
00360
00361 private:
00362
00363     /// Allocate uninitialized buffer memory
00364     auto allocate_accessor(const range<Dimensions> &r) {
00365         auto count = r.get_count();
00366         /// Allocate uninitialized memory
00367         allocation = std::allocator<value_type>{}.allocate(count);
00368         return boost::multi_array_ref<value_type, Dimensions> { allocation, r };
00369     }
00370
00371
00372     /// Deallocate accessor memory
00373     void deallocate_accessor() {
00374         std::allocator<value_type>{}.deallocate(allocation, array.num_elements());
00375     }
00376
00377
00378     // The following function are used from handler
00379     friend handler;
00380
00381
00382 };
00383
00384 /// @} End the data Doxygen group
00385
00386 }
00387 }
00388 }
00389
00390 /*
00391     # Some Emacs stuff:
00392     ### Local Variables:
00393     ### ispell-local-dictionary: "american"
00394     ### eval: (flyspell-prog-mode)
00395     ### End:
00396 */
00397
00398 #endif // TRISYCL_SYCL_ACCESSOR_DETAIL_LOCAL_ACCESSOR_HPP

```

## 11.11 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::ocl\\_type< T, AS >](#)  
Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device. [More...](#)
- struct [cl::sycl::detail::ocl\\_type< T, constant\\_address\\_space >](#)  
Add an attribute for `__constant` address space. [More...](#)
- struct [cl::sycl::detail::ocl\\_type< T, generic\\_address\\_space >](#)  
Add an attribute for `__generic` address space. [More...](#)
- struct [cl::sycl::detail::ocl\\_type< T, global\\_address\\_space >](#)  
Add an attribute for `__global` address space. [More...](#)
- struct [cl::sycl::detail::ocl\\_type< T, local\\_address\\_space >](#)  
Add an attribute for `__local` address space. [More...](#)
- struct [cl::sycl::detail::ocl\\_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  
*Dispatch the address space implementation according to the requested type.*

### 11.11.1 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](#).

## 11.12 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 ocl_type { // NOTE: renamed from opcnl_type because of MSVC bug
00028     using type = T;
00029 };
00030
00031 /// Add an attribute for __constant address space
00032 template <typename T>
00033 struct ocl_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 ocl_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 ocl_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 ocl_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 ocl_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 ocl_type = typename ocl_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 ocl_type = typename ocl_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     ocl_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     using pointer_t = typename super::address_space_fundamental::type
00289 ;
00289     using reference_t = typename std::remove_pointer_t<pointer_t>&;
00290
00291     /** Allow initialization of a pointer type from the address of an
00292         element with the same type and address space
00293     */
00294     address_space_ptr(address_space_fundamental<typename
std::pointer_traits<T>::element_type, AS> *p)
        : address_space_fundamental<T, AS> { p->get_address() } {}
00295
00296
00297     /// Put back the default constructors canceled by the previous definition
00298     address_space_ptr() = default;
00299 };
00300
00301
00302 /** Implementation of an array variable with an OpenCL address space
00303
00304     \param T is the type of the basic object to be created
00305
00306     \param AS is the address space to place the object into
00307 */
00308 template <typename T, address_space AS>
00309 struct address_space_array : public address_space_variable<T, AS>
00310 {
00311     /// Keep track of the base class as a short-cut
00312     using super = address_space_variable<T, AS>;
00313
00314     /// Inherit from base class constructors
00315     using super::address_space_variable;
00316
00317     /** Allow to create an address space array from an array
00318     */
00319     address_space_array(const T &array) {
00320         std::copy(std::begin(array), std::end(array), std::begin(super::variable));
00321     };
00322
00323
00324     /** Allow to create an address space array from an initializer list
00325
00326         \todo Extend to more than 1 dimension
00327     */
00328     address_space_array(std::initializer_list<std::remove_extent_t<T>> list) {
00329         std::copy(std::begin(list), std::end(list), std::begin(super::variable));
00330     };
00331
00332 };
00333
00334
00335 /** Implementation of an object type with an OpenCL address space
00336
00337     \param T is the type of the basic object to be created
00338
00339     \param AS is the address space to place the object into
00340
00341     The class implementation is just inheriting of T so that all methods
00342     and non-member operators on T work also on address_space_object<T>
00343
00344     \todo Verify/improve to deal with const/volatile?
00345
00346     \todo what about T having some final methods?
00347 */
00348 template <typename T, address_space AS>
00349 //struct address_space_object : public opengl_type<T, AS>::type,
00350 struct address_space_object : public ocl_type<T, AS>::type,
00351                             public address_space_base<T, AS> {
00352     /** Store the base type of the object with OpenCL address space modifier
00353
00354         \todo Add to the specification
00355     */
00356     using opengl_type = typename ocl_type<T, AS>::type;
00357
00358     /* C++11 helps a lot to be able to have the same constructors as the
00359         parent class here but with an OpenCL address space
00360
00361         \todo Add this to the list of required C++11 features needed for SYCL
00362     */
00363     using opengl_type::opengl_type;
00364

```

```

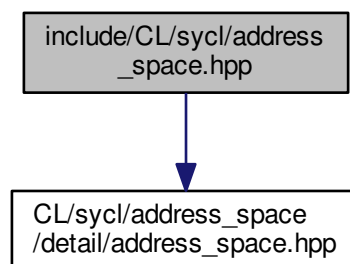
00365  /** Allow to create an address space version of an object or to
00366      convert one */
00367  address_space_object(T && v) : opengl_type(v) { }
00368
00369  /** Conversion operator to allow a address_space_object<T> to be used
00370      as a T so that all the methods of a T and the built-in operators for
00371      T can be used on a address_space_object<T> too.
00372
00373      Use opengl_type so that if we take the address of it, the address
00374      space is kept. */
00375  operator opengl_type & () { return *this; }
00376
00377  };
00378
00379  /// @} End the address_spaces Doxygen group
00380  }
00381  }
00382  }
00383  }
00384
00385  /*
00386      # Some Emacs stuff:
00387      ### Local Variables:
00388      ### ispell-local-dictionary: "american"
00389      ### eval: (flyspell-prog-mode)
00390      ### End:
00391  */
00392
00393  #endif // TRISYCL_SYCL_ADDRESS_SPACES_DETAIL_ADDRESS_SPACES_HPP

```

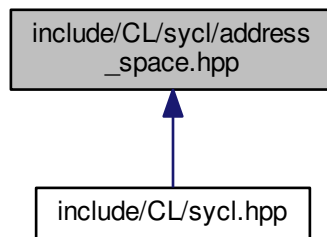
## 11.13 include/CL/sycl/address\_space.hpp File Reference

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::constant\_ptr = constant< T * >`  
*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::global\_ptr = global< T * >`  
*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::local\_ptr = local< T * >`  
*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 T >`  
`using cl::sycl::private\_ptr = priv< T * >`  
*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.

### 11.13.1 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](#).

## 11.14 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
00052     \param T is the type of the object
00053 */
00054 template <typename T>
00055 using constant = detail::addr_space<T, constant_address_space>
00056 ;
00057
00058 /** Declare a variable to be in the OpenCL constant address space
00059
00060     \param T is the type of the object
00061 */
00062 template <typename T>
00063 using constant_ptr = constant<T*>;
00064
00065
00066 /** Declare a variable to be in the OpenCL 2 generic address space
00067
00068     \param T is the type of the object
00069 */
00070 template <typename T>
00071 using generic = detail::addr_space<T, generic_address_space>;
00072
00073
00074 /** Declare a variable to be in the OpenCL global address space
00075
00076     \param T is the type of the object
00077 */
00078 template <typename T>
00079 using global = detail::addr_space<T, global_address_space>
00080 ;
00081
00082 /** Declare a variable to be in the OpenCL global address space
00083
00084     \param T is the type of the object
00085 */
00086
00087 template <typename T>
00088 using global_ptr = global<T*>;
00089
00090
00091 /** Declare a variable to be in the OpenCL local address space
00092
00093     \param T is the type of the object
00094 */
00095 template <typename T>
00096 using local = detail::addr_space<T, local_address_space>;
00097
00098
00099 /** Declare a variable to be in the OpenCL local address space
00100
00101     \param T is the type of the object
00102 */
00103 template <typename T>
00104 using local_ptr = local<T*>;
00105
00106
00107 /** Declare a variable to be in the OpenCL private address space
00108
00109     \param T is the type of the object
00110 */
00111 template <typename T>
00112 using priv = detail::addr_space<T, private_address_space>;
00113
00114
00115 /** Declare a variable to be in the OpenCL private address space

```

```

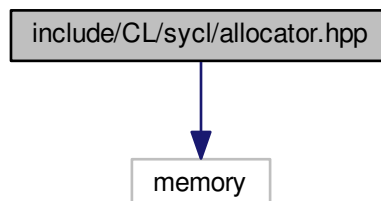
00116
00117     \param T is the type of the object
00118 */
00119 template <typename T>
00120 using private_ptr = priv<T*>;
00121
00122
00123 /** A pointer that can be statically associated to any address-space
00124
00125     \param Pointer is the pointer type
00126
00127     \param AS is the address space to point to
00128
00129     Note that if \a Pointer is not a pointer type, it is an error.
00130 */
00131 template <typename Pointer, address_space AS>
00132 using multi_ptr = detail::address_space_ptr<Pointer, AS>;
00133
00134
00135 /** Construct a cl::sycl::multi_ptr<> with the right type
00136
00137     \param pointer is the address with its address space to point to
00138
00139     \todo Implement the case with a plain pointer
00140 */
00141 template <typename T, address_space AS>
00142 multi_ptr<T, AS> make_multi(multi_ptr<T, AS> pointer) {
00143     return pointer;
00144 }
00145
00146 }
00147 }
00148 /// @} End the parallelism Doxygen group
00149
00150 /*
00151     # Some Emacs stuff:
00152     ### Local Variables:
00153     ### ispell-local-dictionary: "american"
00154     ### eval: (flyspell-prog-mode)
00155     ### End:
00156 */
00157
00158 #endif // TRISYCL_SYCL_ADDRESS_SPACE_HPP

```

## 11.15 include/CL/sycl/allocator.hpp File Reference

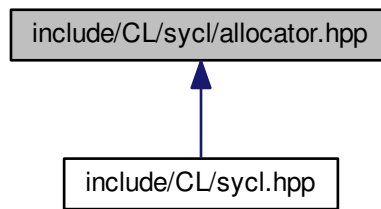
#include <memory>

Include dependency graph for 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 allocator objects give the programmer some control on how the memory is allocated inside SYCL.*
- `template<typename T >`  
`using cl::sycl::image_allocator = std::allocator< T >`  
*The allocator used for the `image` inside SYCL.*
- `template<typename T >`  
`using cl::sycl::map_allocator = std::allocator< T >`  
*The allocator used to map the memory at the same place.*

## 11.16 allocator.hpp

```

00001 #ifndef TRISYCL_SYCL_ALLOCATOR_HPP
00002 #define TRISYCL_SYCL_ALLOCATOR_HPP
00003
00004 /** \file The OpenCL SYCL 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 <memory>
00013
00014 namespace cl {
00015 namespace sycl {
00016
00017 /** \addtogroup data Data access and storage in SYCL
00018     @{
00019 */
00020
00021 /** The allocator objects give the programmer some control on how the
00022     memory is allocated inside SYCL

```

```

00023 */
00024
00025 /** The allocator used for the \c buffer inside SYCL
00026
00027     Just use the default allocator for now.
00028 */
00029 template <typename T>
00030 using buffer_allocator = std::allocator<T>;
00031
00032
00033 /** The allocator used for the \c image inside SYCL
00034
00035     Just use the default allocator for now.
00036 */
00037 template <typename T>
00038 using image_allocator = std::allocator<T>;
00039
00040
00041 /** The allocator used to map the memory at the same place
00042
00043     Just use the default allocator for now.
00044
00045     \todo : implement and clarify the specification. It looks like it
00046     is not really an allocator according the current spec
00047 */
00048 template <typename T>
00049 using map_allocator = std::allocator<T>;
00050
00051
00052 /// @} End the data Doxygen group
00053 }
00054 }
00055 }
00056
00057 /*
00058     # Some Emacs stuff:
00059     ### Local Variables:
00060     ### ispell-local-dictionary: "american"
00061     ### eval: (flyspell-prog-mode)
00062     ### End:
00063 */
00064
00065 #endif // TRISYCL_SYCL_ALLOCATOR_HPP

```

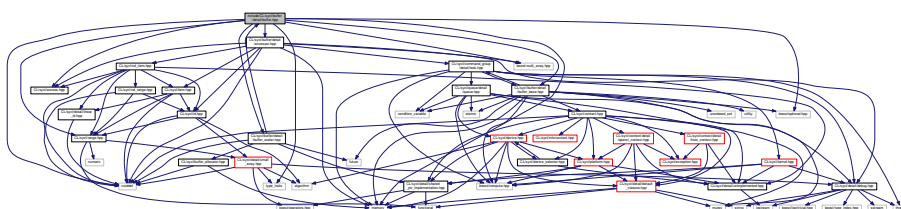
## 11.17 include/CL/sycl/buffer/detail/buffer.hpp File Reference

```

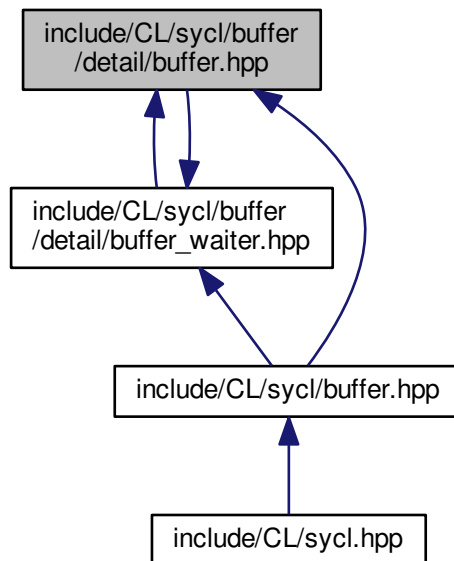
#include <cstdint>
#include <memory>
#include <type_traits>
#include <boost/multi_array.hpp>
#include <boost/optional.hpp>
#include "CL/sycl/access.hpp"
#include "CL/sycl/buffer/detail/accessor.hpp"
#include "CL/sycl/buffer/detail/buffer_base.hpp"
#include "CL/sycl/buffer/detail/buffer_waiter.hpp"
#include "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.

## 11.18 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 <cstddef>
00013 #include <memory>
00014 #include <type_traits>
00015
00016 #include <boost/multi_array.hpp>
00017 // \todo Use C++17 optional when it is mainstream
00018 #include <boost/optional.hpp>
00019
00020 #include "CL/sycl/access.hpp"
00021 #include "CL/sycl/buffer/detail/accessor.hpp"
00022 #include "CL/sycl/buffer/detail/buffer_base.hpp"
00023 #include "CL/sycl/buffer/detail/buffer_waiter.hpp"
00024 #include "CL/sycl/range.hpp"
00025
00026 namespace cl {
00027 namespace sycl {
00028 namespace detail {
00029
00030
00031 /** \addtogroup data Data access and storage in SYCL
00032     @{
00033 */
00034
00035 /** A SYCL buffer is a multidimensional variable length array (à la C99
00036     VLA or even Fortran before) that is used to store data to work on.
00037
00038     In the case we initialize it from a pointer, for now we just wrap the
00039     data with boost::multi_array_ref to provide the VLA semantics without
00040     any storage.
00041 */
00042 template <typename T,
00043           int Dimensions = 1>
00044 class buffer : public detail::buffer_base,
00045               public detail::debug<buffer<T, Dimensions>> {
00046 public:
00047
00048     // Extension to SYCL: provide pieces of STL container interface
00049     using element = T;
00050     using value_type = T;
00051     // Even if the buffer is read-only use a non-const type so at
00052     // least the current implementation can copy the data too */
00053     using non_const_value_type = std::remove_const_t<value_type>;
00054 private:
00055
00056     // \todo Replace U and D somehow by T and Dimensions
00057     // To allow allocation access
00058     template <typename U,
00059              int D,
00060              access::mode Mode,
00061              access::target Target /* = access::global_buffer */>
00062     friend class detail::accessor;
00063
00064
00065     /** The allocator to be used when some memory is needed
00066
00067         \todo Implement user-provided allocator
00068     */
00069     std::allocator<non_const_value_type> alloc;
00070
00071     /** This is the multi-dimensional interface to the data that may point
00072         to either allocation in the case of storage managed by SYCL itself
00073         or to some other memory location in the case of host memory or
00074         storage<> abstraction use
00075     */
00076     boost::multi_array_ref<value_type, Dimensions> access;
00077
00078     /** If some allocation is requested on the host for the buffer
00079         memory, this is where the memory is attached to.
00080
00081         Note that this is uninitialized memory, as stated in SYCL
00082         specification.
00083     */
00084     non_const_value_type *allocation = nullptr;

```

```

00085
00086  /* How to copy back data on buffer destruction, can be modified with
00087   set_final_data( ... )
00088   */
00089  boost::optional<std::function<void(void)>> final_write_back;
00090
00091  // Keep the shared pointer used to create the buffer
00092  shared_ptr_class<T> input_shared_pointer;
00093
00094
00095  // Track if the buffer memory is provided as host memory
00096  bool data_host = false;
00097
00098  // Track if data should be copied if a modification occurs
00099  bool copy_if_modified = false;
00100
00101  // Track if data have been modified
00102  bool modified = false;
00103
00104 public:
00105
00106  /// Create a new read-write buffer of size \param r
00107  buffer(const range<Dimensions> &r) : access {
    allocate_buffer(r) } {}
00108
00109
00110  /** Create a new read-write buffer from \param host_data of size
00111   \param r without further allocation */
00112  buffer(T *host_data, const range<Dimensions> &r) :
00113   access { host_data, r },
00114   data_host { true }
00115  {}
00116
00117
00118  /** Create a new read-only buffer from \param host_data of size \param r
00119   without further allocation
00120
00121   If the buffer is non const, use a copy-on-write mechanism with
00122   internal writable memory.
00123
00124   \todo Clarify the semantics in the spec. What happens if the
00125   host change the host_data after buffer creation?
00126
00127   Only enable this constructor if the value type is not constant,
00128   because if it is constant, the buffer is constant too.
00129   */
00130  template <typename Dependent = T,
00131           typename = std::enable_if_t<!std::is_const<Dependent>::value>>
00132  buffer(const T *host_data, const range<Dimensions> &r) :
00133   /* The buffer is read-only, even if the internal multidimensional
00134   wrapper is not. If a write accessor is requested, there should
00135   be a copy on write. So this pointer should not be written and
00136   this const_cast should be acceptable. */
00137   access { const_cast<T *>(host_data), r },
00138   data_host { true },
00139   /* Set copy_if_modified to true, so that if an accessor with write
00140   access is created, data are copied before to be modified. */
00141   copy_if_modified { true }
00142  {}
00143
00144
00145  /** Create a new buffer with associated memory, using the data in
00146   host_data
00147
00148   The ownership of the host_data is shared between the runtime and the
00149   user. In order to enable both the user application and the SYCL
00150   runtime to use the same pointer, a cl::sycl::mutex_class is
00151   used.
00152   */
00153  buffer(shared_ptr_class<T> &host_data, const
    range<Dimensions> &r) :
00154   access { host_data.get(), r },
00155   input_shared_pointer { host_data },
00156   data_host { true }
00157  {}
00158
00159
00160  /// Create a new allocated 1D buffer from the given elements
00161  template <typename Iterator>
00162  buffer(Iterator start_iterator, Iterator end_iterator) :
00163   access { allocate_buffer(std::distance(start_iterator, end_iterator)) }
00164   {
00165     /* Then assign allocation since this is the only multi_array
00166     method with this iterator interface */
00167     access.assign(start_iterator, end_iterator);
00168   }
00169

```

```

00170
00171 /** Create a new sub-buffer without allocation to have separate
00172     accessors later
00173
00174     \todo To implement and deal with reference counting
00175     buffer(buffer<T, Dimensions> b,
00176           index<Dimensions> base_index,
00177           range<Dimensions> sub_range)
00178 */
00179
00180 /// \todo Allow CLHPP objects too?
00181 ///
00182 /**
00183     buffer(cl_mem mem_object,
00184           queue from_queue,
00185           event available_event)
00186 */
00187
00188
00189 /** The buffer content may be copied back on destruction to some
00190     final location */
00191 ~buffer() {
00192 #ifdef TRISYCL_OPENCL
00193     /* We ensure that the host has the most up-to-date version of the data
00194        before the buffer is destroyed. This is necessary because we do not
00195        systematically transfer the data back from a device with
00196        \c copy_back_cl_buffer any more.
00197        \todo Optimize for the case the buffer is not based on host memory
00198     */
00199     cl::sycl::context ctx;
00200     auto size = access.num_elements() * sizeof(value_type);
00201     call_update_buffer_state(ctx, access::mode::read, size,
00202                             access.data());
00203 #endif
00204     if (modified && final_write_back)
00205         (*final_write_back)();
00206     // Allocate explicitly allocated memory if required
00207     deallocate_buffer();
00208 }
00209
00210
00211 /** Enforce the buffer to be considered as being modified.
00212     Same as creating an accessor with write access.
00213 */
00214 void mark_as_written() {
00215     modified = true;
00216 }
00217
00218
00219 /** This method is to be called whenever an accessor is created
00220
00221     Its current purpose is to track if an accessor with write access
00222     is created and acting accordingly.
00223 */
00224 template <access::mode Mode,
00225           access::target Target = access::target::host_buffer>
00226 void track_access_mode() {
00227     // test if write access is required
00228     if ( Mode == access::mode::write
00229         || Mode == access::mode::read_write
00230         || Mode == access::mode::discard_write
00231         || Mode == access::mode::discard_read_write
00232         || Mode == access::mode::atomic
00233     ) {
00234         modified = true;
00235         if (copy_if_modified) {
00236             // Implement the allocate & copy-on-write optimization
00237             copy_if_modified = false;
00238             data_host = false;
00239             // Since \c allocate_buffer() changes \c access, keep a copy first
00240             auto current_access = access;
00241             /* The range is actually computed from \c access itself, so
00242                save it */
00243             auto current_range = get_range();
00244             allocate_buffer(current_range);
00245             /* Then move everything to the new place
00246
00247                \todo Use std::uninitialized_move instead, when we switch
00248                to full C++17
00249             */
00250             std::copy(current_access.begin(),
00251                     current_access.end(),
00252                     access.begin());
00253         }
00254     }
00255 }

```

```

00256
00257
00258 /** Return a range object representing the size of the buffer in
00259     terms of number of elements in each dimension as passed to the
00260     constructor
00261 */
00262 auto get_range() const {
00263     /* Interpret the shape which is a pointer to the first element as an
00264        array of Dimensions elements so that the range<Dimensions>
00265        constructor is happy with this collection
00266
00267        \todo Add also a constructor in range<> to accept a const
00268        std::size_t */
00269     /*
00270     return range<Dimensions> {
00271         *(const std::size_t (*) [Dimensions]) (access.shape())
00272     };
00273     }
00274
00275
00276 /** Returns the total number of elements in the buffer
00277
00278     Equal to get_range()[0] * ... * get_range()[Dimensions-1].
00279 */
00280 auto get_count() const {
00281     return access.num_elements();
00282 }
00283
00284
00285 /** Returns the size of the buffer storage in bytes
00286
00287     \todo rename to something else. In
00288     http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf
00289     it is named bytes() for example
00290 */
00291 auto get_size() const {
00292     return get_count() * sizeof(value_type);
00293 }
00294
00295
00296 /** Set the weak pointer as destination for write-back on buffer
00297     destruction
00298 */
00299 void set_final_data(std::weak_ptr<T> && final_data) {
00300     final_write_back = [=] {
00301         if (auto sptr = final_data.lock()) {
00302             std::copy_n(access.data(), access.num_elements(), sptr.get());
00303         }
00304     };
00305 }
00306
00307
00308 /** Provide destination for write-back on buffer destruction as a
00309     shared pointer.
00310 */
00311 void set_final_data(std::shared_ptr<T> && final_data) {
00312     final_write_back = [=] {
00313         std::copy_n(access.data(), access.num_elements(), final_data.get());
00314     };
00315 }
00316
00317
00318 /** Disable write-back on buffer destruction as an iterator.
00319 */
00320 void set_final_data(std::nullptr_t) {
00321     final_write_back = boost::none;
00322 }
00323
00324
00325 /** Provide destination for write-back on buffer destruction as an
00326     iterator
00327 */
00328 template <typename Iterator>
00329 void set_final_data(Iterator final_data) {
00330     /* using type_ = typename iterator_value_type<Iterator>::value_type;
00331        static_assert(std::is_same<type_, T>::value, "buffer type mismatch");
00332        static_assert(!std::is_const<type_>::value,
00333            "const iterator is not allowed");*/
00334     final_write_back = [=] {
00335         std::copy_n(access.data(), access.num_elements(), final_data);
00336     };
00337 }
00338
00339
00340 private:
00341
00342     /// Allocate uninitialized buffer memory

```

```

00343 auto allocate_buffer(const range<Dimensions> &r) {
00344     auto count = r.get_count();
00345     // Allocate uninitialized memory
00346     allocation = alloc.allocate(count);
00347     return boost::multi_array_ref<value_type, Dimensions> { allocation, r };
00348 }
00349
00350
00351 /// Deallocate buffer memory if required
00352 void deallocate_buffer() {
00353     if (allocation)
00354         alloc.deallocate(allocation, access.num_elements());
00355 }
00356
00357
00358 /** Function pair to work around the fact that T might be a \c const type.
00359     We call update_buffer_state only if T is not \c const, we have to
00360     use \c enable_if otherwise the compiler will try to cast \c const
00361     \c void* to \c void* if we create a buffer with a \c const type
00362
00363     \todo Use \c if \c constexpr when it is available with C++17
00364 */
00365 template <typename BaseType = T, typename DataType>
00366 void call_update_buffer_state(cl::sycl::context ctx,
00367     access::mode mode,
00368     size_t size, DataType* data,
00369     std::enable_if_t<!std::is_const<BaseType>
00370         ::value>* = 0) {
00371     update_buffer_state(ctx, mode, size, data);
00372 }
00373
00374 /** Version of \c call_update_buffer_state that does nothing. It is called if
00375     the type of the data in the buffer is \c const
00376 */
00377 template <typename BaseType = T, typename DataType>
00378 void call_update_buffer_state(cl::sycl::context ctx,
00379     access::mode mode,
00380     size_t size, DataType* data,
00381     std::enable_if_t<std::is_const<BaseType>
00382         ::value>* = 0) { }
00383
00384 public:
00385
00386 /** Get a \c future to wait from inside the \c cl::sycl::buffer in
00387     case there is something to copy back to the host
00388
00389     \return A \c future in the \c optional if there is something to
00390     wait for, otherwise an empty \c optional
00391     \todo Make the function private again
00392 */
00393 boost::optional<std::future<void>> get_destructor_future() {
00394     /* If there is only 1 shared_ptr user of the buffer, this is the
00395     caller of this function, the \c buffer_waiter, so there is no
00396     need to get a \c future otherwise there will be a dead-lock if
00397     there is only 1 thread waiting for itself.
00398
00399     Since \c use_count() is applied to a \c shared_ptr just created
00400     for this purpose, it actually increase locally the count by 1,
00401     so check for 1 + 1 use count instead...
00402     */
00403     // If the buffer's destruction triggers a write-back, wait
00404     if ((shared_from_this().use_count() > 2) &&
00405         modified && (final_write_back || data_host)) {
00406         // Create a promise to wait for
00407         notify_buffer_destructor = std::promise<void> {};
00408         // And return the future to wait for it
00409         return notify_buffer_destructor->get_future();
00410     }
00411     return boost::none;
00412 }
00413
00414 private:
00415
00416 // Allow buffer_waiter destructor to access get_destructor_future()
00417 // friend detail::buffer_waiter<T, Dimensions>::~buffer_waiter();
00418 /* \todo Work around to Clang bug
00419     https://llvm.org/bugs/show_bug.cgi?id=28873 cannot use destructor
00420     here */
00421 /* \todo solve the fact that get_destructor_future is not accessible
00422     when private and buffer_waiter uses a custom allocator */
00423 friend detail::buffer_waiter<T, Dimensions>;
00424
00425 };
00426
00427

```



```

00428 /** Proxy function to avoid some circular type recursion
00429
00430     \return a shared_ptr<task>
00431
00432     \todo To remove with some refactoring
00433 */
00434 template <typename BufferDetail>
00435 static std::shared_ptr<detail::task>
00436 buffer_add_to_task(BufferDetail buf,
00437                   handler *command_group_handler,
00438                   bool is_write_mode) {
00439     return buf->add_to_task(command_group_handler, is_write_mode);
00440 }
00441
00442 /// @} End the data Doxygen group
00443 }
00444 }
00445 }
00446 }
00447
00448 /*
00449     # Some Emacs stuff:
00450     ### Local Variables:
00451     ###   ispell-local-dictionary: "american"
00452     ###   eval: (flyspell-prog-mode)
00453     ### End:
00454 */
00455
00456 #endif // TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_HPP

```

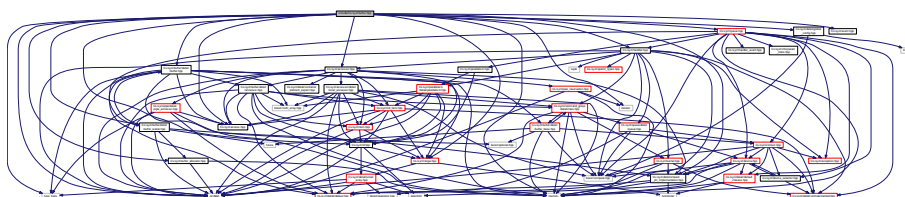
## 11.19 include/CL/sycl/buffer.hpp File Reference

```

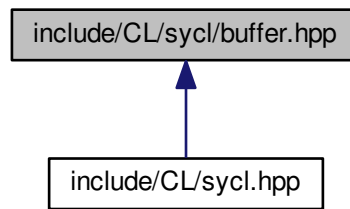
#include <cstddef>
#include <iterator>
#include <memory>
#include <type_traits>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/buffer/detail/buffer.hpp"
#include "CL/sycl/buffer/detail/buffer_waiter.hpp"
#include "CL/sycl/buffer_allocator.hpp"
#include "CL/sycl/detail/global_config.hpp"
#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 >`  
*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 `std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >`

## Namespaces

- `cl`  
*The vector type to be used as SYCL vector.*
- `cl::sycl`
- `std`

## 11.20 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 <cstdint>
00013 #include <iterator>
00014 #include <memory>
00015 #include <type_traits>
00016
00017 #include "CL/sycl/access.hpp"
00018 #include "CL/sycl/accessor.hpp"
00019 #include "CL/sycl/buffer/detail/buffer.hpp"
00020 #include "CL/sycl/buffer/detail/buffer_waiter.hpp"
00021 #include "CL/sycl/buffer_allocator.hpp"
00022 #include "CL/sycl/detail/global_config.hpp"
00023 #include "CL/sycl/detail/shared_ptr_implementation.hpp"
00024 #include "CL/sycl/event.hpp"
00025 #include "CL/sycl/handler.hpp"
00026 #include "CL/sycl/id.hpp"
00027 #include "CL/sycl/queue.hpp"
00028 #include "CL/sycl/range.hpp"
00029
  
```

```

00030 namespace cl {
00031 namespace sycl {
00032
00033 /** \addtogroup data Data access and storage in SYCL
00034     @{
00035 */
00036
00037 /** A SYCL buffer is a multidimensional variable length array (à la C99
00038     VLA or even Fortran before) that is used to store data to work on.
00039
00040     \todo There is a naming inconsistency in the specification between
00041     buffer and accessor on T versus datatype
00042
00043     \todo Finish allocator implementation
00044
00045     \todo Think about the need of an allocator when constructing a buffer
00046     from other buffers
00047
00048     \todo Update the specification to have a non-const allocator for
00049     const buffer? Or do we rely on rebind_alloc<T>. But does this work
00050     with astate-full allocator?
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           int Dimensions = 1,
00059           /* Even a buffer of const T may need to allocate memory, so
00060            need an allocator of non const T */
00061           typename Allocator = buffer_allocator<std::remove_const_t<T>>>
00062 class buffer
00063 /* Use the underlying buffer waiter implementation that can be
00064    shared in the SYCL model */
00065 : public detail::shared_ptr_implementation<
00066     buffer<T, Dimensions, Allocator>,
00067     detail::buffer_waiter<T, Dimensions, Allocator>>,
00068     detail::debug<buffer<T, Dimensions, Allocator>> {
00069 public:
00070
00071     /// The STL-like types
00072     using value_type = T;
00073     using reference = value_type&;
00074     using const_reference = const value_type&;
00075     using allocator_type = Allocator;
00076
00077 private:
00078
00079     // The type encapsulating the implementation
00080     using implementation_t = typename
00081     buffer::shared_ptr_implementation;
00082
00083     // Allows the comparison operation to access the implementation
00084     friend implementation_t;
00085
00086 public:
00087     // Make the implementation member directly accessible in this class
00088     using implementation_t::implementation;
00089
00090     /** Use default constructors so that we can create a new buffer copy
00091         from another one, with either a l-value or an r-value (for
00092         std::move() for example).
00093
00094         Since we just copy the shared_ptr<> from the
00095         shared_ptr_implementation above, this is where/how the sharing
00096         magic is happening with reference counting in this case.
00097     */
00098     buffer() = default;
00099
00100
00101     /** Create a new buffer of the given size with
00102         storage managed by the SYCL runtime
00103
00104         The default behavior is to use the default host buffer
00105         allocator, in order to allow for host accesses. If the type of
00106         the buffer, has the const qualifier, then the default allocator
00107         will remove the qualifier to allow host access to the data.
00108
00109         \param[in] r defines the size
00110
00111         \param[in] allocator is to be used by the SYCL runtime
00112     */
00113     buffer(const range<Dimensions> &r, Allocator allocator = {})
00114         : implementation_t { detail::waiter<T, Dimensions, Allocator>{
00115             new detail::buffer<T, Dimensions> { r } } }

```

```

00116     {}
00117
00118
00119     /** Create a new buffer with associated host memory
00120
00121         \param[in] host_data points to the storage and values used by
00122         the buffer
00123
00124         \param[in] r defines the size
00125
00126         \param[in] allocator is to be used by the SYCL runtime, of type
00127         \c cl::sycl::buffer_allocator<T> by default
00128
00129         The host address is \code const T* \endcode, so the host memory
00130         is read-only.
00131
00132         However, the typename T is not const so the device accesses can
00133         be both read and write accesses. Since, the host_data is const,
00134         this buffer is only initialized with this memory and there is
00135         no write after its destruction, unless there is another final
00136         data address given after construction of the buffer.
00137
00138         Only enable this constructor if it is not the same as the one
00139         with \code const T *host_data \endcode, which is when \c T is
00140         already a constant type.
00141
00142         \todo Actually this is redundant.
00143     */
00144     template <typename Dependent = T,
00145               typename = std::enable_if_t<!std::is_const<Dependent>::value>>
00146     buffer(const T *host_data,
00147            const range<Dimensions> &r,
00148            Allocator allocator = {})
00149     : implementation_t { detail::waiter(new
00150       detail::buffer<T, Dimensions>
00151         { host_data, r }) }
00152     {}
00153
00154     /** Create a new buffer with associated host memory
00155
00156         \param[inout] host_data points to the storage and values used by
00157         the buffer
00158
00159         \param[in] r defines the size
00160
00161         \param[in] allocator is to be used by the SYCL runtime, of type
00162         cl::sycl::buffer_allocator<T> by default
00163
00164         The memory is owned by the runtime during the lifetime of the
00165         object. Data is copied back to the host unless the user
00166         overrides the behavior using the set_final_data method. host_data
00167         points to the storage and values used by the buffer and
00168         range<Dimensions> defines the size.
00169     */
00170     buffer(T *host_data,
00171            const range<Dimensions> &r,
00172            Allocator allocator = {})
00173     : implementation_t { detail::waiter(new
00174       detail::buffer<T, Dimensions>
00175         { host_data, r }) }
00176     {}
00177
00178     /** Create a new buffer with associated memory, using the data in
00179     host_data
00180
00181         \param[inout] host_data points to the storage and values used by
00182         the buffer
00183
00184         \param[in] r defines the size
00185
00186         \param[in] allocator is to be used by the SYCL runtime, of type
00187         cl::sycl::buffer_allocator<T> by default
00188
00189         The ownership of the host_data is shared between the runtime and the
00190         user. In order to enable both the user application and the SYCL
00191         runtime to use the same pointer, a cl::sycl::mutex_class is
00192         used. The mutex m is locked by the runtime whenever the data is in
00193         use and unlocked otherwise. Data is synchronized with host_data, when
00194         the mutex is unlocked by the runtime.
00195
00196         \todo update the specification to replace the pointer by a
00197         reference and provide the constructor with and without a mutex
00198     */
00199     buffer(shared_ptr_class<T> &host_data,
00200            const range<Dimensions> &buffer_range,

```

```

00201         cl::sycl::mutex_class &m,
00202         Allocator allocator = {}) {
00203     detail::unimplemented();
00204 }
00205
00206
00207 /** Create a new buffer with associated memory, using the data in
00208     host_data
00209
00210     \param[inout] host_data points to the storage and values used by
00211     the buffer
00212
00213     \param[in] r defines the size
00214
00215     \param[inout] m is the mutex used to protect the data access
00216
00217     \param[in] allocator is to be used by the SYCL runtime, of type
00218     cl::sycl::buffer_allocator<T> by default
00219
00220     The ownership of the host_data is shared between the runtime and the
00221     user. In order to enable both the user application and the SYCL
00222     runtime to use the same pointer, a cl::sycl::mutex_class is
00223     used.
00224
00225     \todo add this mutex-less constructor to the specification
00226 */
00227 buffer(shared_ptr_class<T> host_data,
00228         const range<Dimensions> &buffer_range,
00229         Allocator allocator = {})
00230 : implementation_t { detail::waiter(new
detail::buffer<T, Dimensions>
00231         { host_data, buffer_range }) }
00232 {}
00233
00234
00235 /** Create a new allocated 1D buffer initialized from the given
00236     elements ranging from first up to one before last
00237
00238     The data is copied to an intermediate memory position by the
00239     runtime. Data is written back to the same iterator set if the
00240     iterator is not a const iterator.
00241
00242     \param[inout] start_iterator points to the first element to copy
00243
00244     \param[in] end_iterator points to just after the last element to copy
00245
00246     \param[in] allocator is to be used by the SYCL runtime, of type
00247     cl::sycl::buffer_allocator<T> by default
00248
00249     \todo Implement the copy back at buffer destruction
00250
00251     \todo Generalize this for n-D and provide column-major and row-major
00252     initialization
00253
00254     \todo a reason to have this nD is that
00255     set_final_data(weak_ptr_class<T> & finalData) is actually
00256     doing this linearization anyway
00257
00258     \todo Allow read-only buffer construction too
00259
00260     \todo update the specification to deal with forward iterators
00261     instead and rewrite back only when it is non const and output
00262     iterator at least
00263
00264     \todo Allow initialization from ranges and collections à la STL
00265 */
00266 template <typename InputIterator,
00267         /* To force some iterator concept checking to avoid GCC 4.9
00268         diving into this when initializing from ({ int, int })
00269         which is a range<> and not an iterator... */
00270         typename ValueType =
00271         typename std::iterator_traits<InputIterator>::value_type>
00272 buffer(InputIterator start_iterator,
00273         InputIterator end_iterator,
00274         Allocator allocator = {}) :
00275     implementation_t { detail::waiter(new
detail::buffer<T, Dimensions>
00276         { start_iterator, end_iterator }) }
00277 {}
00278
00279
00280 /** Create a new sub-buffer without allocation to have separate
00281     accessors later
00282
00283     \param[inout] b is the buffer with the real data
00284
00285     \param[in] base_index specifies the origin of the sub-buffer inside the

```

```

00286     buffer b
00287
00288     \param[in] sub_range specifies the size of the sub-buffer
00289
00290     \todo To be implemented
00291
00292     \todo Update the specification to replace index by id
00293 */
00294 buffer(buffer<T, Dimensions, Allocator> &b,
00295         const id<Dimensions> &base_index,
00296         const range<Dimensions> &sub_range,
00297         Allocator allocator = {}) { detail::unimplemented(); }
00298
00299 #ifndef TRISYCL_OPENCL
00300 /** Create a buffer from an existing OpenCL memory object associated
00301     with a context after waiting for an event signaling the
00302     availability of the OpenCL data
00303
00304     \param[inout] mem_object is the OpenCL memory object to use
00305
00306     \param[inout] from_queue is the queue associated to the memory
00307     object
00308
00309     \param[in] available_event specifies the event to wait for if
00310     non null
00311
00312     Note that a buffer created from a cl_mem object will only have
00313     one underlying cl_mem for the lifetime of the buffer and use on
00314     an incompatible queue constitutes an error.
00315
00316     \todo To be implemented
00317
00318     \todo Improve the specification to allow CLHPP objects too
00319 */
00320 buffer(cl_mem mem_object,
00321         queue from_queue,
00322         event available_event = {},
00323         Allocator allocator = {}) { detail::unimplemented(); }
00324 #endif
00325
00326 // Use BOOST_DISABLE_ASSERTS at some time to disable range checking
00327
00328 /** Get an accessor to the buffer with the required mode
00329
00330     \param Mode is the requested access mode
00331
00332     \param Target is the type of object to be accessed
00333
00334     \param[in] command_group_handler is the command group handler in
00335     which the kernel is to be executed
00336
00337     \todo Do we need for an accessor to increase the reference count of
00338     a buffer object? It does make more sense for a host-side accessor.
00339
00340     \todo Implement the modes and targets
00341 */
00342 template <access::mode Mode,
00343           access::target Target = access::target::global_buffer>
00344
00345 >
00346 accessor<T, Dimensions, Mode, Target>
00347 get_access(handler &command_group_handler) {
00348     static_assert(Target == access::target::global_buffer
00349         || Target == access::target::constant_buffer,
00350         "get_access(handler) can only deal with access::global_buffer"
00351         " or access::constant_buffer (for host_buffer accessor)"
00352         " do not use a command group handler");
00353     implementation->implementation->template track_access_mode<Mode, Target>();
00354     return { *this, command_group_handler };
00355 }
00356
00357 /** Force the buffer to behave like if we had created
00358     an accessor in write mode.
00359 */
00360 void mark_as_written() {
00361     return implementation->implementation->mark_as_written();
00362 }
00363
00364 /** Get a host accessor to the buffer with the required mode
00365
00366     \param Mode is the requested access mode
00367
00368     \todo Implement the modes
00369
00370
00371

```

```

00372     \todo More elegant solution
00373 */
00374 template <access::mode Mode,
00375           access::target Target = access::target::host_buffer>
00376 accessor<T, Dimensions, Mode, Target>
00377 get_access() {
00378     static_assert(Target == access::target::host_buffer,
00379                   "get_access() without a command group handler is only"
00380                   " for host_buffer accessor");
00381     implementation->implementation->template track_access_mode<Mode, Target>();
00382     return { *this };
00383 }
00384
00385
00386 /** Return a range object representing the size of the buffer in
00387     terms of number of elements in each dimension as passed to the
00388     constructor
00389
00390     \todo rename to the equivalent from array_ref proposals? Such
00391     as size() in
00392     http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0009r2.html
00393 */
00394 auto get_range() const {
00395     /* Interpret the shape which is a pointer to the first element as an
00396        array of Dimensions elements so that the range<Dimensions>
00397        constructor is happy with this collection
00398     */
00399     return implementation->implementation->get_range();
00400 }
00401
00402
00403 /** Returns the total number of elements in the buffer
00404
00405     Equal to get_range()[0] * ... * get_range()[Dimensions-1].
00406 */
00407 auto get_count() const {
00408     return implementation->implementation->get_count();
00409 }
00410
00411
00412 /** Returns the size of the buffer storage in bytes
00413
00414     Equal to get_count()*sizeof(T).
00415
00416     \todo rename to something else. In
00417     http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf
00418     it is named bytes() for example
00419 */
00420 size_t get_size() const {
00421     return implementation->implementation->get_size();
00422 }
00423
00424
00425 /** Returns the number of buffers that are shared/referenced
00426
00427     For example
00428     \code
00429     cl::sycl::buffer<int> b { 1000 };
00430     // Here b.use_count() should return 1
00431     cl::sycl::buffer<int> c { b };
00432     // Here b.use_count() and c.use_count() should return 2
00433     \endcode
00434
00435     \todo Add to the specification, useful for validation
00436 */
00437 auto use_count() const {
00438     // Rely on the shared_ptr<> use_count()
00439     return implementation.use_count();
00440 }
00441
00442
00443 /** Ask for read-only status of the buffer
00444
00445     \todo Add to specification
00446 */
00447 bool constexpr is_read_only() const {
00448     return std::is_const<T>::value;
00449 }
00450
00451
00452 /** Set destination of buffer data on destruction
00453
00454     The finalData points to the host memory to which, the outcome of all
00455     the buffer processing is going to be copied to.
00456
00457     This is the final pointer, which is going to be accessible after the
00458     destruction of the buffer and in the case where this is a valid

```

```

00459     pointer, the data are going to be copied to this host address.
00460
00461     finalData is different from the original host address, if the buffer
00462     was created associated with one. This is mainly to be used when a
00463     shared_ptr is given in the constructor and the output data will
00464     reside in a different location from the initialization data.
00465
00466     It is defined as a weak_ptr referring to a shared_ptr that is not
00467     associated with the cl::sycl::buffer, and so the cl::sycl::buffer
00468     will have no ownership of finalData.
00469
00470     \todo Update the API to take finalData by value instead of by
00471           reference. This way we can have an implicit conversion
00472           possible at the API call from a shared_ptr<>, avoiding an
00473           explicit weak_ptr<> creation
00474
00475     \todo figure out how set_final_data() interact with the other
00476           way to write back some data or with some data sharing with the
00477           host that can not be undone
00478 */
00479 void set_final_data(shared_ptr_class<T> finalData) {
00480     implementation->implementation->set_final_data(std::move(finalData));
00481 }
00482
00483 /** Set destination of buffer data on destruction.
00484 */
00485 void set_final_data(weak_ptr_class<T> finalData) {
00486     implementation->implementation->set_final_data(std::move(finalData));
00487 }
00488
00489 /** Disable write-back on buffer destruction.
00490 */
00491 void set_final_data(std::nullptr_t) {
00492     implementation->implementation->set_final_data(nullptr);
00493 }
00494
00495 #ifndef TRISYCL_OPENCL
00496 /** Check if the buffer is already cached in a certain context
00497 */
00498 bool is_cached(cl::sycl::context& ctx) {
00499     return implementation->implementation->is_cached(ctx);
00500 }
00501
00502 /** Check if the data stored in the buffer is up-to-date in a certain context
00503 */
00504 bool is_data_up_to_date(cl::sycl::context& ctx) {
00505     return implementation->implementation->is_data_up_to_date(ctx);
00506 }
00507 #endif
00508
00509 /** Set destination of buffer data on destruction.
00510
00511     WARNING: the user has to ensure that the object referred to by the
00512     iterator will be alive after buffer destruction, otherwise the behaviour
00513     is undefined.
00514 */
00515 template<typename Iterator>
00516 void set_final_data(Iterator&& finalData) {
00517     implementation->implementation->
00518         set_final_data(std::forward<Iterator>(finalData));
00519 }
00520
00521 };
00522
00523 /// @} End the data Doxygen group
00524
00525 }
00526
00527 /* Inject a custom specialization of std::hash to have the buffer
00528    usable into an unordered associative container
00529
00530    \todo Add this to the spec
00531 */
00532 namespace std {
00533     template <typename T,
00534              int Dimensions,
00535              typename Allocator>
00536     struct hash<cl::sycl::buffer<T, Dimensions, Allocator>> {
00537         auto operator() (const cl::sycl::buffer<T, Dimensions, Allocator>
00538             &b) const {

```



```

00545     // Forward the hashing to the implementation
00546     return b.hash();
00547 }
00548 };
00549 };
00550
00551 }
00552
00553 /*
00554  # Some Emacs stuff:
00555  ### Local Variables:
00556  ### ispell-local-dictionary: "american"
00557  ### eval: (flyspell-prog-mode)
00558  ### End:
00559 */
00560
00561 #endif // TRISYCL_SYCL_BUFFER_HPP

```

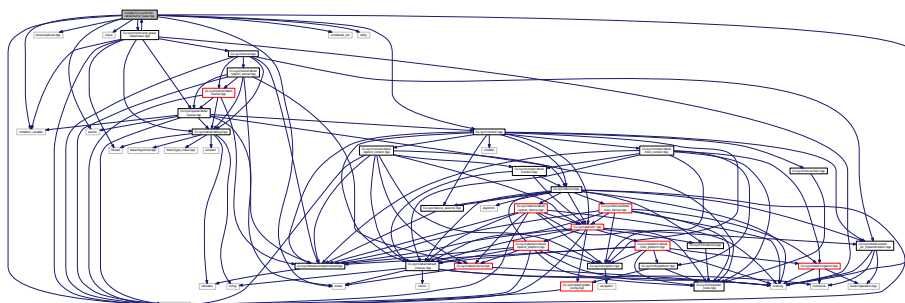
## 11.21 include/CL/sycl/buffer/detail/buffer\_base.hpp File Reference

```

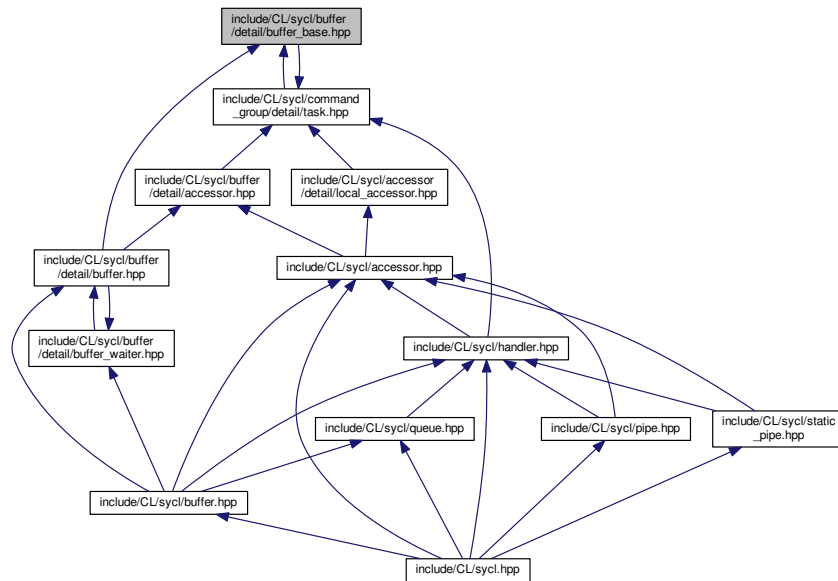
#include <atomic>
#include <boost/compute.hpp>
#include <boost/optional.hpp>
#include <condition_variable>
#include <future>
#include <memory>
#include <mutex>
#include <unordered_set>
#include <utility>
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/context.hpp"

```

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. [More...](#)*

## 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)

## 11.22 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 #ifdef TRISYCL_OPENCL
00015 #include <boost/compute.hpp>
00016 #endif
00017 // \todo Use C++17 optional when it is mainstream
00018 #include <boost/optional.hpp>
00019 #include <condition_variable>
00020 #include <future>
00021 #include <memory>
00022 #include <mutex>
00023 #include <unordered_set>
00024 #include <utility>
00025
00026 #include "CL/sycl/command_group/detail/task.hpp"
00027 #include "CL/sycl/context.hpp"
00028
00029 namespace cl {
00030 namespace sycl {
00031
00032 class handler;
00033
00034 namespace detail {
00035
00036 /** \addtogroup data Data access and storage in SYCL
00037     @{
00038 */
00039
00040 struct task;
00041 struct buffer_base;
00042 inline static std::shared_ptr<detail::task>
00043 add_buffer_to_task(handler *command_group_handler,
00044                   std::shared_ptr<detail::buffer_base> b,
00045                   bool is_write_mode);
00046
00047 /** Factorize some template independent buffer aspects in a base class
00048 */
00049 struct buffer_base : public std::enable_shared_from_this<buffer_base> {
00050
00051     /// Keep track of the number of kernel accessors using this buffer
00052     std::atomic<size_t> number_of_users;
00053
00054     /// Track the latest task to produce this buffer
00055     std::weak_ptr<detail::task> latest_producer;
00056     /// To protect the access to latest_producer
00057     std::mutex latest_producer_mutex;
00058
00059     /// To signal when this buffer ready
00060     std::condition_variable ready;
00061     /// To protect the access to the condition variable
00062     std::mutex ready_mutex;
00063
00064     /** If the SYCL user buffer destructor is blocking, use this to
00065         block until this buffer implementation is destroyed.
00066
00067         Use a void promise since there is no value to send, only
00068         waiting */
00069     boost::optional<std::promise<void>> notify_buffer_destructor;
00070
00071     /// To track contexts in which the data is up-to-date
00072     std::unordered_set<cl::sycl::context> fresh_ctx;
00073
00074 #ifdef TRISYCL_OPENCL
00075     /** Buffer-side cache that keeps the \c boost::compute::buffer (and the
00076         underlying \c cl_buffer ) so that if the buffer already exists inside
00077         the same context it is not recreated.
00078     */
00079     std::unordered_map<cl::sycl::context, boost::compute::buffer> buffer_cache;
00080 #endif
00081
00082     /** Create a buffer base and marks the host context as the context that
00083         holds the most recent version of the data
00084         \todo Use lazy allocation for the context tracking set
00085     */
00086     buffer_base() : number_of_users { 0 },
00087                   fresh_ctx { cl::sycl::context {} } {}
00088
00089     /// The destructor wait for not being used anymore
00090     ~buffer_base() {
00091         wait();
00092         // If there is the last SYCL user buffer waiting, notify it
00093         if (notify_buffer_destructor)
00094             notify_buffer_destructor->set_value();
00095     }
00096
00097
00098

```

```

00099  /// Wait for this buffer to be ready, which is no longer in use
00100  void wait() {
00101      std::unique_lock<std::mutex> ul { ready_mutex };
00102      ready.wait(ul, [&] {
00103          // When there is no producer for this buffer, we are ready to use it
00104          return number_of_users == 0;
00105      });
00106  }
00107
00108
00109  /// Mark this buffer in use by a task
00110  void use() {
00111      // Increment the use count
00112      ++number_of_users;
00113  }
00114
00115
00116  /// A task has released the buffer
00117  void release() {
00118      if (--number_of_users == 0)
00119          // Notify the host consumers or the buffer destructor that it is ready
00120          ready.notify_all();
00121  }
00122
00123
00124  /// Return the latest producer for the buffer
00125  std::shared_ptr<detail::task> get_latest_producer() {
00126      std::lock_guard<std::mutex> lg { latest_producer_mutex };
00127      // Return the valid shared_ptr to the task, if any
00128      return latest_producer.lock();
00129  }
00130
00131
00132  /** Return the latest producer for the buffer and set another
00133      future producer
00134  */
00135  std::shared_ptr<detail::task>
00136  set_latest_producer(std::weak_ptr<detail::task> newer_latest_producer) {
00137      std::lock_guard<std::mutex> lg { latest_producer_mutex };
00138      using std::swap;
00139
00140      swap(newer_latest_producer, latest_producer);
00141      // Return the valid shared_ptr to the previous producing task, if any
00142      return newer_latest_producer.lock();
00143  }
00144
00145
00146  /// Add a buffer to the task running the command group
00147  std::shared_ptr<detail::task>
00148  add_to_task(handler *command_group_handler, bool is_write_mode) {
00149      return add_buffer_to_task(command_group_handler,
00150          shared_from_this(),
00151          is_write_mode);
00152  }
00153
00154
00155  #ifndef TRISYCL_OPENCL
00156      /// Check if the data of this buffer is up-to-date in a certain context
00157      bool is_data_up_to_date(const cl::sycl::context& ctx) {
00158          return fresh_ctx.count(ctx);
00159      }
00160
00161
00162      /// Check if the buffer is already cached for a certain context
00163      bool is_cached(const cl::sycl::context& ctx) {
00164          return buffer_cache.count(ctx);
00165      }
00166
00167
00168      /** Create a \c boost::compute::buffer for this \c cl::sycl::buffer in the
00169          cache and associate it with a given context
00170      */
00171      void create_in_cache(const cl::sycl::context& ctx, size_t size,
00172          cl_mem_flags flags, void* data) {
00173          buffer_cache[ctx] = boost::compute::buffer
00174              { ctx.get_boost_compute(),
00175                size,
00176                flags,
00177                data
00178              };
00179      }
00180
00181
00182      /** Transfer the most up-to-date version of the data to the host
00183          if the host version is not already up-to-date
00184      */
00185      void sync_with_host(std::size_t size, void* data) {

```

```

00186     cl::sycl::context host_context;
00187     if (!is_data_up_to_date(host_context) && !fresh_ctx.empty()) {
00188         /* We know that the context(s) in \c fresh_ctx hold the most recent
00189            version of the buffer
00190            */
00191         auto fresh_context = *(fresh_ctx.begin());
00192         auto fresh_q = fresh_context.get_boost_queue();
00193         fresh_q.enqueue_read_buffer(buffer_cache[fresh_context], 0, size, data);
00194         fresh_ctx.insert(host_context);
00195     }
00196 }
00197
00198 /** When a transfer is requested this function is called, it will
00199     update the state of the buffer according to the context in which
00200     the accessor is created and the access mode
00201     */
00202 void update_buffer_state(const cl::sycl::context& target_ctx,
00203                         access::mode mode, std::size_t size, void* data) {
00204     /* The \c cl_buffer we put in the cache might get accessed again in the
00205        future, this means that we have to always create it in read/write
00206        mode to be able to write to it if it is accessed through a
00207        write accessor in the future
00208        */
00209     auto constexpr flag = CL_MEM_READ_WRITE;
00210
00211     /* The buffer is accessed in read mode, we want to transfer the data only if
00212        necessary. We start a transfer if the data on the target context is not
00213        up to date and then update the fresh context set.
00214        */
00215     if (mode == access::mode::read) {
00216         if (is_data_up_to_date(target_ctx))
00217             // If read mode and the data is up-to-date there is nothing to do
00218             return;
00219
00220         // The data is not up-to-date, we need a transfer
00221         // We also want to be sure that the host holds the most recent data
00222         sync_with_host(size, data);
00223
00224         if (!target_ctx.is_host()) {
00225             // If the target context is a device context
00226             if (!is_cached(target_ctx)) {
00227                 /* If not cached, we create the buffer and copy the data
00228                    at the same time
00229                    */
00230                 create_in_cache(target_ctx, size,
00231                                (flag | CL_MEM_COPY_HOST_PTR), data);
00232                 fresh_ctx.insert(target_ctx);
00233                 return;
00234             }
00235
00236             /* Else we transfer the data to the existing buffer associated
00237                with the target context buffer
00238                */
00239             auto q = target_ctx.get_boost_queue();
00240             q.enqueue_write_buffer(buffer_cache[target_ctx], 0, size, data);
00241             fresh_ctx.insert(target_ctx);
00242         }
00243         return;
00244     }
00245
00246     /* The buffer might be written to, this means that we have to consider
00247        every version of the data obsolete except in the target context
00248
00249        We go through the same process as in read mode but in addition
00250        we empty the fresh context set and just add the target context
00251
00252        If the data is up to date on the target we just have to update
00253        the context set and nothing else
00254        */
00255     if (!is_data_up_to_date(target_ctx)) {
00256         if (mode == access::mode::read_write
00257             || mode == access::mode::write
00258             || mode == access::mode::atomic) {
00259             // If the data is not up-to-date in the target context
00260             // We want to host to be up-to-date
00261             sync_with_host(size, data);
00262
00263             if (!target_ctx.is_host()) {
00264                 // If the target context is a device context
00265                 if (!is_cached(target_ctx)) {
00266                     create_in_cache(target_ctx, size,
00267                                    (flag | CL_MEM_COPY_HOST_PTR), data);
00268                 }
00269             }
00270             else {

```

```

00273         // We update the buffer associated with the target context
00274         auto q = target_ctx.get_boost_queue();
00275         q.enqueue_write_buffer(buffer_cache[target_ctx], 0, size, data);
00276     }
00277 }
00278 }
00279
00280 /* When in discard mode we don't need to transfer any data, we just create
00281    the \c cl_buffer if it doesn't exist in the cache
00282 */
00283 if ( mode == access::mode::discard_write
00284     || mode == access::mode::discard_read_write) {
00285     /* We only need to create the buffer if it doesn't exist
00286        but without copying any data because of the discard mode
00287     */
00288     if (!target_ctx.is_host() && !is_cached(target_ctx)) {
00289         // If the context doesn't exist we create it.
00290         /* We don't want to transfer any data so we don't
00291            add \c CL_MEM_COPY_HOST_PTR
00292         */
00293         create_in_cache(target_ctx, size, flag, 0);
00294     }
00295 }
00296 }
00297 /* Here we are sure that we are in some kind of write mode,
00298    we indicate that all contexts except the target context
00299    are not up-to-date anymore
00300 */
00301 fresh_ctx.clear();
00302 fresh_ctx.insert(target_ctx);
00303 }
00304
00305
00306 /// Returns the cl_buffer for a given context.
00307 boost::compute::buffer get_cl_buffer(const cl::sycl::context&
00308 context) {
00309     return buffer_cache[context];
00310 }
00311 #endif
00312 };
00313 };
00314
00315 /// @} End the data Doxygen group
00316
00317 }
00318 }
00319 }
00320
00321 /*
00322    # Some Emacs stuff:
00323    ### Local Variables:
00324    ### ispell-local-dictionary: "american"
00325    ### eval: (flyspell-prog-mode)
00326    ### End:
00327 */
00328
00329 #endif // TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_BASE_HPP

```

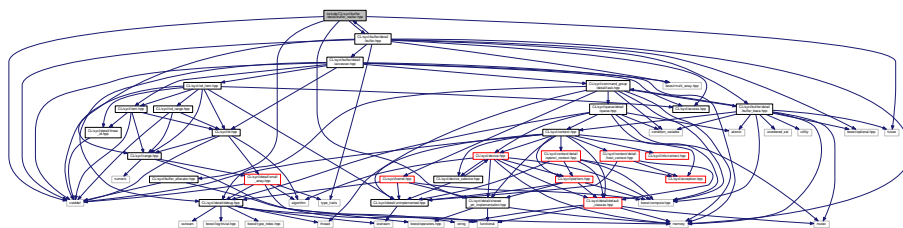
## 11.23 include/CL/sycl/buffer/detail/buffer\_waiter.hpp File Reference

```

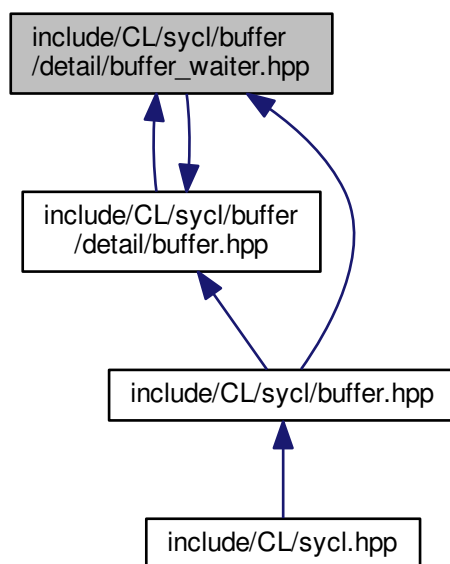
#include <cstdint>
#include <future>
#include "CL/sycl/buffer/detail/buffer.hpp"
#include "CL/sycl/buffer_allocator.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"

```

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, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_const_t<T>>>>`  
`auto cl::sycl::detail::waiter (detail::buffer< T, Dimensions > *b)`

*Helper function to create a new [buffer\\_waiter](#).*

## 11.24 `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 <cstdint>
00013 #include <future>
00014
00015 #include "CL/sycl/buffer/detail/buffer.hpp"
00016 #include "CL/sycl/buffer_allocator.hpp"
00017 #include "CL/sycl/detail/shared_ptr_implementation.hpp"
00018
00019 namespace cl {
00020 namespace sycl {
00021 namespace detail {
00022
00023 /** \addtogroup data Data access and storage in SYCL
00024     @{
00025 */
00026
00027 /** A helper class to wait for the final buffer destruction if the
00028     conditions for blocking are met
00029 */
00030 template <typename T,
00031           int Dimensions = 1,
00032           typename Allocator = buffer_allocator<std::remove_const_t<T>>>>
00033 class buffer_waiter :
00034     public detail::shared_ptr_implementation<buffer_waiter<T,
00035                                             Dimensions,
00036                                             Allocator>,
00037                                             detail::buffer<T, Dimensions>>,
00038     detail::debug<buffer_waiter<T, Dimensions, Allocator>> {
00039
00040     // The type encapsulating the implementation
00041     using implementation_t = typename
00042     buffer_waiter::shared_ptr_implementation;
00043
00044     // Allows the comparison operation to access the implementation
00045     friend implementation_t;
00046
00047 public:
00048     // Make the implementation member directly accessible in this class
00049     using implementation_t::implementation;
00050
00051     /// Create a new buffer_waiter on top of a detail::buffer
00052     buffer_waiter(detail::buffer<T, Dimensions> *b) :
00053     implementation_t { b } {}
00054
00055     /** The buffer_waiter destructor waits for any data to be written
00056         back to the host, if any
00057     */
00058     ~buffer_waiter() {
00059         /* Get a future from the implementation if we have to wait for its
00060             destruction */
00061         auto f = implementation->get_destructor_future();
00062         if (f) {
00063             /* No longer carry for the implementation buffer which is free to
00064                 live its life up to its destruction */
00065             implementation.reset();
00066             TRISYCL_DUMP_T("~buffer_waiter() is waiting");
00067             // Then wait for its end in some other thread
00068             f->wait();
00069             TRISYCL_DUMP_T("~buffer_waiter() is done");

```



```

00070     }
00071   }
00072 };
00073
00074
00075 /// Helper function to create a new buffer_waiter
00076 template <typename T,
00077           int Dimensions = 1,
00078           typename Allocator = buffer_allocator<std::remove_const_t<T>
00079 >>
00079 inline auto waiter(detail::buffer<T, Dimensions> *b) {
00080   return new buffer_waiter<T, Dimensions, Allocator> { b };
00081 }
00082
00083 /// @} End the data Doxygen group
00084
00085 }
00086 }
00087 }
00088
00089 /*
00090  # Some Emacs stuff:
00091  ### Local Variables:
00092  ###   ispell-local-dictionary: "american"
00093  ###   eval: (flyspell-prog-mode)
00094  ### End:
00095  */
00096
00097 #endif // TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_WAITER_HPP

```

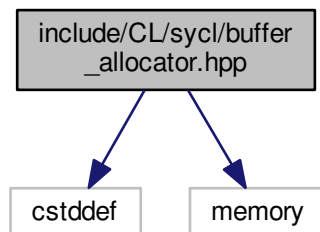
## 11.25 include/CL/sycl/buffer\_allocator.hpp File Reference

```

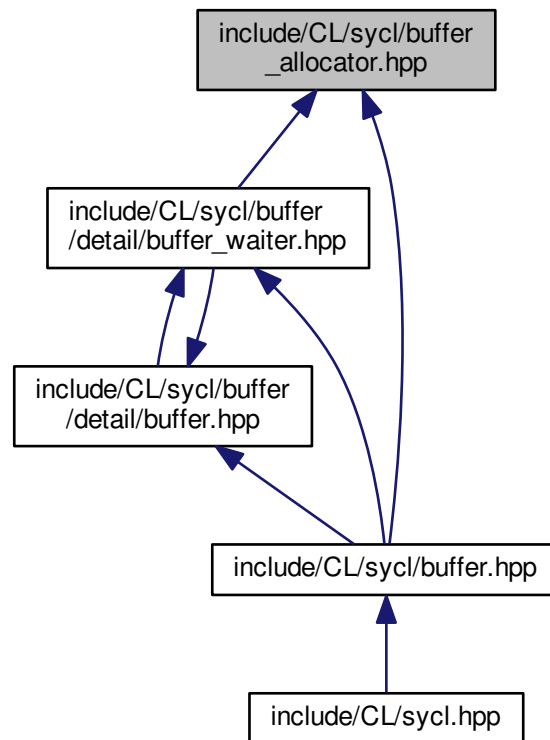
#include <cstdint>
#include <memory>

```

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](#)

## 11.26 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

```

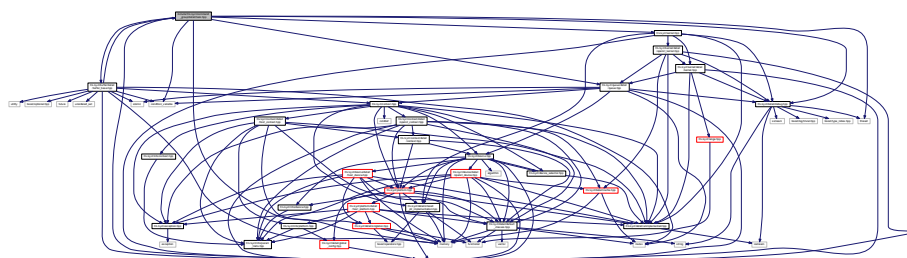
## 11.27 include/CL/sycl/command\_group/detail/task.hpp File Reference

```

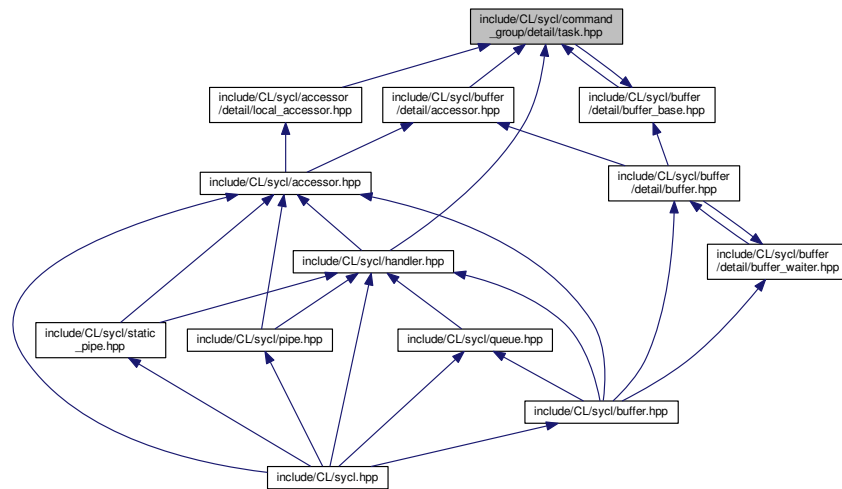
#include <condition_variable>
#include <memory>
#include <thread>
#include <boost/compute.hpp>
#include "CL/sycl/buffer/detail/buffer_base.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/kernel.hpp"
#include "CL/sycl/queue/detail/queue.hpp"

```

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`

## 11.28 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 #ifndef TRISYCL_NO_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     {
00146         std::unique_lock<std::mutex> ul { ready_mutex };
00147         execution_ended = true;
00148     }
00149     /* \todo Verify that the memory model with the notify does not
00150        require some fence or atomic */
00151     ready.notify_all();
00152 }
00153
00154
00155     /** Wait for this task to be ready
00156
00157         This is to be called from another thread
00158     */
00159 void wait() {
00160     TRISYCL_DUMP_T("The task wait for task " << this << " to end");
00161     std::unique_lock<std::mutex> ul { ready_mutex };
00162     ready.wait(ul, [&] { return execution_ended; });
00163 }
00164
00165
00166     /** Register a buffer to this task
00167
00168         This is how the dependency graph is incrementally built.
00169     */
00170 void add_buffer(std::shared_ptr<detail::buffer_base> &buf,
00171                bool is_write_mode) {
00172     TRISYCL_DUMP_T("Add buffer " << buf << " in task " << this);
00173     /* Keep track of the use of the buffer to notify its release at
00174        the end of the execution */
00175     buffers_in_use.push_back(buf);
00176     // To be sure the buffer does not disappear before the kernel can run
00177     buf->use();
00178
00179     std::shared_ptr<detail::task> latest_producer;
00180     if (is_write_mode) {
00181         /* Set this task as the latest producer of the buffer so that
00182            another kernel may wait on this task */
00183         latest_producer = buf->set_latest_producer(shared_from_this());
00184     }
00185     else
00186         latest_producer = buf->get_latest_producer();
00187
00188     /* If the buffer is to be produced by a task, add the task in the
00189        producer list to wait on it before running the task core
00190
00191        If a buffer is accessed first in write mode and then in read mode,
00192        the task will add itself as a producer and will wait for itself
00193        when calling \c wait_for_producers, we avoid this by checking that
00194        \c latest_producer is not \c this
00195     */
00196     if (latest_producer && latest_producer != shared_from_this())
00197         producer_tasks.push_back(latest_producer);
00198 }
00199

```

```

00200
00201     /// Execute the prologues
00202     void prelude() {
00203         for (const auto &p : prologues)
00204             p();
00205         /* Free the functors that may own an accessor owning a buffer
00206            preventing the command group to complete */
00207         prologues.clear();
00208     }
00209
00210
00211     /// Execute the epilogues
00212     void postlude() {
00213         for (const auto &p : epilogues)
00214             p();
00215         /* Free the functors that may own an accessor owning a buffer
00216            preventing the command group to complete */
00217         epilogues.clear();
00218     }
00219
00220
00221     /// Add a function to the prelude to run before kernel execution
00222     void add_prelude(const std::function<void(void)> &f) {
00223         prologues.push_back(f);
00224     }
00225
00226
00227     /// Add a function to the postlude to run after kernel execution
00228     void add_postlude(const std::function<void(void)> &f) {
00229         epilogues.push_back(f);
00230     }
00231
00232
00233     /// Get the queue behind the task to run a kernel on
00234     auto get_queue() {
00235         return owner_queue;
00236     }
00237
00238
00239     /// Set the kernel running this task if any
00240     void set_kernel(const std::shared_ptr<cl::sycl::detail::kernel> &k) {
00241         kernel = k;
00242     }
00243
00244
00245     /** Get the kernel running if any
00246
00247         \todo Specify this error in the spec
00248     */
00249     cl::sycl::detail::kernel &get_kernel() {
00250         if (!kernel)
00251             throw non_cl_error("Cannot use an OpenCL kernel in this context");
00252         return *kernel;
00253     }
00254
00255 };
00256
00257 }
00258 }
00259 }
00260
00261 /*
00262     # Some Emacs stuff:
00263     ### Local Variables:
00264     ### ispell-local-dictionary: "american"
00265     ### eval: (flyspell-prog-mode)
00266     ### End:
00267 */
00268
00269 #endif // TRISYCL_SYCL_TASK_HPP

```

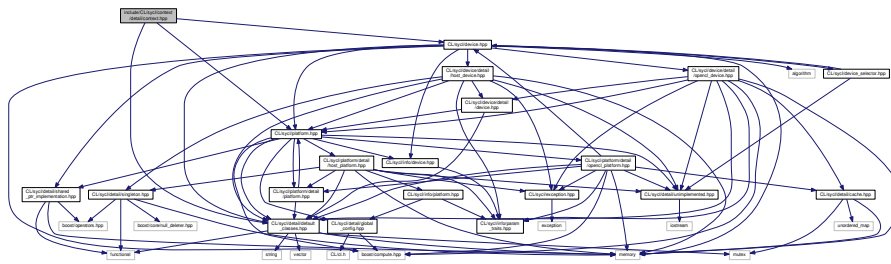
## 11.29 include/CL/sycl/context/detail/context.hpp File Reference

```

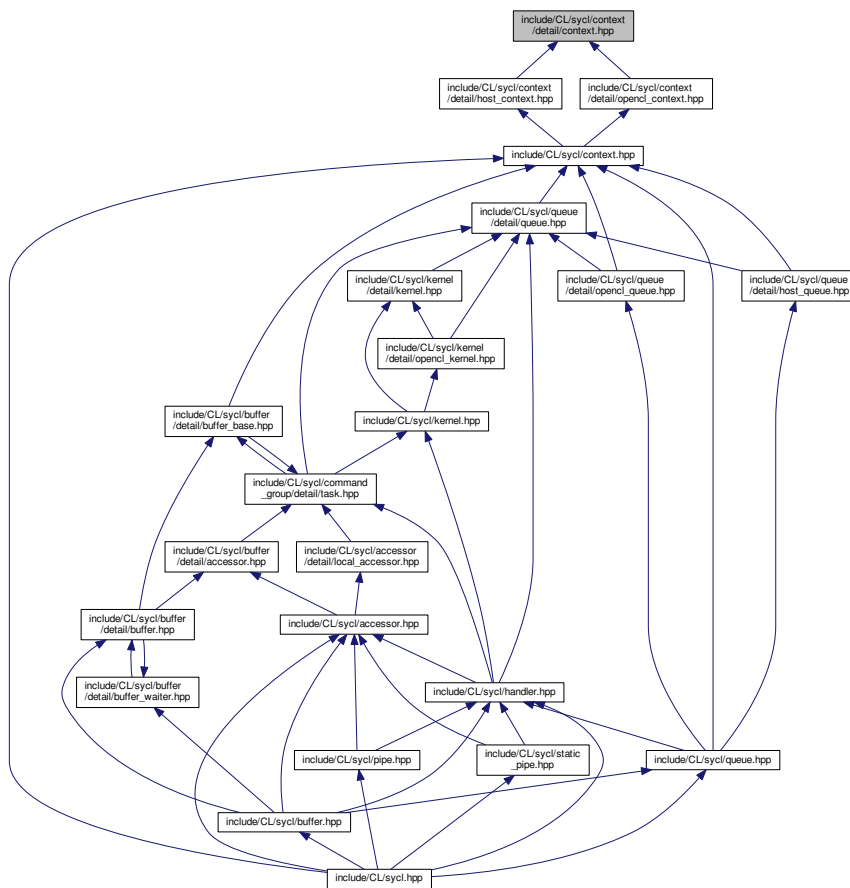
#include "CL/sycl/device.hpp"
#include "CL/sycl/platform.hpp"
#include "CL/sycl/detail/default_classes.hpp"

```

Include dependency graph for context.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class `cl::sycl::detail::context`

## Namespaces

- `cl`  
*The vector type to be used as SYCL vector.*
- `cl::sycl`
- `cl::sycl::detail`



## 11.30 context.hpp

```

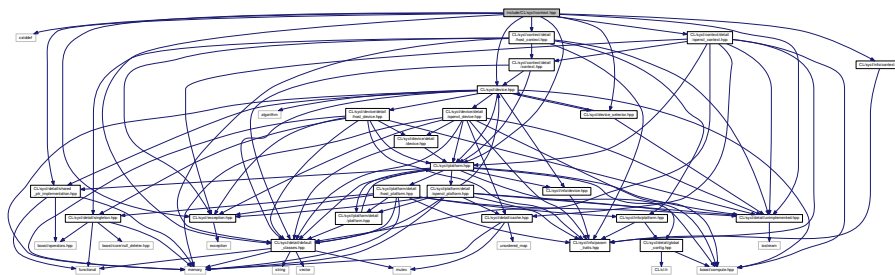
00001 #ifndef TRISYCL_SYCL_CONTEXT_DETAIL_CONTEXT_HPP
00002 #define TRISYCL_SYCL_CONTEXT_DETAIL_CONTEXT_HPP
00003
00004 /** \file The OpenCL SYCL abstract context
00005
00006     a-doumoulakis at gmail dot com (Anastasios Doumoulakis)
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include "CL/sycl/device.hpp"
00013 #include "CL/sycl/platform.hpp"
00014 #include "CL/sycl/detail/default_classes.hpp"
00015
00016 namespace cl {
00017 namespace sycl {
00018 namespace detail {
00019
00020     /** \addtogroup execution Platforms, contexts, devices and queues
00021     @{
00022     */
00023
00024     class context {
00025     public:
00026
00027     #ifdef TRISYCL_OPENCL
00028         /// Return the underlying \c cl_context of the \c cl::sy::context
00029         virtual cl_context get() const = 0;
00030
00031         /** Return the underlying \c boost::compute::context
00032         of the \c cl::sy::context
00033         */
00034         virtual boost::compute::context &get_boost_compute() = 0;
00035
00036         /** Return the underlying \c boost::compute::command_queue associated
00037         with the context
00038         */
00039         virtual boost::compute::command_queue &get_boost_queue() = 0;
00040     #endif
00041
00042         /// Returns true is the context is a SYCL host context
00043         virtual bool is_host() const = 0;
00044
00045         /// Returns the SYCL platform that the context is initialized for
00046         virtual cl::sy::platform get_platform() const = 0;
00047
00048         /** \todo virtual cannot be templated
00049         template <info::context Param>
00050         typename info::param_traits<info::context, Param>::type
00051         get_info() const = 0;
00052         */
00053
00054         /// Returns the set of devices that are part of this context.
00055         virtual vector_class<cl::sy::device>
00056         get_devices() const = 0;
00057
00058         /// Virtual to call the real destructor
00059         virtual ~context() {}
00060     };
00061
00062     /** @} to end the execution Doxygen group
00063
00064     }
00065     }
00066     }
00067
00068     /*
00069     # Some Emacs stuff:
00070     ### Local Variables:
00071     ### ispell-local-dictionary: "american"
00072     ### eval: (flyspell-prog-mode)
00073     ### End:
00074     */
00075
00076 #endif // TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_HPP

```

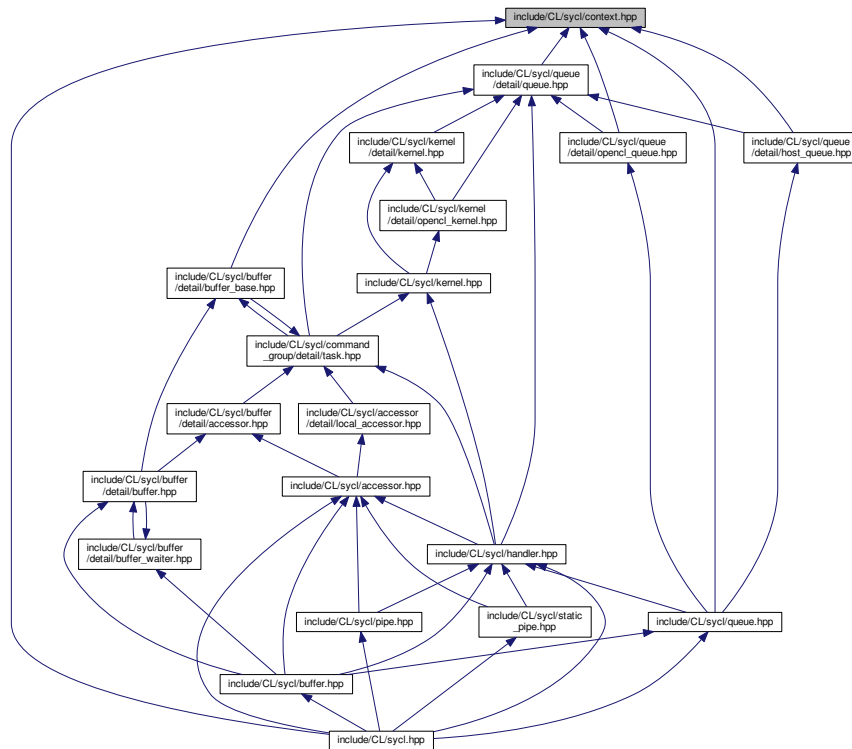
## 11.31 include/CL/sycl/context.hpp File Reference

```
#include <stddef>
#include "CL/sycl/context/detail/host_context.hpp"
#include "CL/sycl/context/detail/opengl_context.hpp"
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.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/context.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for context.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class `cl::sycl::context`  
SYCL context. [More...](#)
- struct `std::hash< cl::sycl::context >`

## Namespaces

- `cl`  
*The vector type to be used as SYCL vector.*
- `cl::sycl`
- `std`

## 11.32 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/context/detail/host_context.hpp"
00015 #ifdef TRISYCL_OPENCL
00016 #include "CL/sycl/context/detail/opencl_context.hpp"
00017 #endif
00018
00019 #include "CL/sycl/detail/default_classes.hpp"
00020 #include "CL/sycl/detail/shared_ptr_implementation.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/info/context.hpp"
00026 #include "CL/sycl/platform.hpp"
00027
00028 namespace cl {
00029 namespace sycl {
00030
00031 /** \addtogroup execution Platforms, contexts, devices and queues
00032     @{
00033 */
00034
00035 /** SYCL context
00036
00037     The context class encapsulates an OpenCL context, which is implicitly
00038     created and the lifetime of the context instance defines the lifetime
00039     of the underlying OpenCL context instance.
00040
00041     On destruction clReleaseContext is called.
00042
00043     The default context is the SYCL host context containing only the SYCL
00044     host device.
00045
00046     \todo The implementation is quite minimal for now.
00047 */
00048 class context
00049 {
00050     /* Use the underlying context implementation that can be shared in the
00051        SYCL model */
00052     : public detail::shared_ptr_implementation<context, detail::context> {
00053
00054     // The type encapsulating the implementation
00055     using implementation_t =
00056         detail::shared_ptr_implementation<context, detail::context>
00057
00058     ;

```

```

00058 public:
00059
00060     // Make the implementation member directly accessible in this class
00061     using implementation_t::implementation;
00062
00063     /** Constructs a context object for SYCL host using an async_handler for
00064         handling asynchronous errors
00065
00066         Note that the default case asyncHandler = nullptr is handled by the
00067         default constructor.
00068     */
00069     explicit context(async_handler asyncHandler) {
00070         detail::unimplemented();
00071     }
00072
00073
00074 #ifndef TRISYCL_OPENCL
00075     /** Make a SYCL context from an OpenCL context
00076
00077         The constructor executes a retain on the \c cl_context.
00078
00079         Return synchronous errors via the SYCL exception class and
00080         asynchronous errors are handled via the \c async_handler, if
00081         provided.
00082     */
00083     context(cl_context clContext, async_handler asyncHandler = nullptr)
00084         : context { boost::compute::context { clContext }, asyncHandler } {}
00085
00086
00087     /// Build a SYCL context from a Boost.Compute context
00088     context(const boost::compute::context &c,
00089             async_handler asyncHandler = nullptr)
00090         : implementation_t { detail::opengl_context::instance(c
00091 ) } {}
00092 #endif
00093
00094     /** Constructs a context object using a device_selector object
00095
00096         The context is constructed with a single device retrieved from the
00097         device_selector object provided.
00098
00099         Return synchronous errors via the SYCL exception class and
00100         asynchronous errors are handled via the async_handler, if provided.
00101     */
00102     context(const device_selector &deviceSelector,
00103             info::gl_context_interop interopFlag,
00104             async_handler asyncHandler = nullptr) {
00105         detail::unimplemented();
00106     }
00107
00108     /** Constructs a context object using a device object
00109
00110         Return synchronous errors via the SYCL exception class and
00111         asynchronous errors are handled via the async_handler, if provided.
00112     */
00113     context(const device &dev,
00114             info::gl_context_interop interopFlag,
00115             async_handler asyncHandler = nullptr) {
00116         detail::unimplemented();
00117     }
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
00132     /** Constructs a context object using a vector_class of device objects
00133
00134         Return synchronous errors via the SYCL exception class and
00135         asynchronous errors are handled via the async_handler, if provided.
00136
00137         \todo Update the specification to replace vector by collection
00138         concept.
00139     */
00140     context(const vector_class<device> &deviceList,
00141             info::gl_context_interop interopFlag,
00142             async_handler asyncHandler = nullptr) {
00143         detail::unimplemented();

```

```

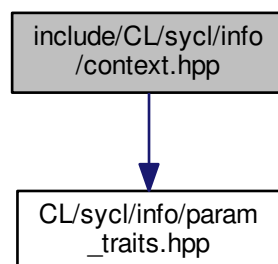
00144     }
00145
00146     /** Default constructor that chooses the context according the
00147         heuristics of the default selector
00148
00149         Return synchronous errors via the SYCL exception class.
00150
00151         Get the default constructors back.
00152     */
00153     context() : implementation_t {
00154         detail::host_context::instance() } {}
00155
00156 #ifndef TRISYCL_OPENCL
00157     /** Return the underlying \c cl_context object, after retaining
00158         the \c cl_context.
00159
00160         Retains a reference to the returned \c cl_context object.
00161
00162         Caller should release it when finished.
00163     */
00164     cl_context get() const {
00165         return implementation->get();
00166     }
00167
00168     /** Return the underlying \c boost::compute::context
00169         of the \c cl::sycl::context
00170     */
00171     boost::compute::context &get_boost_compute() const {
00172         return implementation->get_boost_compute();
00173     }
00174
00175     /** Return the internal queue that is associated to the context and
00176         used by triSYCL to move data between some different contexts for
00177         example
00178     */
00179     boost::compute::command_queue &get_boost_queue() const {
00180         return implementation->get_boost_queue();
00181     }
00182 #endif
00183
00184
00185     /// Specifies whether the context is in SYCL Host Execution Mode.
00186     bool is_host() const {
00187         return implementation->is_host();
00188     }
00189
00190
00191     /** Returns the SYCL platform that the context is initialized for
00192
00193         \todo To be implemented
00194     */
00195     platform get_platform();
00196
00197
00198     /** Returns the set of devices that are part of this context
00199
00200         \todo To be implemented
00201     */
00202     vector_class<device> get_devices() const {
00203         detail::unimplemented();
00204         return {};
00205     }
00206
00207
00208     /** Queries OpenCL information for the under-lying cl context
00209
00210         \todo To be implemented
00211     */
00212     template <info::context Param>
00213     typename info::param_traits<info::context, Param>::type
00214     get_info() const {
00215         detail::unimplemented();
00216         return {};
00217     }
00218 };
00219
00220 /// @} to end the execution Doxygen group
00221
00222 }
00223 }
00224
00225 namespace std {
00226
00227     template <> struct hash<cl::sycl::context> {
00228         auto operator()(const cl::sycl::context &c) const {

```

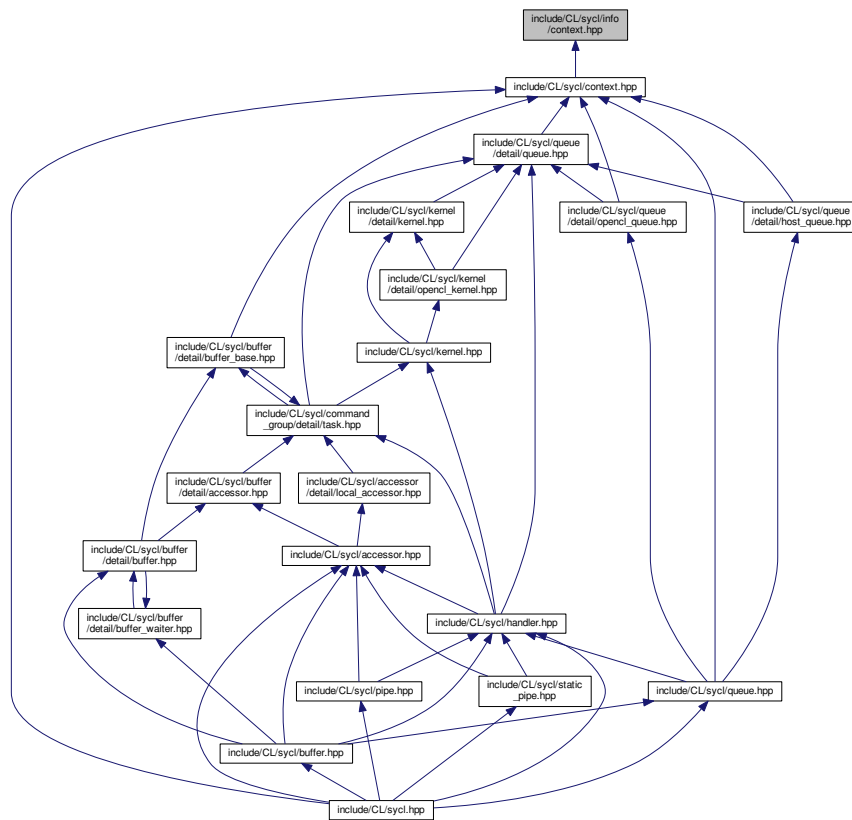
```
00229     return c.hash();
00230   }
00231 };
00232
00233 }
00234
00235 /*
00236   # Some Emacs stuff:
00237   ### Local Variables:
00238   ### ispell-local-dictionary: "american"
00239   ### eval: (flyspell-prog-mode)
00240   ### End:
00241 */
00242
00243 #endif // TRISYCL_SYCL_CONTEXT_HPP
```

### 11.33 include/CL/sycl/info/context.hpp File Reference

#include "CL/sycl/info/param\_traits.hpp"  
Include dependency graph for context.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::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::devices`, `cl::sycl::info::context::gl_interop` }  
Context information descriptors.

## 11.34 context.hpp

```

00001 #ifndef TRISYCL_SYCL_INFO_CONTEXT_HPP
00002 #define TRISYCL_SYCL_INFO_CONTEXT_HPP
00003
00004 /** \file The OpenCL SYCL context information parameters
00005
00006     Anastasi at Xilinx dot com
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 namespace info {
00021
00022 using gl_context_interop = bool;
00023
00024 /** Context information descriptors
00025
00026     \todo Should be unsigned int to be consistent with others?
00027 */
00028 enum class context : int {
00029     reference_count,
00030     num_devices,
00031     devices,
00032     gl_interop
00033 };
00034
00035
00036 /** Query the return type for get_info() on context stuff
00037
00038     \todo To be implemented, return always void.
00039 */
00040 TRISYCL_INFO_PARAM_TRAITS_ANY_T(info::context, void)
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
00055 #endif //TRISYCL_SYCL_INFO_CONTEXT_HPP

```

## 11.35 include/CL/sycl/context/detail/host\_context.hpp File Reference

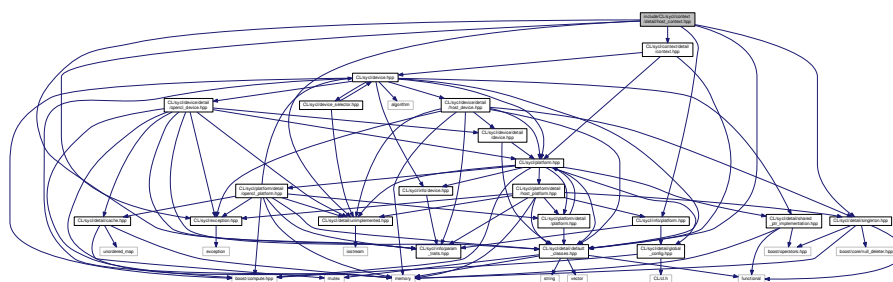
```

#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/detail singleton.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/info/platform.hpp"
#include "CL/sycl/context/detail/context.hpp"

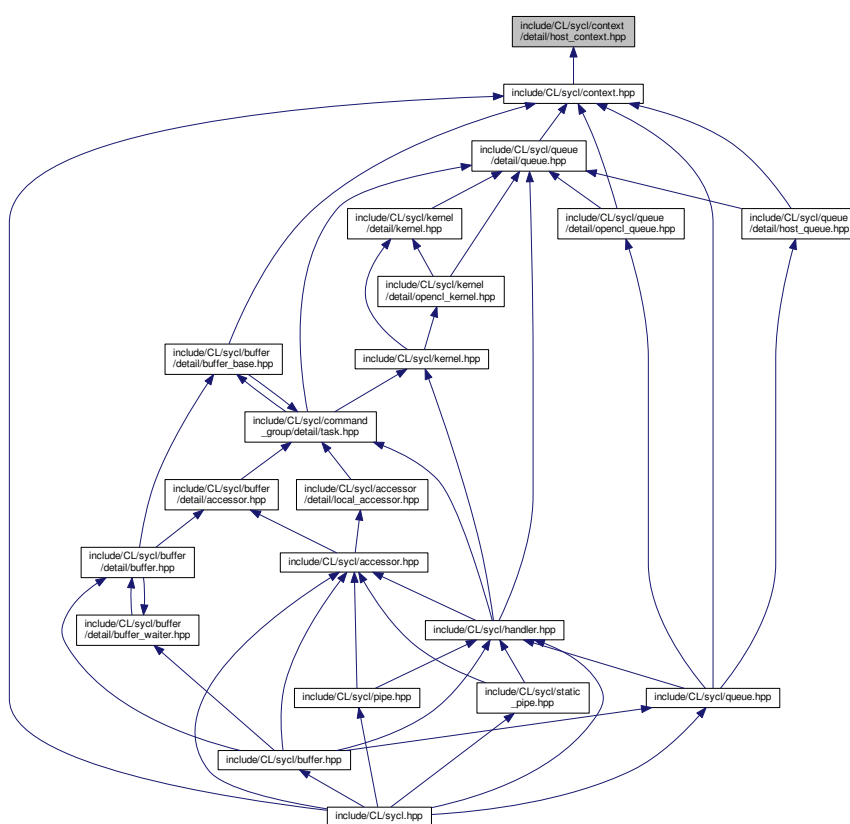
```



Include dependency graph for host\_context.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [cl::sycl::detail::host\\_context](#)

## Namespaces

- [cl](#)  
The vector type to be used as SYCL vector.
- [cl::sycl](#)
- [cl::sycl::detail](#)

## 11.36 host\_context.hpp

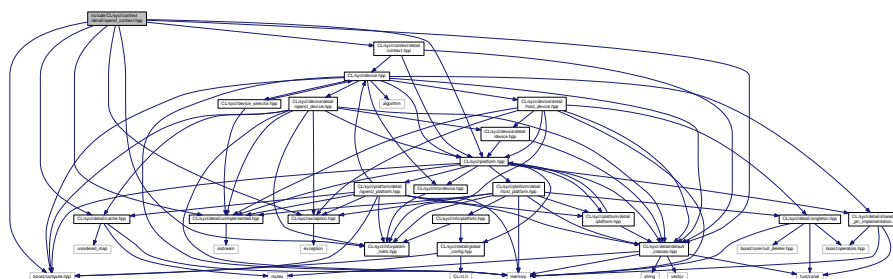
```

00001 #ifndef TRISYCL_SYCL_CONTEXT_DETAIL_HOST_CONTEXT_HPP
00002 #define TRISYCL_SYCL_CONTEXT_DETAIL_HOST_CONTEXT_HPP
00003
00004 /** \file The OpenCL SYCL host context implementation
00005
00006     a-doumoulakis at gmail dot com (Anastasios Doumoulakis)
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 #include "CL/sycl/detail/unimplemented.hpp"
00014 #include "CL/sycl/detail/singleton.hpp"
00015 #include "CL/sycl/exception.hpp"
00016 #include "CL/sycl/info/param_traits.hpp"
00017 #include "CL/sycl/info/platform.hpp"
00018 #include "CL/sycl/context/detail/context.hpp"
00019
00020 namespace cl {
00021 namespace sycl {
00022 namespace detail {
00023
00024
00025 /** \addtogroup execution Platforms, contexts, devices and queues
00026     @{
00027
00028     SYCL host context
00029
00030     \todo The implementation is quite minimal for now. :-)
00031 */
00032 class host_context : public detail::context,
00033                     public detail::singleton<host_context> {
00034
00035 public:
00036
00037 #ifndef TRISYCL_OPENCL
00038     /** Return the underlying \c cl_context of the \c cl::sycl::context
00039
00040         This throws an error since there is no OpenCL context associated
00041         to the host device.
00042     */
00043     cl_context get() const override {
00044         throw non_cl_error("The host context has no OpenCL context");
00045     }
00046
00047
00048     /** Return the SYCL platform that the context is initialized for
00049
00050         This throws an error since there is no \c boost::compute context
00051         associated to the host device.
00052     */
00053     boost::compute::context &get_boost_compute() override {
00054         throw non_cl_error("The host context has no OpenCL context");
00055     }
00056
00057
00058     /** Return the internal OpenCL queue that is associated to the host
00059         context
00060
00061         This throws an error since there is no \c
00062         boost::compute::command_queue associated to the host context.
00063     */
00064     boost::compute::command_queue &get_boost_queue() override {
00065         throw non_cl_error("The host context cannot have an OpenCL queue");
00066     }
00067 #endif
00068
00069
00070     /// Return true since the context is a SYCL host context
00071     bool is_host() const override {
00072         return true;
00073     }
00074
00075
00076     /** Return the platform of the context
00077
00078         Return synchronous errors via the SYCL exception class.
00079     */
00080     cl::sycl::platform get_platform() const override {
00081         // Return the host platform
00082         return {};
00083     }
00084

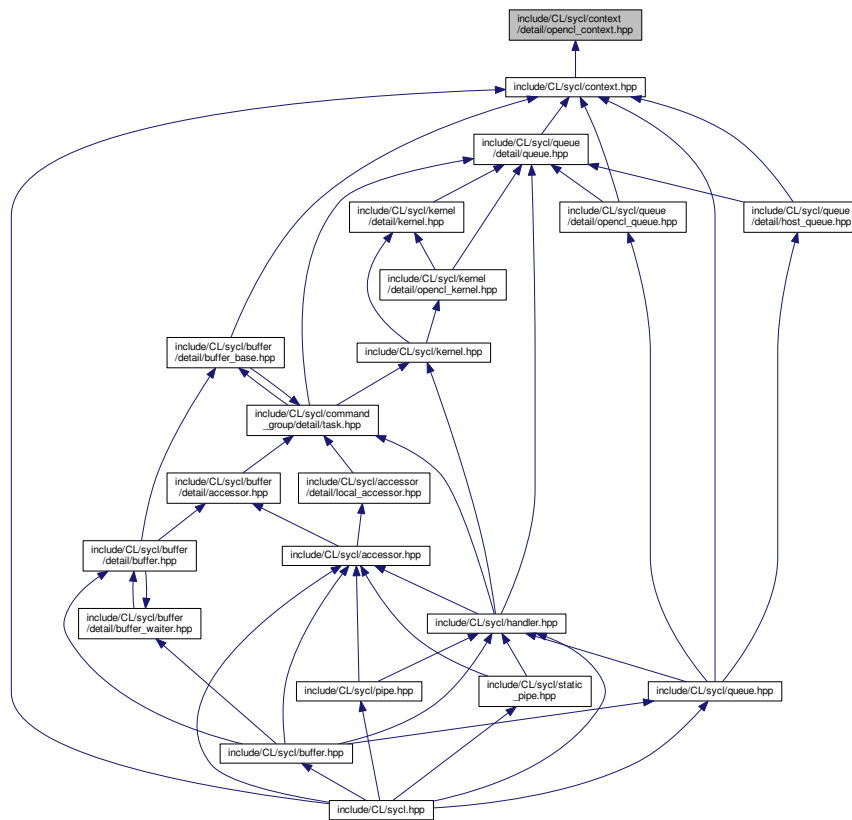
```

## 11.37 include/CL/sycl/context/detail/opencl\_context.hpp File Reference

Include dependency graph for `opengl_context.hpp`:



This graph shows which files directly or indirectly include this file:



## Classes

- class [cl::sycl::detail::opengl\\_context](#)  
*SYCL OpenCL context.*

## Namespaces

- [cl](#)  
*The vector type to be used as SYCL vector.*
- [cl::sycl](#)
- [cl::sycl::detail](#)

## Variables

- [TRISYCL\\_WEAK\\_ATTRIB\\_PREFIX](#) [detail::cache< cl\\_context, detail::opengl\\_context > opengl\\_context](#)↔  
[::cache](#) [cl::sycl::detail::TRISYCL\\_WEAK\\_ATTRIB\\_SUFFIX](#)

## 11.38 opengl\_context.hpp

```

00001 #ifndef TRISYCL_SYCL_CONTEXT_DETAIL_OPENGL_CONTEXT_HPP
00002 #define TRISYCL_SYCL_CONTEXT_DETAIL_OPENGL_CONTEXT_HPP
00003
00004 /** \file The SYCL OpenCL context implementation
00005
00006     a-doumoulakis at gmail dot com (Anastasios Doumoulakis)
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012
00013 #include <boost/compute.hpp>
00014
00015 #include "CL/sycl/detail/default_classes.hpp"
00016 #include "CL/sycl/detail/unimplemented.hpp"
00017 #include "CL/sycl/detail/cache.hpp"
00018
00019 #include "CL/sycl/context/detail/context.hpp"
00020 #include "CL/sycl/platform.hpp"
00021 #include "CL/sycl/info/param_traits.hpp"
00022 #include "CL/sycl/exception.hpp"
00023
00024
00025 namespace cl {
00026 namespace sycl {
00027 namespace detail {
00028
00029 /// SYCL OpenCL context
00030 class opengl_context : public detail::context {
00031
00032     /// User the Boost Compute abstraction of the OpenCL context
00033     boost::compute::context c;
00034
00035     /** A boost \c command_queue associated to an OpenCL context for
00036         when we need to transfer data but no queue is given
00037         (eg. When a buffer accessor is created)
00038     */
00039     boost::compute::command_queue q;
00040
00041     /** A cache to always return the same alive context for a given OpenCL
00042         context
00043
00044         C++11 guaranties the static construction is thread-safe
00045     */
00046     static detail::cache<cl_context, detail::opengl_context>
00047     cache;
00048 public:
00049
00050     /// Return the underlying \c cl_context of the \c cl::synd::context
00051     cl_context get() const override {
00052         return c.get();
00053     }
00054
00055
00056     /** Return the underlying \c boost::compute::context
00057         of the \c cl::synd::context
00058     */
00059     boost::compute::context &get_boost_compute() override {
00060         return c;
00061     }
00062
00063
00064     /// Return the queue that is associated to the context
00065     boost::compute::command_queue &get_boost_queue() override {
00066         return q;
00067     }
00068
00069
00070     /// Return false because the context is not a SYCL host context
00071     bool is_host() const override {
00072         return false;
00073     }
00074
00075 #if 0
00076     /** Query the context for OpenCL \c info::context info
00077
00078         Return synchronous errors via the SYCL exception class.
00079
00080         \todo To be implemented
00081     */
00082     template <info::context Param>
00083     typename info::param_traits<info::context, Param>::type

```

```

00084     get_info() const override{
00085         detail::unimplemented();
00086         return {};
00087     }
00088 #endif
00089
00090     /** Return the platform of the context
00091
00092         Return synchronous errors via the SYCL exception class.
00093
00094         \todo To be implemented
00095     */
00096     cl::sycl::platform get_platform() const override {
00097         detail::unimplemented();
00098         return {};
00099     }
00100
00101
00102     /** Returns the set of devices that are part of this context.
00103
00104         \todo To be implemented
00105     */
00106     vector_class<cl::sycl::device> get_devices() const override {
00107         detail::unimplemented();
00108         return {};
00109     }
00110
00111     /// Get a singleton instance of the \c opencil_context
00112     static std::shared_ptr<opencil_context>
00113     instance(const boost::compute::context &c) {
00114         return cache.get_or_register(c.get(),
00115                                     [&] { return new opencil_context { c }; });
00116     }
00117
00118 private:
00119
00120     /// Only the instance factory can build it
00121     opencil_context(const boost::compute::context &c) :
00122         c { c },
00123         q { boost::compute::command_queue { c, c.get_device() } } {}
00124
00125 public:
00126
00127     /// Unregister from the cache on destruction
00128     ~opencil_context() override {
00129         cache.remove(c.get());
00130     }
00131
00132 };
00133
00134
00135
00136 /* Allocate the cache here but since this is a pure-header library,
00137    use a weak symbol so that only one remains when SYCL headers are
00138    used in different compilation units of a program
00139 */
00140 TRISYCL_WEAK_ATTRIB_PREFIX
00141 detail::cache<cl_context, detail::opencil_context>
00142 opencil_context::cache
00143 TRISYCL_WEAK_ATTRIB_SUFFIX;
00144 }
00145 }
00146 }
00147
00148 /*
00149     # Some Emacs stuff:
00150     ### Local Variables:
00151     ###  ispell-local-dictionary: "american"
00152     ###  eval: (flyspell-prog-mode)
00153     ###  End:
00154 */
00155
00156 #endif //TRISYCL_SYCL_CONTEXT_DETAIL_OPENCL_CONTEXT_HPP

```

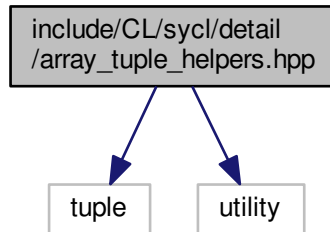
## 11.39 include/CL/sycl/detail/array\_tuple\_helpers.hpp File Reference

Some helpers to do array-tuple conversions.

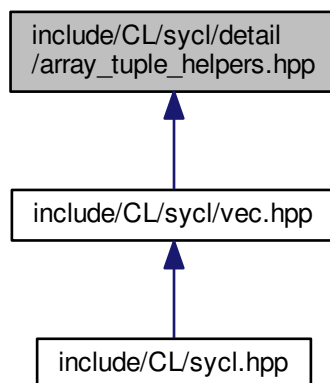
```
#include <tuple>
```

```
#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.*

### 11.39.1 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](#).

### 11.40 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

```

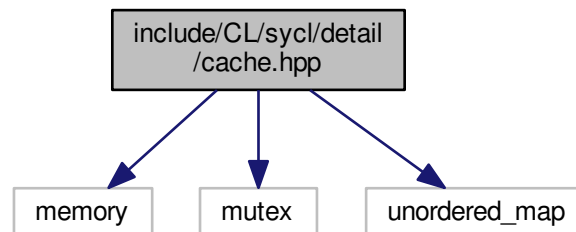
## 11.41 include/CL/sycl/detail/cache.hpp File Reference

```

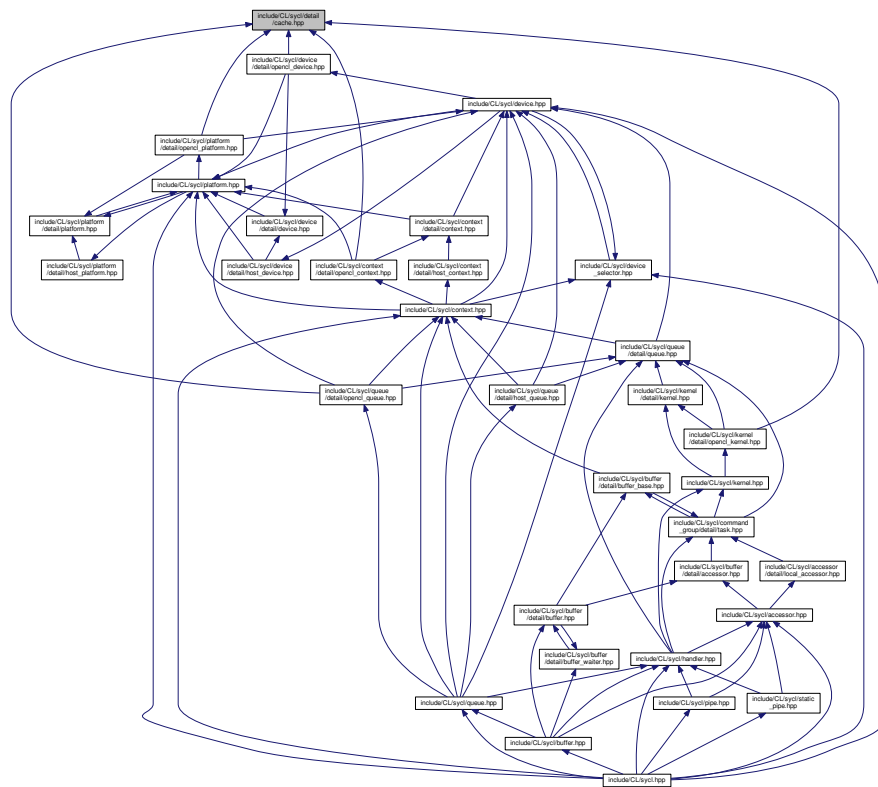
#include <memory>
#include <mutex>
#include <unordered_map>

```

Include dependency graph for cache.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class `cl::sycl::detail::cache< Key, Value >`

*A simple thread safe cache mechanism to cache `std::shared_ptr` of values indexed by keys.*

## Namespaces

- `cl`  
*The vector type to be used as SYCL vector.*
- `cl::sycl`
- `cl::sycl::detail`

## 11.42 cache.hpp

```

00001 #ifndef TRISYCL_SYCL_DETAIL_CACHE_HPP
00002 #define TRISYCL_SYCL_DETAIL_CACHE_HPP
00003
00004 /** \file A simple thread-safe cache
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 #include <mutex>

```

```

00014 #include <unordered_map>
00015
00016 namespace cl {
00017 namespace sycl {
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 */

```

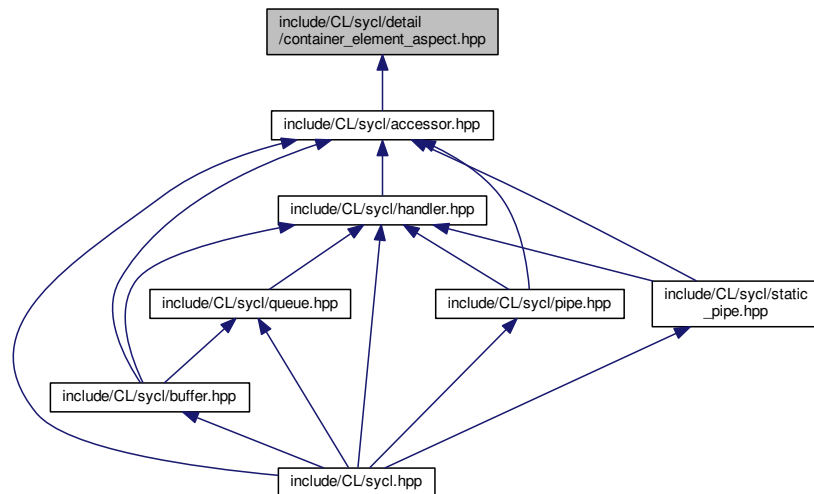
```

00101
00102 #endif // TRISYCL_SYCL_DEVICE_CACHE_HPP

```

## 11.43 include/CL/sycl/detail/container\_element\_aspect.hpp File Reference

This graph shows which files directly or indirectly include this file:



### Classes

- `struct cl::sycl::detail::container_element_aspect< T >`  
A mix-in to add some container element aspects. [More...](#)

### Namespaces

- `cl`  
The vector type to be used as SYCL vector.
- `cl::sycl`
- `cl::sycl::detail`

## 11.44 container\_element\_aspect.hpp

```

00001 #ifndef TRISYCL_SYCL_DETAIL_CONTAINER_ELEMENT_ASPECT_HPP
00002 #define TRISYCL_SYCL_DETAIL_CONTAINER_ELEMENT_ASPECT_HPP
00003
00004 /** \file Implement basic types à la STL related to container
00005     elements, such as value_type, reference...
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 namespace cl {
00014 namespace sycl {
00015 namespace detail {
00016
00017 /** \addtogroup helpers Some helpers for the implementation
00018     @{
00019 */
00020
00021 /// A mix-in to add some container element aspects
00022 template <typename T>
00023 struct container_element_aspect {
00024
00025     using value_type = T;
00026     using pointer = value_type*;
00027     using const_pointer = const value_type*;
00028     using reference = value_type&;
00029     using const_reference = const value_type&;
00030
00031 };
00032
00033 /// @} End the helpers Doxygen group
00034
00035 }
00036 }
00037 }
00038
00039 /*
00040     # Some Emacs stuff:
00041     ### Local Variables:
00042     ###   ispell-local-dictionary: "american"
00043     ###   eval: (flyspell-prog-mode)
00044     ### End:
00045 */
00046
00047 #endif // TRISYCL_SYCL_DETAIL_CONTAINER_ELEMENT_ASPECT_HPP

```

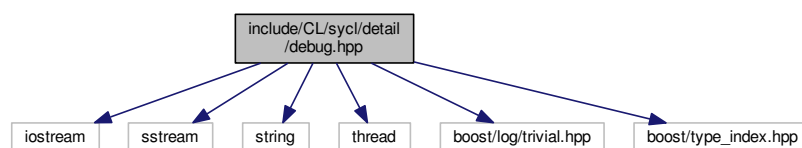
## 11.45 include/CL/sycl/detail/debug.hpp File Reference

```

#include <iostream>
#include <sstream>
#include <string>
#include <thread>
#include <boost/log/trivial.hpp>
#include <boost/type_index.hpp>

```

Include dependency graph for debug.hpp:



[illegible]

- 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...](#)*

- `cl`  
*The vector type to be used as SYCL vector.*
- `cl::sycl`
- `cl::sycl::detail`

- #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.*

- `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.*

## 11.45.1 Macro Definition Documentation

### 11.45.1.1 TRISYCL\_DUMP

```
#define TRISYCL_DUMP (
    expression ) TRISYCL_INTERNAL_DUMP(expression)
```

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

### 11.45.1.2 TRISYCL\_DUMP\_T

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

### 11.45.1.3 TRISYCL\_INTERNAL\_DUMP

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



## 11.46 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 ostringstream 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     }

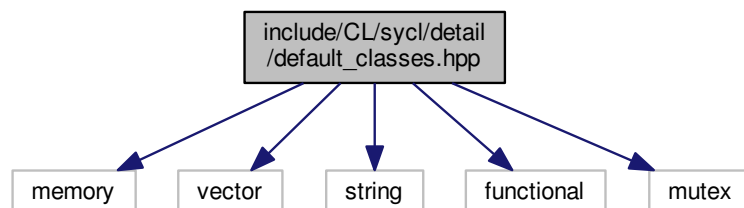
```

```
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
```

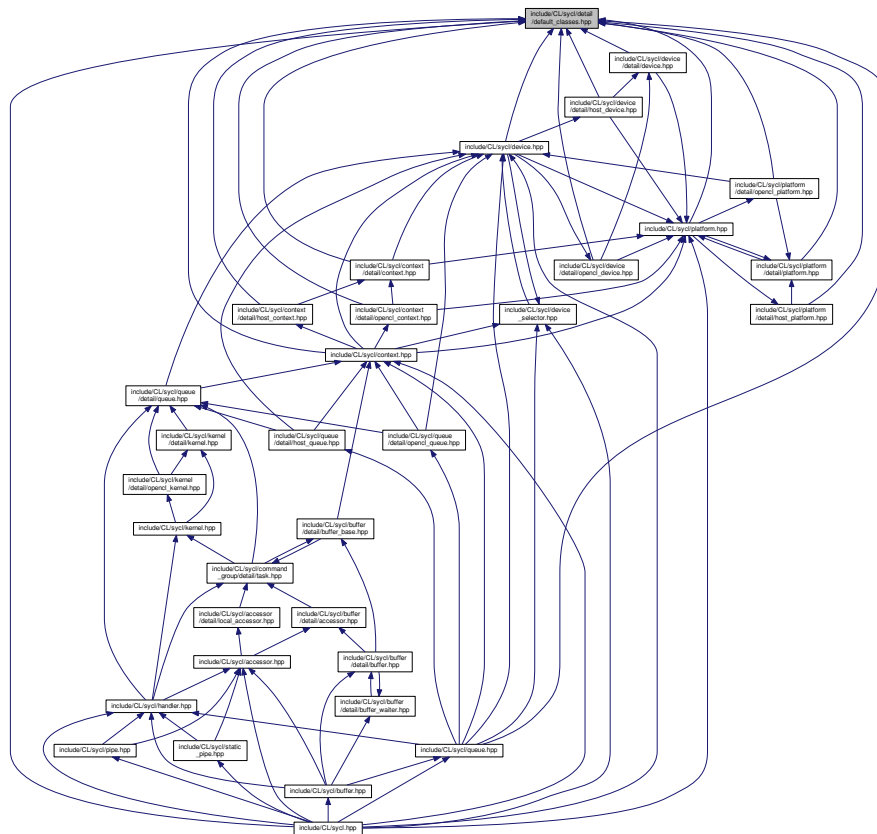
## 11.47 include/CL/sycl/detail/default\_classes.hpp File Reference

```
#include <memory>
#include <vector>
#include <string>
#include <functional>
#include <mutex>
```

Include dependency graph for default\_classes.hpp:



This graph shows which files directly or indirectly include this file:



## Namespaces

- `cl`

*The vector type to be used as SYCL vector.*

- `cl::sycl`

## Typedefs

- `template<class T, class Alloc = std::allocator<T>>`  
using `cl::sycl::vector_class` = `std::vector< T, Alloc >`
- using `cl::sycl::string_class` = `std::string`
- `template<class R, class... ArgTypes>`  
using `cl::sycl::function_class` = `std::function< R(ArgTypes...)>`
- using `cl::sycl::mutex_class` = `std::mutex`
- `template<class T, class D = std::default_delete<T>>`  
using `cl::sycl::unique_ptr_class` = `std::unique_ptr< T[], D >`
- `template<class T >`  
using `cl::sycl::shared_ptr_class` = `std::shared_ptr< T >`
- `template<class T >`  
using `cl::sycl::weak_ptr_class` = `std::weak_ptr< T >`

## 11.48 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

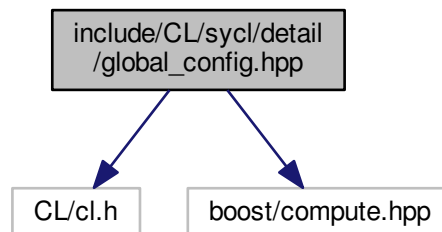
```

## 11.49 include/CL/sycl/detail/global\_config.hpp File Reference

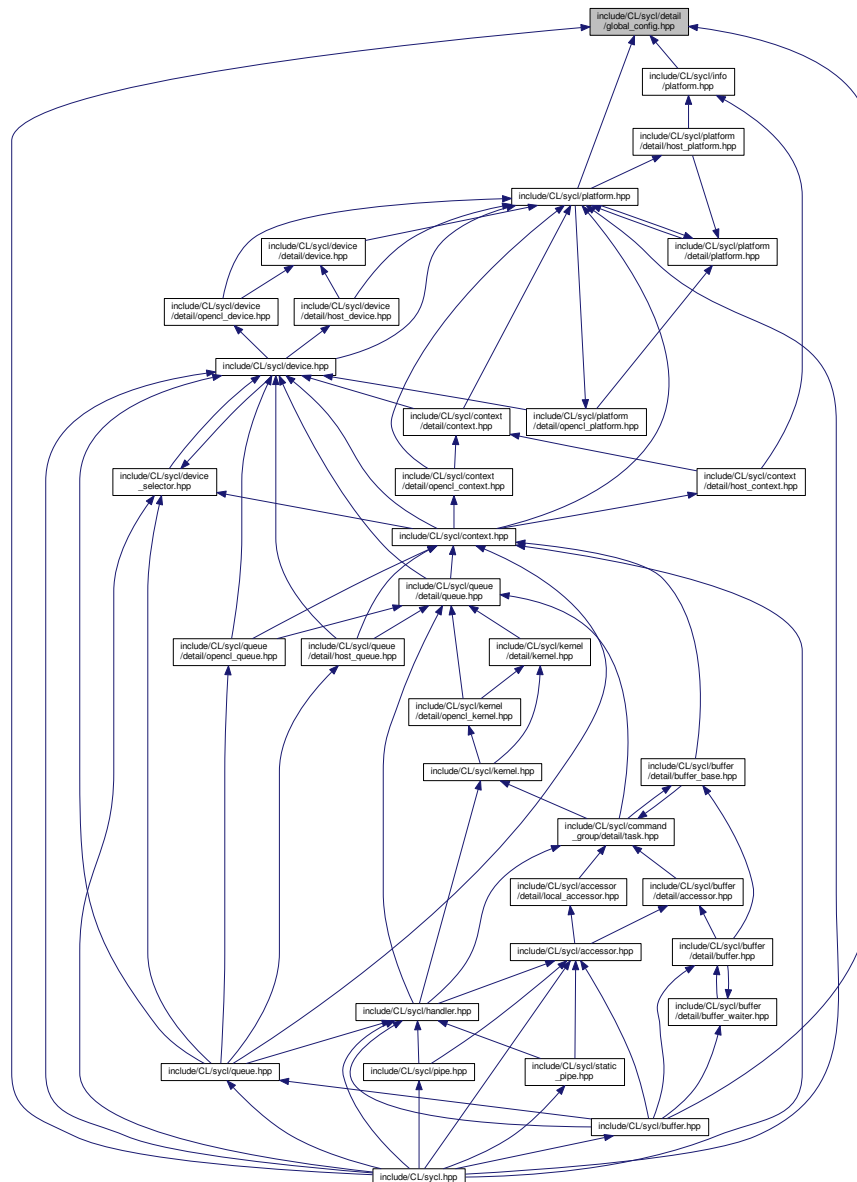
```
#include <CL/cl.h>
```

```
#include <boost/compute.hpp>
```

Include dependency graph for global\_config.hpp:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define CL_SYCL_LANGUAGE_VERSION 220`  
*This implement SYCL 2.2.*
- `#define TRISYCL_CL_LANGUAGE_VERSION 220`  
*This implement triSYCL 2.2.*
- `#define __SYCL_SINGLE_SOURCE__`  
*This source is compiled by a single source compiler.*
- `#define TRISYCL_MAKE_BOOST_CIRCULARBUFFER_THREAD_SAFE`
- `#define TRISYCL_SKIP_OPENCL(x) x`  
*Define TRISYCL\_OPENCL to add OpenCL.*
- `#define TRISYCL_WEAK_ATTRIB_PREFIX`
- `#define TRISYCL_WEAK_ATTRIB_SUFFIX __attribute__((weak))`

## 11.49.1 Macro Definition Documentation

### 11.49.1.1 TRISYCL\_WEAK\_ATTRIB\_PREFIX

```
#define TRISYCL_WEAK_ATTRIB_PREFIX
```

Definition at line 65 of file [global\\_config.hpp](#).

Referenced by [cl::sycl::detail::opencl\\_kernel::TRISYCL\\_ParallelForKernel\\_RANGE\(\)](#), [cl::sycl::detail::opencl\\_↵  
context::~~opencl\\_context\(\)](#), [cl::sycl::detail::opencl\\_device::~~opencl\\_device\(\)](#), [cl::sycl::detail::opencl\\_platform\\_↵  
::~~opencl\\_platform\(\)](#), and [cl::sycl::detail::opencl\\_queue::~~opencl\\_queue\(\)](#).

### 11.49.1.2 TRISYCL\_WEAK\_ATTRIB\_SUFFIX

```
#define TRISYCL_WEAK_ATTRIB_SUFFIX __attribute__((weak))
```

Definition at line 66 of file [global\\_config.hpp](#).

Referenced by [cl::sycl::device::get\\_platform\(\)](#).

## 11.50 global\_config.hpp

```
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 TRISYCL_CL_LANGUAGE_VERSION
00023 /// This implement triSYCL 2.2
00024 #define TRISYCL_CL_LANGUAGE_VERSION 220
00025 #endif
00026
00027 /// This source is compiled by a single source compiler
00028 #define __SYCL_SINGLE_SOURCE__
00029
00030
00031 /* Work-around an old Boost.CircularBuffer bug if a pre 1.62 Boost
00032    version is used */
00033 #define TRISYCL_MAKE_BOOST_CIRCULARBUFFER_THREAD_SAFE
00034
00035
00036 /** Define TRISYCL_OPENCL to add OpenCL
00037
00038     triSYCL can indeed work without OpenCL if only host support is needed.
00039 */
```



```

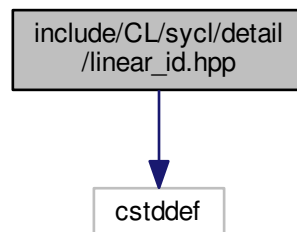
00040 #ifdef TRISYCL_OPENCL
00041
00042 // SYCL interoperation API with OpenCL requires some OpenCL C types:
00043 #if defined(__APPLE__)
00044 #include <OpenCL/cl.h>
00045 #else
00046 #include <CL/cl.h>
00047 #endif
00048 // But the triSYCL OpenCL implementation is actually based on Boost.Compute
00049 #include <boost/compute.hpp>
00050 /// A macro to keep some stuff in OpenCL mode
00051 #define TRISYCL_SKIP_OPENCL(x) x
00052 #else
00053 /// A macro to skip stuff when not supporting OpenCL
00054 #define TRISYCL_SKIP_OPENCL(x)
00055 #endif
00056
00057 /// @} End the defaults Doxygen group
00058
00059 // Compiler specific weak linking (until changing to C++17 inline variables/functions)
00060 #ifndef TRISYCL_WEAK_ATTRIB_PREFIX
00061 #ifdef _MSC_VER
00062 #define TRISYCL_WEAK_ATTRIB_PREFIX __declspec(selectany)
00063 #define TRISYCL_WEAK_ATTRIB_SUFFIX
00064 #else
00065 #define TRISYCL_WEAK_ATTRIB_PREFIX
00066 #define TRISYCL_WEAK_ATTRIB_SUFFIX __attribute__((weak))
00067 #endif
00068 #endif
00069
00070 // Suppress usage/leak of macros originating from Visual C++ headers
00071 #ifdef _MSC_VER
00072 #define NOMINMAX
00073 #endif
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_DETAIL_GLOBAL_CONFIG_HPP

```

## 11.51 include/CL/sycl/detail/linear\_id.hpp File Reference

#include <cstdint>

Include dependency graph for linear\_id.hpp:





```

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

```

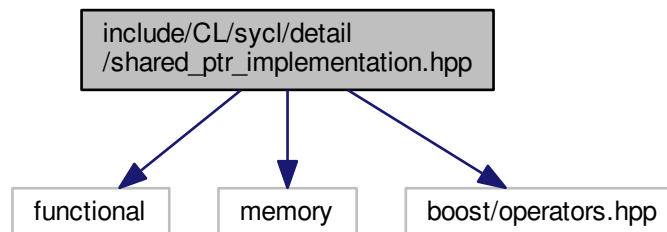
## 11.53 include/CL/sycl/detail/shared\_ptr\_implementation.hpp File Reference

```

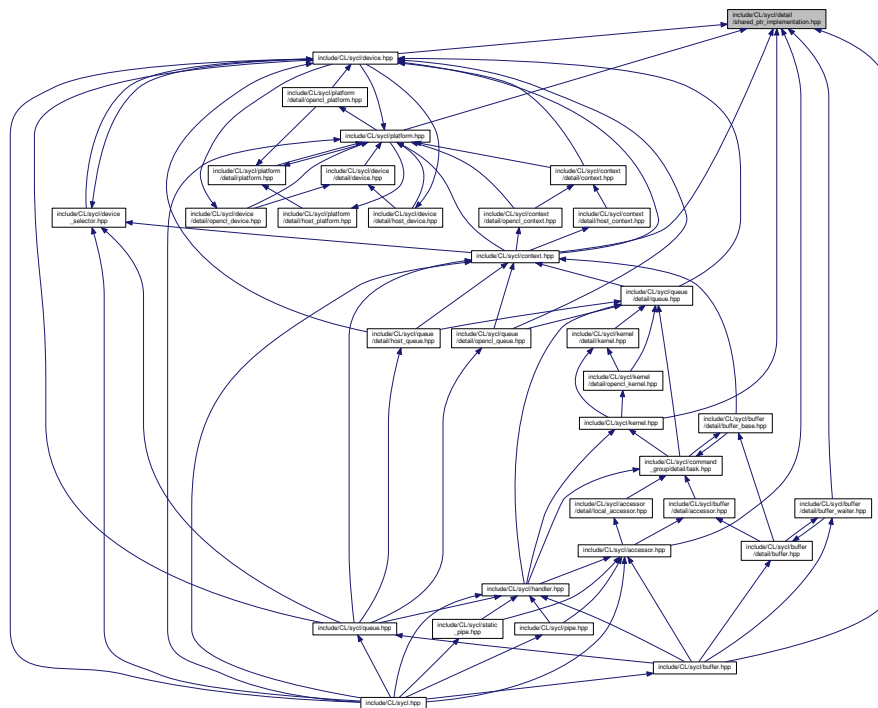
#include <functional>
#include <memory>
#include <boost/operators.hpp>

```

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`

## 11.54 `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>

```

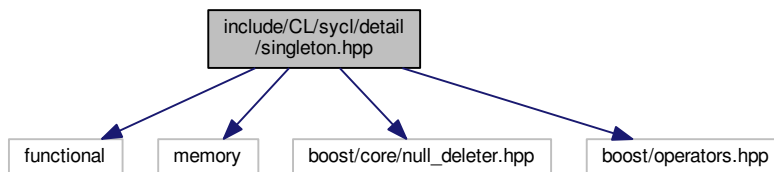
```

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     /// The implementation directly as a shared pointer
00046     shared_ptr_implementation(std::shared_ptr<Implementation> i)
00047         : implementation { i } {}
00048
00049
00050     /// The implementation takes the ownership from a raw pointer
00051     shared_ptr_implementation(Implementation *i) : implementation { i } {}
00052
00053
00054     /// Keep all other constructors to have usual shared_ptr behaviour
00055     shared_ptr_implementation() = default;
00056
00057
00058     /** Equality operator
00059
00060         This is generalized by boost::equality_comparable from
00061         boost::totally_ordered to implement the equality comparable
00062         concept
00063     */
00064     bool operator==(const Parent &other) const {
00065         return implementation == other.implementation;
00066     }
00067
00068
00069     /** Inferior operator
00070
00071         This is generalized by boost::less_than_comparable from
00072         boost::totally_ordered to implement the equality comparable
00073         concept
00074
00075         \todo Add this to the spec
00076     */
00077     bool operator<(const Parent &other) const {
00078         return implementation < other.implementation;
00079     }
00080
00081
00082     /// Forward the hashing for unordered containers to the implementation
00083     auto hash() const {
00084         return std::hash<decltype(implementation)>{}(implementation);
00085     }
00086
00087 };
00088
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_DETAIL_SHARED_PTR_IMPLEMENTATION_HPP

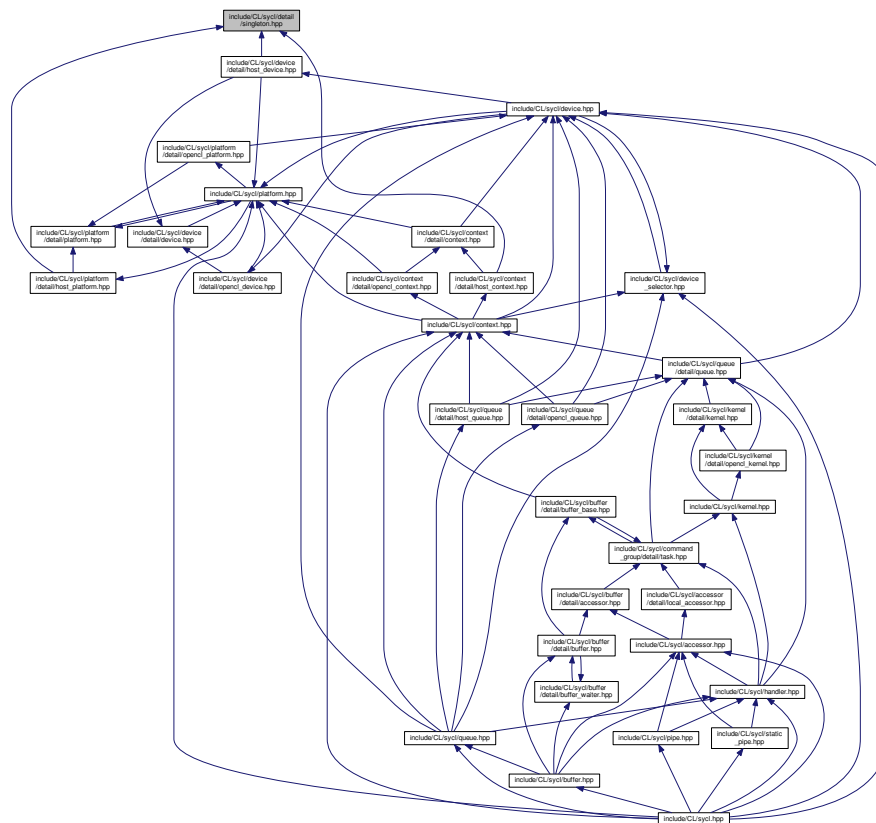
```

## 11.55 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`

## 11.56 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 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         }
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

```

## 11.57 include/CL/sycl/detail/small\_array.hpp File Reference

```

#include <algorithm>
#include <array>
#include <cstdint>
#include <type_traits>
#include <boost/operators.hpp>

```





## Macros

- `#define TRISYCL_BOOST_OPERATOR_VECTOR_OP(op)`  
*Helper macro to declare a vector operation with the given side-effect operator.*
- `#define TRISYCL_LOGICAL_OPERATOR_VECTOR_OP(op)`

## 11.58 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 TRISYCL_LOGICAL_OPERATOR_VECTOR_OP(op) \
00042     FinalType operator op(const FinalType &rhs) { \
00043         FinalType res; \
00044         for (std::size_t i = 0; i != Dims; ++i) \
00045             res[i] = (*this)[i] op rhs[i]; \
00046         return res; \
00047     }
00048
00049
00050 /** Define a multi-dimensional index, used for example to locate a work
00051     item or a buffer element
00052
00053     Unfortunately, even if std::array is an aggregate class allowing
00054     native list initialization, it is no longer an aggregate if we derive
00055     from an aggregate. Thus we have to redeclare the constructors.
00056
00057     \param BasicType is the type element, such as int
00058
00059     \param Dims is the dimension number, typically between 1 and 3
00060
00061     \param FinalType is the final type, such as range<> or id<>, so that
00062     boost::operator can return the right type
00063
00064     \param EnableArgsConstructor adds a constructors from Dims variadic
00065     elements when true. It is false by default.
00066
00067     std::array<> provides the collection concept, with .size(), == and !=
00068     too.
00069 */
00070 template <typename BasicType,
00071           typename FinalType,

```

```

00072         std::size_t Dims,
00073         bool EnableArgsConstructor = false>
00074 struct small_array : std::array<BasicType, Dims>,
00075 // To have all the usual arithmetic operations on this type
00076 boost::euclidean_ring_operators<FinalType>,
00077 // Bitwise operations
00078 boost::bitwise<FinalType>,
00079 // Shift operations
00080 boost::shiftable<FinalType>,
00081 // Already provided by array<> lexicographically:
00082 // boost::equality_comparable<FinalType>,
00083 // boost::less_than_comparable<FinalType>,
00084 // Add a display() method
00085 detail::display_vector<FinalType> {
00086
00087 /// \todo add this Boost::multi_array or STL concept to the
00088 /// specification?
00089 static const auto dimensionality = Dims;
00090
00091 /* Note that constexpr size() from the underlying std::array provides
00092 the same functionality */
00093 static const size_t dimension = Dims;
00094 using element_type = BasicType;
00095
00096
00097 /** A constructor from another array
00098
00099 Make it explicit to avoid spurious range<> constructions from int *
00100 for example
00101 */
00102 template <typename SourceType>
00103 small_array(const SourceType src[Dims]) {
00104     // (*this)[0] is the first element of the underlying array
00105     std::copy_n(src, Dims, &(*this)[0]);
00106 }
00107
00108
00109 /** An accessor to the first variable of a small array
00110 */
00111 BasicType& x(){
00112     static_assert(Dims >= 1, "can't access to small_array[0] if Dims < 1");
00113     return (*this)[0];
00114 }
00115
00116
00117 /** An accessor to the second variable of a small array
00118 */
00119 BasicType& y(){
00120     static_assert(Dims >= 2, "can't access to small_array[1] if Dims < 2");
00121     return (*this)[1];
00122 }
00123
00124
00125 /** An accessor to the third variable of a small array
00126 */
00127 BasicType& z(){
00128     static_assert(Dims >= 3, "can't access to small_array[2] if Dims < 3");
00129     return (*this)[2];
00130 }
00131
00132
00133 /// A constructor from another small_array of the same size
00134 template <typename SourceBasicType,
00135         typename SourceFinalType,
00136         bool SourceEnableArgsConstructor>
00137 small_array(const small_array<SourceBasicType,
00138         SourceFinalType,
00139         Dims,
00140         SourceEnableArgsConstructor> &src) {
00141     std::copy_n(&src[0], Dims, &(*this)[0]);
00142 }
00143
00144
00145 /** Initialize the array from a list of elements
00146
00147 Strangely, even when using the array constructors, the
00148 initialization of the aggregate is not available. So recreate an
00149 equivalent here.
00150
00151 Since there are inherited types that defines some constructors with
00152 some conflicts, make it optional here, according to
00153 EnableArgsConstructor template parameter.
00154 */
00155 template <typename... Types,
00156         // Just to make enable_if depend of the template and work
00157         bool Depend = true,
00158         typename = typename std::enable_if_t<EnableArgsConstructor

```

```

00159                                     && Depend>>
00160 small_array(const Types &... args)
00161 : std::array<BasicType, Dims> {
00162     // Allow a loss of precision in initialization with the static_cast
00163     { static_cast<BasicType>(args)... }
00164 }
00165 {
00166     static_assert(sizeof...(args) == Dims,
00167         "The number of initializing elements should match "
00168         "the dimension");
00169 }
00170
00171
00172 /// Construct a small_array from a std::array
00173 template <typename SourceBasicType>
00174 small_array(const std::array<SourceBasicType, Dims> &src)
00175 : std::array<BasicType, Dims>(src) {}
00176
00177
00178 /// Keep other constructors from the underlying std::array
00179 using std::array<BasicType, Dims>::array;
00180
00181 /// Keep the synthesized constructors
00182 small_array() = default;
00183
00184 /// Return the element of the array
00185 auto get(std::size_t index) const {
00186     return (*this)[index];
00187 }
00188
00189 /* Implement minimal methods boost::euclidean_ring_operators needs to
00190    generate everything */
00191 /// Add + like operations on the id<> and others
00192 TRISYCL_BOOST_OPERATOR_VECTOR_OP(+=)
00193
00194 /// Add - like operations on the id<> and others
00195 TRISYCL_BOOST_OPERATOR_VECTOR_OP(-=)
00196
00197 /// Add * like operations on the id<> and others
00198 TRISYCL_BOOST_OPERATOR_VECTOR_OP(*=)
00199
00200 /// Add / like operations on the id<> and others
00201 TRISYCL_BOOST_OPERATOR_VECTOR_OP(/=)
00202
00203 /// Add % like operations on the id<> and others
00204 TRISYCL_BOOST_OPERATOR_VECTOR_OP(%=)
00205
00206 /// Add << like operations on the id<> and others
00207 TRISYCL_BOOST_OPERATOR_VECTOR_OP(<<=)
00208
00209 /// Add >> like operations on the id<> and others
00210 TRISYCL_BOOST_OPERATOR_VECTOR_OP(>>=)
00211
00212 /// Add & like operations on the id<> and others
00213 TRISYCL_BOOST_OPERATOR_VECTOR_OP(&=)
00214
00215 /// Add ^ like operations on the id<> and others
00216 TRISYCL_BOOST_OPERATOR_VECTOR_OP(^=)
00217
00218 /// Add | like operations on the id<> and others
00219 TRISYCL_BOOST_OPERATOR_VECTOR_OP(|=)
00220
00221 TRISYCL_LOGICAL_OPERATOR_VECTOR_OP(&&)
00222
00223 TRISYCL_LOGICAL_OPERATOR_VECTOR_OP(||)
00224
00225
00226 /** Since the boost::operator work on the small_array, add an implicit
00227     conversion to produce the expected type */
00228 operator FinalType () {
00229     return *static_cast<FinalType *>(this);
00230 }
00231
00232 };
00233
00234
00235 /** A small array of 1, 2 or 3 elements with the implicit constructors */
00236 template <typename BasicType, typename FinalType, std::size_t Dims>
00237 struct small_array_123 : small_array<BasicType, FinalType, Dims> {
00238     static_assert(1 <= Dims && Dims <= 3,
00239         "Dimensions are between 1 and 3");
00240 };
00241
00242
00243 /** Use some specializations so that some function overloads can be
00244     determined according to some implicit constructors and to have an
00245     implicit conversion from/to BasicType (such as an int typically) if

```

```

00246     Dimensions = 1
00247 */
00248 template <typename BasicType, typename FinalType>
00249 struct small_array_123<BasicType, FinalType, 1>
00250 : public small_array<BasicType, FinalType, 1> {
00251     /// A 1-D constructor to have implicit conversion from from 1 integer
00252     /// and automatic inference of the dimensionality
00253     small_array_123(BasicType x) {
00254         (*this)[0] = x;
00255     }
00256
00257
00258     /// Keep other constructors
00259     small_array_123() = default;
00260
00261     using small_array<BasicType, FinalType, 1>::small_array;
00262
00263     /** Conversion so that an for example an id<1> can basically be used
00264         like an integer */
00265     operator BasicType() const {
00266         return (*this)[0];
00267     }
00268 };
00269
00270
00271 template <typename BasicType, typename FinalType>
00272 struct small_array_123<BasicType, FinalType, 2>
00273 : public small_array<BasicType, FinalType, 2> {
00274     /// A 2-D constructor to have implicit conversion from from 2 integers
00275     /// and automatic inference of the dimensionality
00276     small_array_123(BasicType x, BasicType y) {
00277         (*this)[0] = x;
00278         (*this)[1] = y;
00279     }
00280
00281
00282     /** Broadcasting constructor initializing all the elements with the
00283         same value
00284
00285         \todo Add to the specification of the range, id...
00286     */
00287     explicit small_array_123(BasicType e) : small_array_123 { e, e } { }
00288
00289
00290     /// Keep other constructors
00291     small_array_123() = default;
00292
00293     using small_array<BasicType, FinalType, 2>::small_array;
00294 };
00295
00296
00297 template <typename BasicType, typename FinalType>
00298 struct small_array_123<BasicType, FinalType, 3>
00299 : public small_array<BasicType, FinalType, 3> {
00300     /// A 3-D constructor to have implicit conversion from from 3 integers
00301     /// and automatic inference of the dimensionality
00302     small_array_123(BasicType x, BasicType y, BasicType z) {
00303         (*this)[0] = x;
00304         (*this)[1] = y;
00305         (*this)[2] = z;
00306     }
00307
00308
00309     /** Broadcasting constructor initializing all the elements with the
00310         same value
00311
00312         \todo Add to the specification of the range, id...
00313     */
00314     explicit small_array_123(BasicType e) : small_array_123 { e, e, e } { }
00315
00316
00317     /// Keep other constructors
00318     small_array_123() = default;
00319
00320     using small_array<BasicType, FinalType, 3>::small_array;
00321 };
00322
00323 /// @} End the helpers Doxygen group
00324
00325 }
00326 }
00327 }
00328
00329 /*
00330     # Some Emacs stuff:
00331     ### Local Variables:
00332     ### ispell-local-dictionary: "american"

```

```

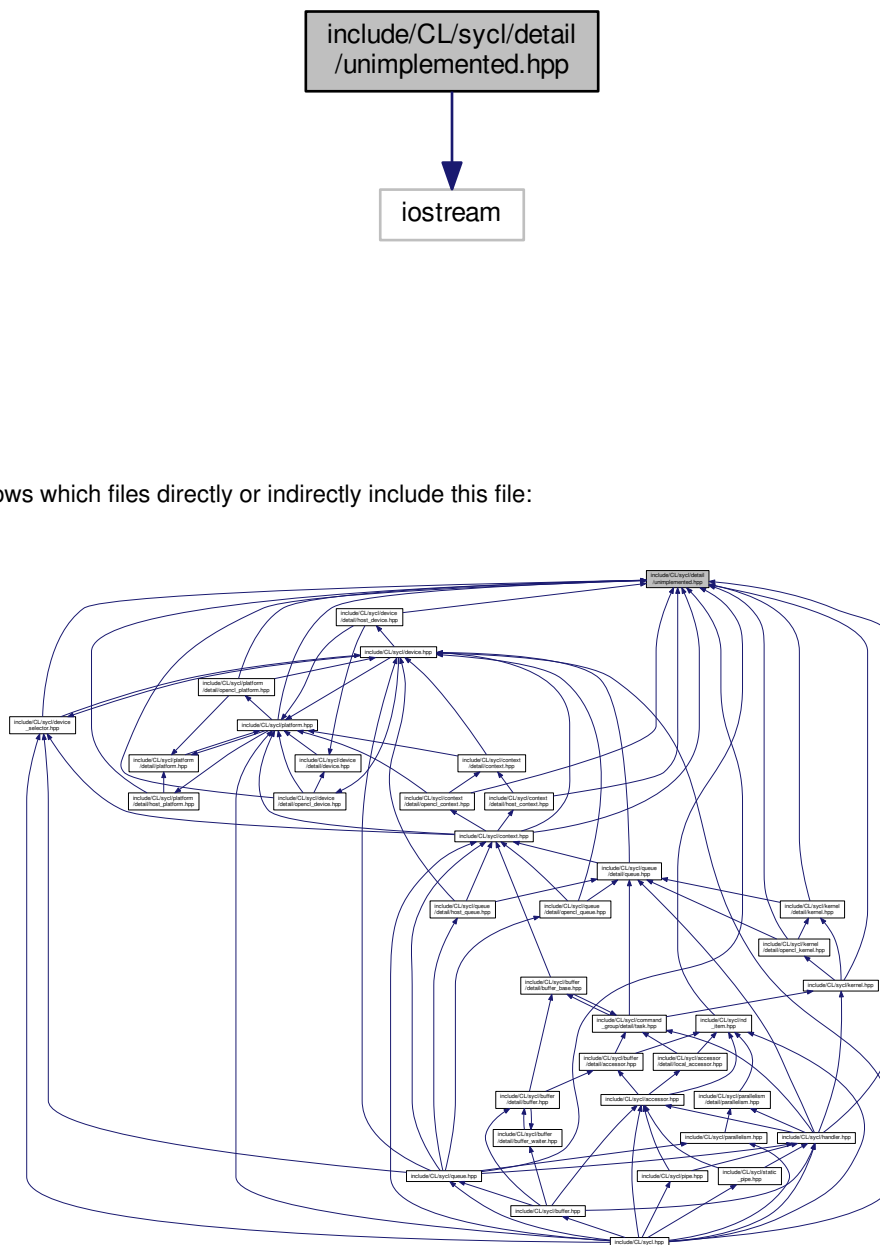
00333     ## eval: (flyspell-prog-mode)
00334     ## End:
00335 */
00336
00337 #endif // TRISYCL_SYCL_DETAIL_SMALL_ARRAY_HPP

```

## 11.59 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.*

## 11.60 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.
00024 */
00025 inline void unimplemented() {
00026 #ifndef NDEBUG
00027     std::cerr << "Error: using a non implemented feature!!!" << std::endl
00028               << "Please contribute to the open source implementation. :-)"
00029               << std::endl;
00030 #endif
00031 }
00032
00033 /// @} End the helpers Doxygen group
00034
00035 }
00036 }
00037 }
00038
00039 /*
00040     # Some Emacs stuff:
00041     ### Local Variables:
00042     ### ispell-local-dictionary: "american"
00043     ### eval: (flyspell-prog-mode)
00044     ### End:
00045 */
00046
00047 #endif // TRISYCL_SYCL_DETAIL_UNIMPLEMENTED_HPP

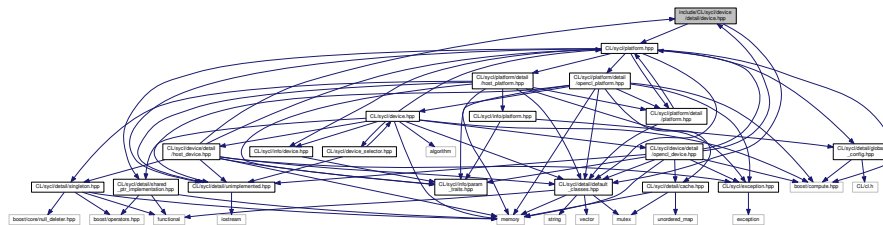
```

## 11.61 include/CL/sycl/device/detail/device.hpp File Reference

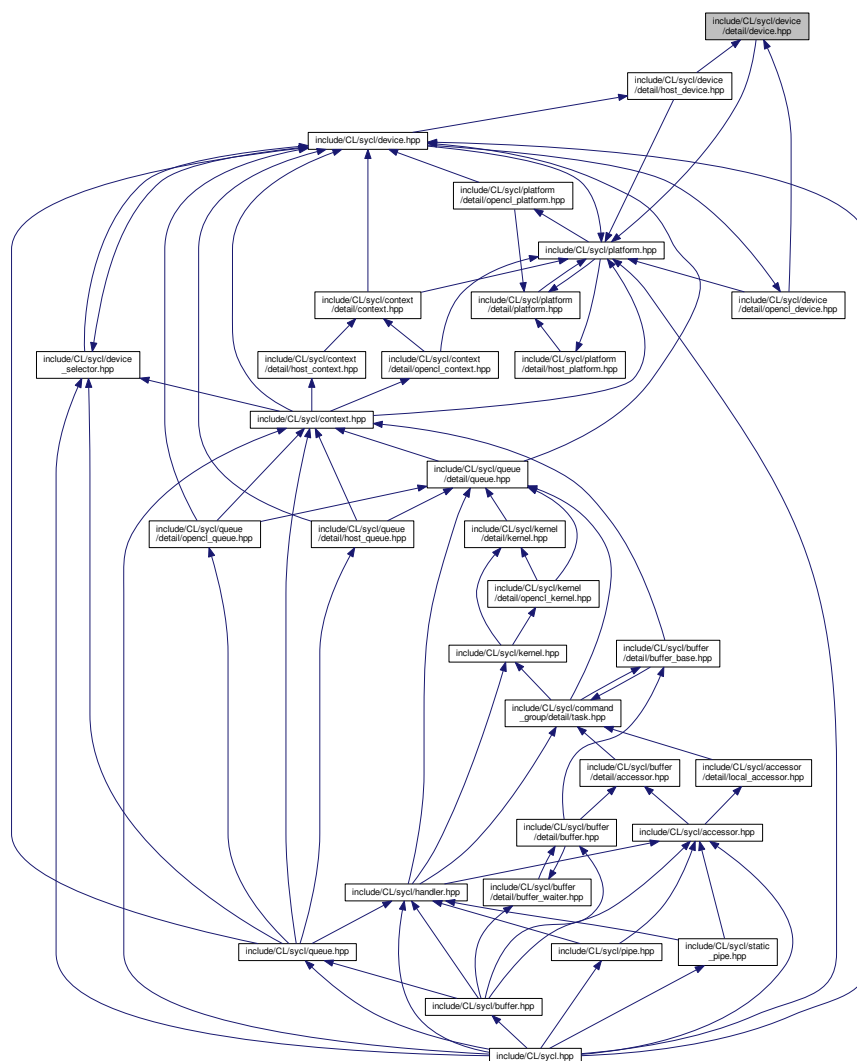
```
#include "CL/sycl/detail/default_classes.hpp"
```

```
#include "CL/sycl/platform.hpp"
```

Include dependency graph for device.hpp:



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`

## 11.62 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     public:
00027
00028     #ifndef TRISYCL_OPENCL
00029         /// Return the cl_device_id of the underlying OpenCL platform
00030         virtual cl_device_id get() const = 0;
00031
00032         /// Return the underlying Boost.Compute device, if any
00033         virtual boost::compute::device &get_boost_compute() = 0;
00034     #endif
00035     #endif
00036
00037         /// Return true if the device is a SYCL host device
00038         virtual bool is_host() const = 0;
00039
00040         /// Return true if the device is an OpenCL CPU device
00041         virtual bool is_cpu() const = 0;
00042
00043         /// Return true if the device is an OpenCL GPU device
00044         virtual bool is_gpu() const = 0;
00045
00046         /// Return true if the device is an OpenCL accelerator device
00047         virtual bool is_accelerator() const = 0;
00048
00049         /// Return the platform of device
00050         virtual cl::sycl::platform get_platform() const = 0;
00051
00052         /// Query the device for OpenCL info::device info
00053         /** \todo virtual cannot be templated
00054         template <typename T>

```



```

00061     virtual T get_info(info::device param) const = 0;
00062     */
00063
00064
00065     /// Specify whether a specific extension is supported on the device.
00066     virtual bool has_extension(const string_class &extension) const = 0;
00067
00068
00069     // Virtual to call the real destructor
00070     virtual ~device() {}
00071
00072 };
00073
00074 /// @} to end the execution Doxygen group
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_DEVICE_DETAIL_DEVICE_HPP

```

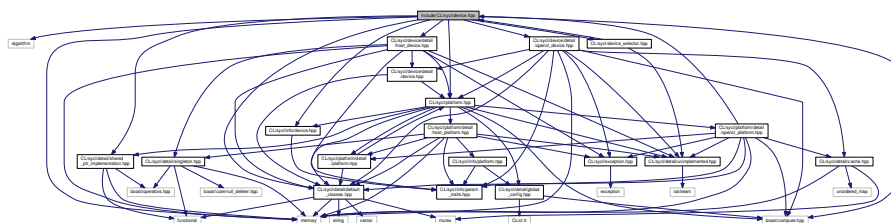
## 11.63 include/CL/sycl/device.hpp File Reference

```

#include <algorithm>
#include <memory>
#include <boost/compute.hpp>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/device/detail/host_device.hpp"
#include "CL/sycl/device/detail/opencl_device.hpp"
#include "CL/sycl/info/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/platform.hpp"

```

Include dependency graph for device.hpp:





## 11.64 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/opencv_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 */
00045     : public detail::shared_ptr_implementation<device, detail::device> {
00046     // The type encapsulating the implementation
00047     using implementation_t =
00048         detail::shared_ptr_implementation<device, detail::device>
00049     ;
00050 public:
00051     // Make the implementation member directly accessible in this class
00052     using implementation_t::implementation;
00053
00054     /// The default constructor uses the SYCL host device
00055     device() : implementation_t {
00056         detail::host_device::instance() } {}
00057
00058 #ifdef TRISYCL_OPENCL
00059     /** Construct a device class instance using cl_device_id of the
00060         OpenCL device
00061
00062         Return synchronous errors via the SYCL exception class.
00063
00064         Retain a reference to the OpenCL device and if this device was
00065         an OpenCL subdevice the device should be released by the caller
00066         when it is no longer needed.
00067     */
00068     device(cl_device_id device_id)
00069         : device { boost::compute::device { device_id } } {}
00070
00071     /** Construct a device class instance using a boost::compute::device
00072
00073         This is a triSYCL extension for boost::compute interoperation.
00074
00075         Return synchronous errors via the SYCL exception class.
00076     */
00077     device(const boost::compute::device &d)
00078         : implementation_t { detail::opencv_device::instance(d) } {}
00079 #endif
00080 }
00081 #endif

```

```

00082
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 #ifdef 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
00128
00129  /** Return the underlying Boost.Compute device if it is an
00130      OpenCL device
00131
00132      This is a triSYCL extension
00133  */
00134  boost::compute::device get_boost_compute() const {
00135      return implementation->get_boost_compute();
00136  }
00137 #endif
00138
00139
00140  /// Return true if the device is the SYCL host device
00141  bool is_host() const {
00142      return implementation->is_host();
00143  }
00144
00145
00146  /// Return true if the device is an OpenCL CPU device
00147  bool is_cpu() const {
00148      return implementation->is_cpu();
00149  }
00150
00151
00152  /// Return true if the device is an OpenCL GPU device
00153  bool is_gpu() const {
00154      return implementation->is_gpu();
00155  }
00156
00157
00158  /// Return true if the device is an OpenCL accelerator device
00159  bool is_accelerator() const {
00160      return implementation->is_accelerator();
00161  }
00162
00163
00164
00165  /** Return the device_type of a device
00166
00167      \todo Present in Boost.Compute, to be added to the specification
00168  */

```

```

00169     info::device_type type() const {
00170         if (is_host())
00171             return info::device_type::host;
00172         else if (is_cpu())
00173             return info::device_type::cpu;
00174         else if (is_gpu())
00175             return info::device_type::gpu;
00176         else if (is_accelerator())
00177             return info::device_type::accelerator;
00178         else
00179             // \todo Put a SYCL exception
00180             throw std::domain_error("Unknown cl::sycl::info::device_type");
00181     }
00182
00183
00184     /** Return the platform of device
00185
00186         Return synchronous errors via the SYCL exception class.
00187     */
00188     platform get_platform() const {
00189         return implementation->get_platform();
00190     }
00191
00192
00193     /** Return a list of all available devices
00194
00195         Return synchronous errors via SYCL exception classes.
00196     */
00197     #ifdef _MSC_VER
00198         inline
00199     #endif
00200     static vector_class<device>
00201     get_devices(info::device_type device_type =
00202     info::device_type::all)
00203     TRISYCL_WEAK_ATTRIB_SUFFIX;
00204
00205     /** Query the device for OpenCL info::device info
00206
00207         Return synchronous errors via the SYCL exception class.
00208
00209         \todo
00210     */
00211     template <typename T>
00212     T get_info(info::device param) const {
00213         //return implementation->get_info<Param>(param);
00214     }
00215
00216     /** Query the device for OpenCL info::device info
00217
00218         Return synchronous errors via the SYCL exception class.
00219
00220         \todo
00221     */
00222     template <info::device Param>
00223     inline auto get_info() const;
00224     /*{
00225         // Forward to the version where the info parameter is not a template
00226         //return get_info<typename info::param_traits_t<info::device, Param>>(Param);
00227         detail::unimplemented();
00228         return 0;
00229     }*/
00230
00231
00232     /// Test if a specific extension is supported on the device
00233     bool has_extension(const string_class &extension) const {
00234         return implementation->has_extension(extension);
00235     }
00236
00237
00238     #ifdef XYZTRISYCL_OPENCL
00239     /** Partition the device into sub devices based upon the properties
00240         provided
00241
00242         Return synchronous errors via SYCL exception classes.
00243
00244         \todo
00245     */
00246     vector_class<device>
00247     create_sub_devices(info::device_partition_type partition_type,
00248         info::device_partition_property partition_property,
00249         info::device_affinity_domain affinity_domain) const {
00250         return implementation->create_sub_devices(partition_type,
00251             partition_property,
00252             affinity_domain);
00253     }
00254     #endif

```

```

00255
00256 };
00257
00258
00259 template <>
00260 inline auto device::get_info<info::device::max_work_group_size>() const {
00261     return size_t { 8 };
00262 }
00263
00264 template <>
00265 inline auto device::get_info<info::device::max_compute_units>() const {
00266     return size_t { 8 };
00267 }
00268
00269 template <>
00270 inline auto device::get_info<info::device::device_type>() const {
00271     return info::device_type::cpu;
00272 }
00273
00274 template <>
00275 inline auto device::get_info<info::device::local_mem_size>() const {
00276     return size_t { 32000 };
00277 }
00278
00279 template <>
00280 inline auto device::get_info<info::device::max_mem_alloc_size>() const {
00281     return size_t { 32000 };
00282 }
00283
00284 template <>
00285 inline auto device::get_info<info::device::vendor>() const {
00286     return string_class {};
00287 }
00288
00289 template <>
00290 inline auto device::get_info<info::device::name>() const {
00291     return string_class {};
00292 }
00293
00294 template <>
00295 inline auto device::get_info<info::device::profile>() const {
00296     return string_class { "FULL_PROFILE" };
00297 }
00298
00299 /// @} to end the Doxygen group
00300
00301 }
00302 }
00303
00304
00305 /* Inject a custom specialization of std::hash to have the buffer
00306     usable into an unordered associative container
00307
00308     \todo Add this to the spec
00309 */
00310 namespace std {
00311
00312 template <> struct hash<cl::sycl::device> {
00313
00314     auto operator()(const cl::sycl::device &d) const {
00315         // Forward the hashing to the implementation
00316         return d.hash();
00317     }
00318
00319 };
00320
00321 }
00322
00323 /*
00324     # Some Emacs stuff:
00325     ### Local Variables:
00326     ### ispell-local-dictionary: "american"
00327     ### eval: (flyspell-prog-mode)
00328     ### End:
00329 */
00330
00331 #endif // TRISYCL_SYCL_DEVICE_HPP

```

## 11.65 include/CL/sycl/info/device.hpp File Reference

```
#include "CL/sycl/info/param_traits.hpp"
```

```
graph TD; A[include/CL/sycl/info/device.hpp] --> B[CL/sycl/info/param_traits.hpp]
```

[illegible]

- `cl`  
*The vector type to be used as SYCL vector.*
- `cl::sycl`
- `cl::sycl::info`

- using `cl::sycl::info::device_fp_config` = unsigned int
- using `cl::sycl::info::device_exec_capabilities` = unsigned int
- using `cl::sycl::info::device_queue_properties` = unsigned int

## Enumerations

- enum `cl::sycl::info::device_type` : unsigned int {  
`cl::sycl::info::device_type::cpu`, `cl::sycl::info::device_type::gpu`, `cl::sycl::info::device_type::accelerator`, `cl::sycl::info::device_type::custom`,  
`cl::sycl::info::device_type::defaults`, `cl::sycl::info::device_type::host`, `cl::sycl::info::device_type::opencl`, `cl::sycl::info::device_type::all` }

*Type of devices.*

- enum `cl::sycl::info::device` : int {  
`cl::sycl::info::device::device_type`, `cl::sycl::info::device::vendor_id`, `cl::sycl::info::device::max_compute_units`,  
`cl::sycl::info::device::max_work_item_dimensions`,  
`cl::sycl::info::device::max_work_item_sizes`, `cl::sycl::info::device::max_work_group_size`, `cl::sycl::info::device::preferred_vector_width_char`,  
`cl::sycl::info::device::preferred_vector_width_short`,  
`cl::sycl::info::device::preferred_vector_width_int`, `cl::sycl::info::device::preferred_vector_width_long_long`, `cl::sycl::info::device::preferred_vector_width_float`,  
`cl::sycl::info::device::preferred_vector_width_double`,  
`cl::sycl::info::device::preferred_vector_width_half`, `cl::sycl::info::device::native_vector_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_queue_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` }

## 11.66 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,

```

```

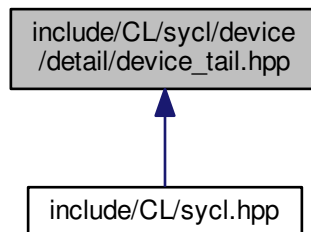
00063     preferred_vector_width_float,
00064     preferred_vector_width_double,
00065     preferred_vector_width_half,
00066     native_vector_witdth_char,
00067     native_vector_witdth_short,
00068     native_vector_witdth_int,
00069     native_vector_witdth_long_long,
00070     native_vector_witdth_float,
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 };
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     Ll_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
```

## 11.67 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](#)

## 11.68 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

```

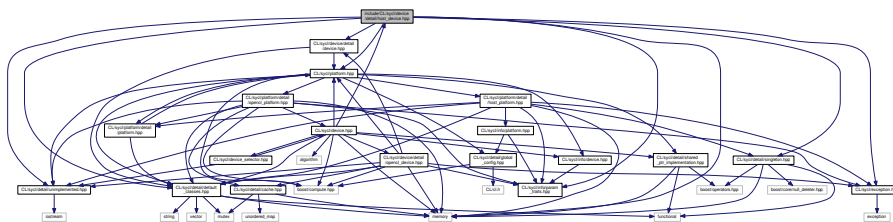
## 11.69 include/CL/sycl/device/detail/host\_device.hpp File Reference

```

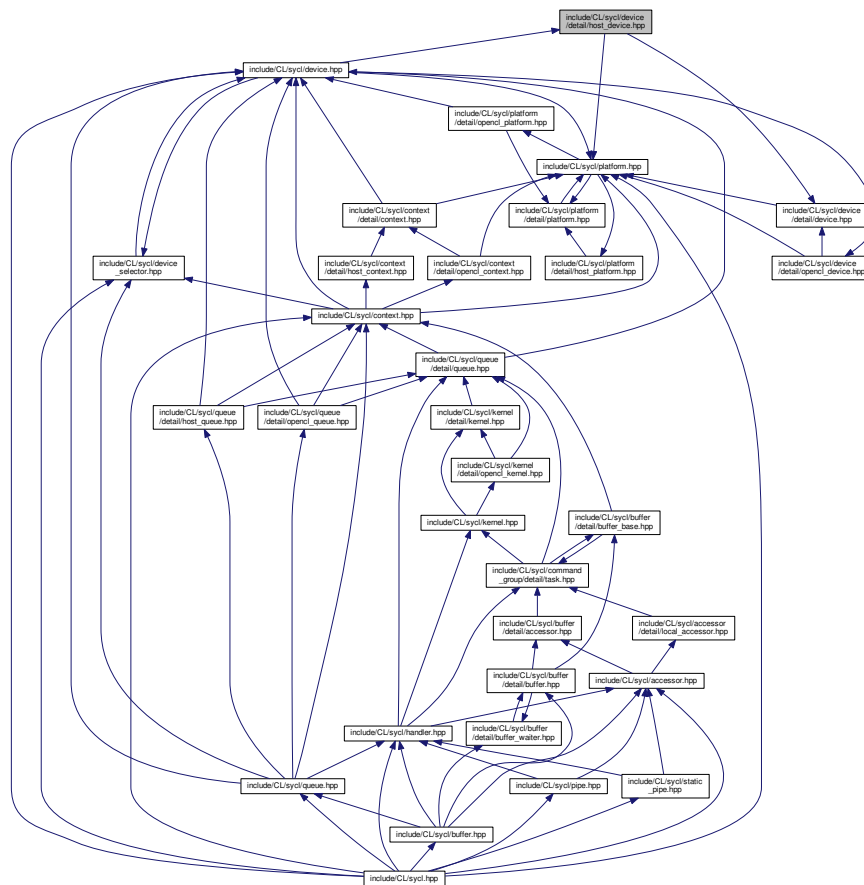
#include <memory>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/singleton.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device/detail/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform.hpp"

```

Include dependency graph for host\_device.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class `cl::sycl::detail::host_device`  
*SYCL host device.*

## Namespaces

- `cl`  
*The vector type to be used as SYCL vector.*
- `cl::sycl`
- `cl::sycl::detail`

## 11.70 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 #ifndef 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
00045 /** Return the underlying Boost.Compute device
00046
00047     This throws an error since there is no OpenCL device associated
00048     to the host device.
00049 */
00050 boost::compute::device &get_boost_compute() override {
00051     throw non_cl_error("The host device has no underlying OpenCL device");
00052 }
00053 #endif
00054
00055 /// Return true since the device is a SYCL host device
00056 bool is_host() const override {
00057     return true;
00058 }
00059
00060 /// Return false since the host device is not an OpenCL CPU device
00061 bool is_cpu() const override {
00062     return false;
00063 }
00064
00065 /// Return false since the host device is not an OpenCL GPU device
00066 bool is_gpu() const override {
00067     return false;
00068 }
00069
00070 /// Return false since the host device is not an OpenCL accelerator device
00071 bool is_accelerator() const override {
00072     return false;
00073 }
00074
00075 /** Return the platform of device
00076
00077     Return synchronous errors via the SYCL exception class.
00078
00079     \todo To be implemented
00080 */
00081 cl::sycl::platform get_platform() const override {
00082     detail::unimplemented();
00083     return {};
00084 }
00085
00086 #if 0
00087 /** Query the device for OpenCL info::device info

```

```

00095
00096     Return synchronous errors via the SYCL exception class.
00097
00098     \todo To be implemented
00099 */
00100 template <info::device Param>
00101 typename info::param_traits<info::device, Param>::type
00102 get_info() const override {
00103     detail::unimplemented();
00104     return {};
00105 }
00106 #endif
00107
00108 /** Specify whether a specific extension is supported on the device
00109
00110     \todo To be implemented
00111 */
00112 bool has_extension(const string_class &extension) const override {
00113     detail::unimplemented();
00114     return {};
00115 }
00116
00117 };
00118 };
00119
00120 /// @} to end the execution Doxygen group
00121
00122 }
00123 }
00124 }
00125
00126 /*
00127     # Some Emacs stuff:
00128     ### Local Variables:
00129     ### ispell-local-dictionary: "american"
00130     ### eval: (flyspell-prog-mode)
00131     ### End:
00132 */
00133
00134 #endif // TRISYCL_SYCL_DEVICE_DETAIL_HOST_DEVICE_HPP

```

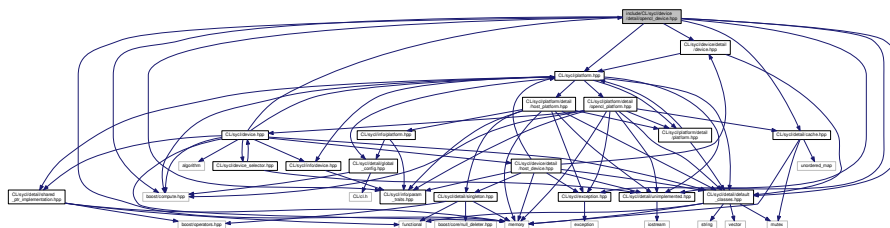
## 11.71 include/CL/sycl/device/detail/opencl\_device.hpp File Reference

```

#include <memory>
#include <boost/compute.hpp>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device/detail/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform.hpp"

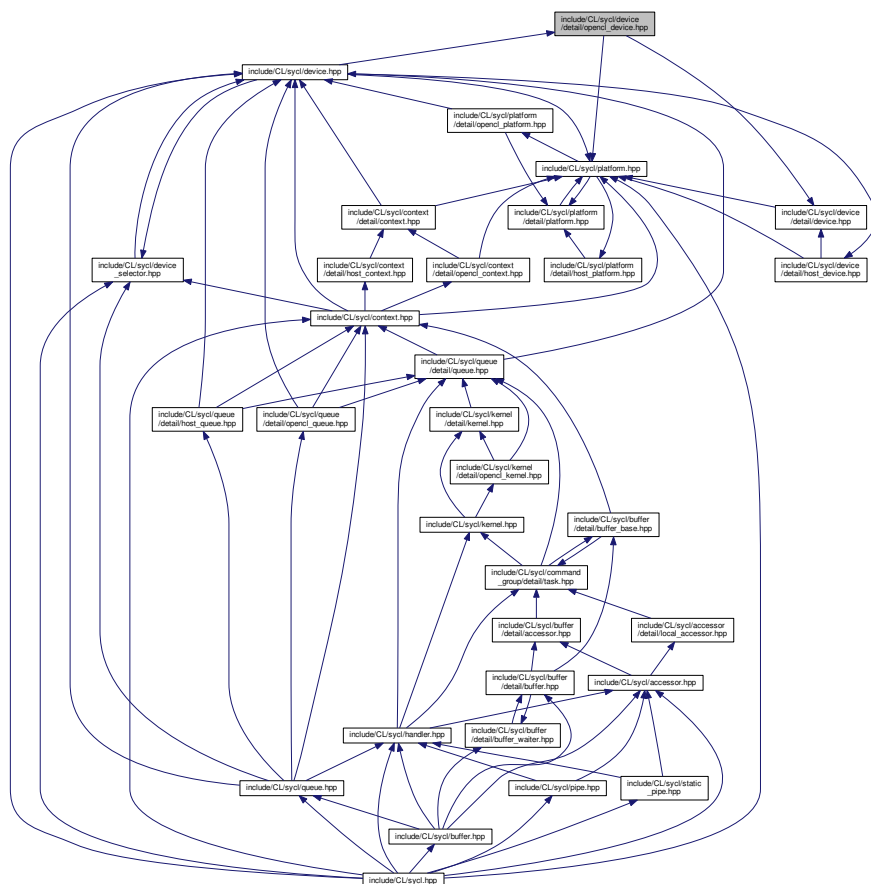
```

Include dependency graph for opencl\_device.hpp:





This graph shows which files directly or indirectly include this file:



## Classes

- class `cl::sycl::detail::opencl_device`  
*SYCL OpenCL device.*

## Namespaces

- `cl`  
*The vector type to be used as SYCL vector.*
- `cl::sycl`
- `cl::sycl::detail`

## 11.72 opengl\_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>
00041     cache;
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     /// Return the underlying Boost.Compute device
00050     boost::compute::device &get_boost_compute() override {
00051         return d;
00052     }
00053
00054     /// Return false since an OpenCL device is not the SYCL host device
00055     bool is_host() const override {
00056         return false;
00057     }
00058
00059     /// Test if the OpenCL is a CPU device
00060     bool is_cpu() const override {
00061         // Even in Boost.Compute the type is a bit-field, so use & instead of ==
00062         return d.type() & boost::compute::device::cpu;
00063     }
00064
00065     /// Test if the OpenCL is a GPU device
00066     bool is_gpu() const override {
00067         // Even in Boost.Compute the type is a bit-field, so use & instead of ==
00068         return d.type() & boost::compute::device::gpu;
00069     }
00070
00071     /// Test if the OpenCL is an accelerator device
00072     bool is_accelerator() const override {
00073         // Even in Boost.Compute the type is a bit-field, so use & instead of ==
00074         return d.type() & boost::compute::device::accelerator;
00075     }
00076
00077     /** Return the platform of device
00078
00079         Return synchronous errors via the SYCL exception class.
00080     */
00081     cl::sycl::platform get_platform() const override {
00082         return d.platform();
00083     }
00084
00085     #if 0
00086     /** Query the device for OpenCL info::device info
00087
00088     */
00089
00090
00091
00092
00093

```

```

00094         Return synchronous errors via the SYCL exception class.
00095
00096         \todo To be implemented
00097     */
00098     template <info::device Param>
00099     typename info::param_traits<info::device, Param>::type
00100     get_info() const override {
00101         detail::unimplemented();
00102         return {};
00103     }
00104 #endif
00105
00106     /** Specify whether a specific extension is supported on the device.
00107
00108         \todo To be implemented
00109     */
00110     bool has_extension(const string_class &extension) const override {
00111         detail::unimplemented();
00112         return {};
00113     }
00114
00115
00116     ///// Get a singleton instance of the opencil_device
00117     static std::shared_ptr<opencil_device>
00118     instance(const boost::compute::device &d) {
00119         return cache.get_or_register(d.id(),
00120                                     [&] { return new opencil_device { d }; });
00121     }
00122
00123 private:
00124
00125     /// Only the instance factory can build it
00126     opencil_device(const boost::compute::device &d) : d { d } {}
00127
00128 public:
00129
00130     /// Unregister from the cache on destruction
00131     ~opencil_device() override {
00132         cache.remove(d.id());
00133     }
00134
00135 };
00136
00137 /* Allocate the cache here but since this is a pure-header library,
00138    use a weak symbol so that only one remains when SYCL headers are
00139    used in different compilation units of a program
00140 */
00141 TRISYCL_WEAK_ATTRIB_PREFIX
00142 detail::cache<cl_device_id, detail::opencil_device>
00143 opencil_device::cache
00144 TRISYCL_WEAK_ATTRIB_SUFFIX;
00145 }
00146 }
00147 }
00148
00149 /*
00150     # Some Emacs stuff:
00151     ### Local Variables:
00152     ### ispell-local-dictionary: "american"
00153     ### eval: (flyspell-prog-mode)
00154     ### End:
00155 */
00156
00157 #endif // TRISYCL_SYCL_DEVICE_DETAIL_OPENCL_DEVICE_HPP

```

## 11.73 include/CL/sycl/device\_selector.hpp File Reference

```

#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"

```

[illegible]

- class `cl::sycl::device_selector`  
*The SYCL heuristics to select a device. [More...](#)*

- `cl`  
*The vector type to be used as SYCL vector.*
- `cl::sycl`

## 11.74 device\_selector.hpp

```

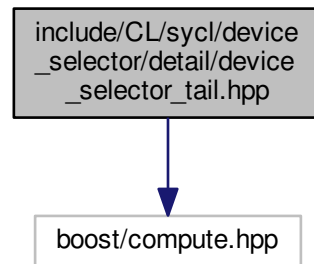
00001 #ifndef TRISYCL_SYCL_DEVICE_SELECTOR_HPP
00002 #define TRISYCL_SYCL_DEVICE_SELECTOR_HPP
00003
00004 /** \file The OpenCL SYCL device_selector
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT 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 public:
00028
00029     /** Returns a selected device using the functor operator defined in
00030         sub-classes operator()(const device &dev)
00031
00032         \todo Remove this from specification
00033     */
00034 void /* device */ select_device() const {
00035     // return {};
00036 }
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

```

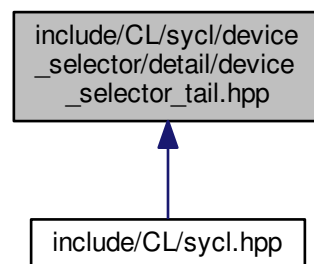
## 11.75 include/CL/sycl/device\_selector/detail/device\_selector\_tail.hpp File Reference

```
#include <boost/compute.hpp>
```

Include dependency graph for device\_selector\_tail.hpp:



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\\_typename\\_selector< DeviceType >](#)  
Select a device by template device\_type parameter. [More...](#)

## Namespaces

- [cl](#)  
The vector type to be used as SYCL vector.
- [cl::sycl](#)

## Typedefs

- using `cl::sycl::default_selector` = `device_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.*

## 11.76 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 #ifdef TRISYCL_OPENCL
00017 #include <boost/compute.hpp>
00018 #endif
00019
00020 namespace cl {
00021 namespace sycl {
00022
00023 /** \addtogroup execution Platforms, contexts, devices and queues
00024     @{
00025 */
00026
00027
00028 /** A device selector by device_type
00029
00030     \todo To be added to the specification
00031 */
00032 class device_type_selector : public device_selector {
00033
00034 private:
00035
00036     /// The device_type to select
00037     info::device_type device_type;
00038
00039     /** Cache the default device to select with the default device
00040         selector.
00041
00042         This is the host device at construction time and remains as is
00043         if there is no openCL device */
00044     device default_device;
00045
00046 public:
00047
00048     device_type_selector(info::device_type device_type)
00049         : device_type { device_type } {
00050         // The default device selection heuristic
00051 #ifdef TRISYCL_OPENCL
00052         if (device_type == info::device_type::defaults) {
00053             // Ask Boost.Compute for the default OpenCL device
00054             try {
00055                 default_device = boost::compute::system::default_device();
00056             }
00057             catch (...) {
00058                 /* If there is no OpenCL device, just keep the
00059                    default-constructed device, which is the host device */
00060             }
00061         }
00062 #endif

```

```

00063     }
00064
00065
00066     // To select only the requested device_type
00067     int operator()(const device &dev) const override {
00068         if (device_type == info::device_type::all)
00069             // All devices fit all
00070             return 1;
00071
00072         if (device_type == info::device_type::defaults)
00073             // Only select the default device
00074             return dev == default_device ? 1 : -1;
00075
00076         if (device_type == info::device_type::opencl)
00077             // For now, any non host device is an OpenCL device
00078             return dev.is_host() ? -1 : 1;
00079
00080         return dev.type() == device_type ? 1 : -1;
00081     }
00082
00083 };
00084
00085
00086 /** Select a device by template device_type parameter
00087
00088     \todo To be added to the specification
00089 */
00090 template <info::device_type DeviceType>
00091 class device_typename_selector : public
00092     device_type_selector {
00093 public:
00094
00095     device_typename_selector() : device_type_selector {
00096         DeviceType } {}
00097 };
00098
00099
00100 /** Devices selected by heuristics of the system
00101
00102     If no OpenCL device is found then it defaults to the SYCL host device.
00103
00104     To influence the default device selection, use the Boost.Compute
00105     environment variables:
00106
00107     - \c BOOST_COMPUTE_DEFAULT_DEVICE
00108
00109     - \c BOOST_COMPUTE_DEFAULT_DEVICE_TYPE
00110
00111     - \c BOOST_COMPUTE_DEFAULT_PLATFORM
00112
00113     - \c BOOST_COMPUTE_DEFAULT_VENDOR
00114 */
00115 using default_selector =
00116     device_typename_selector<info::device_type::defaults>;
00117
00118 /** Select devices according to device type info::device::device_type::gpu
00119     from all the available OpenCL devices.
00120
00121     If no OpenCL GPU device is found the selector fails.
00122
00123     Select the best GPU, if any.
00124 */
00125 using gpu_selector =
00126     device_typename_selector<info::device_type::gpu>;
00127
00128 /** Select devices according to device type info::device::device_type::cpu
00129     from all the available devices and heuristics
00130
00131     If no OpenCL CPU device is found the selector fails.
00132 */
00133 using cpu_selector =
00134     device_typename_selector<info::device_type::cpu>;
00135
00136 /** Selects the SYCL host CPU device that does not require an OpenCL
00137     runtime
00138 */
00139 using host_selector =
00140     device_typename_selector<info::device_type::host>;
00141
00142 /// @} to end the execution Doxygen group
00143 }

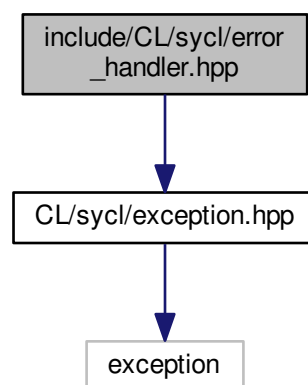
```



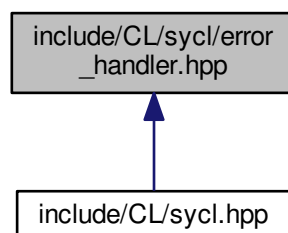
```
00144 }
00145
00146 /*
00147  # Some Emacs stuff:
00148  ### Local Variables:
00149  ### ispell-local-dictionary: "american"
00150  ### eval: (flyspell-prog-mode)
00151  ### End:
00152 */
00153
00154 #endif // TRISYCL_SYCL_DEVICE_SELECTOR_DETAIL_DEVICE_SELECTOR_TAIL_HPP
```

## 11.77 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](#)

## 11.78 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
00028 /** User supplied error handler to call a user-provided function when an
00029     error happens from a SYCL object that was constructed with this error
00030     handler
00031 */
00032 struct error_handler {
00033     /** The method to define to be called in the case of an error
00034
00035         \todo Add "virtual void" to the specification
00036     */
00037     virtual void report_error(exception &error) = 0;
00038
00039     /** Add a default_handler to be used by default
00040
00041         \todo add this concept to the specification?
00042     */
00043     static trisycl::default_error_handler
    default_handler;
00044
00045     virtual ~error_handler() = 0;
00046 };
00047
00048
00049 namespace trisycl {
00050
00051     struct default_error_handler : error_handler {
00052     void report_error(exception &) override {
00053     }
00054     };
00055
00056 
```

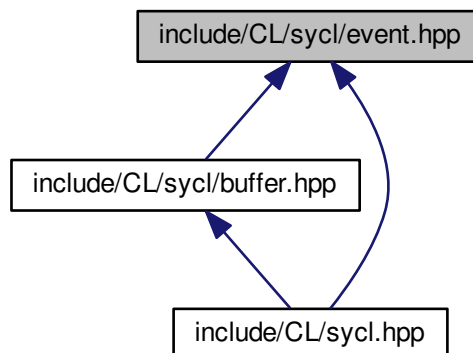
```

00057 }
00058
00059 // \todo finish initialization
00060 //error_handler::default_handler = nullptr;
00061
00062
00063 /// @} End the error_handling Doxygen group
00064
00065 }
00066 }
00067
00068 /*
00069  # Some Emacs stuff:
00070  ### Local Variables:
00071  ### ispell-local-dictionary: "american"
00072  ### eval: (flyspell-prog-mode)
00073  ### End:
00074 */
00075
00076 #endif // TRISYCL_SYCL_ERROR_HANDLER_HPP

```

## 11.79 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](#)

## 11.80 event.hpp

```

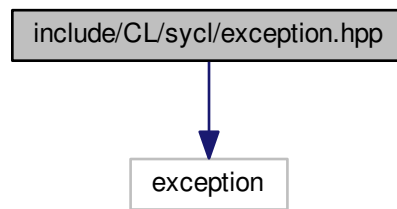
00001 #ifndef TRISYCL_SYCL_EVENT_HPP
00002 #define TRISYCL_SYCL_EVENT_HPP
00003
00004 /** \file The event class
00005
00006     Ronan at keryell dot FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011 namespace cl {
00012 namespace sycl {
00013
00014 class event {
00015
00016 public:
00017
00018     event() = default;
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

```

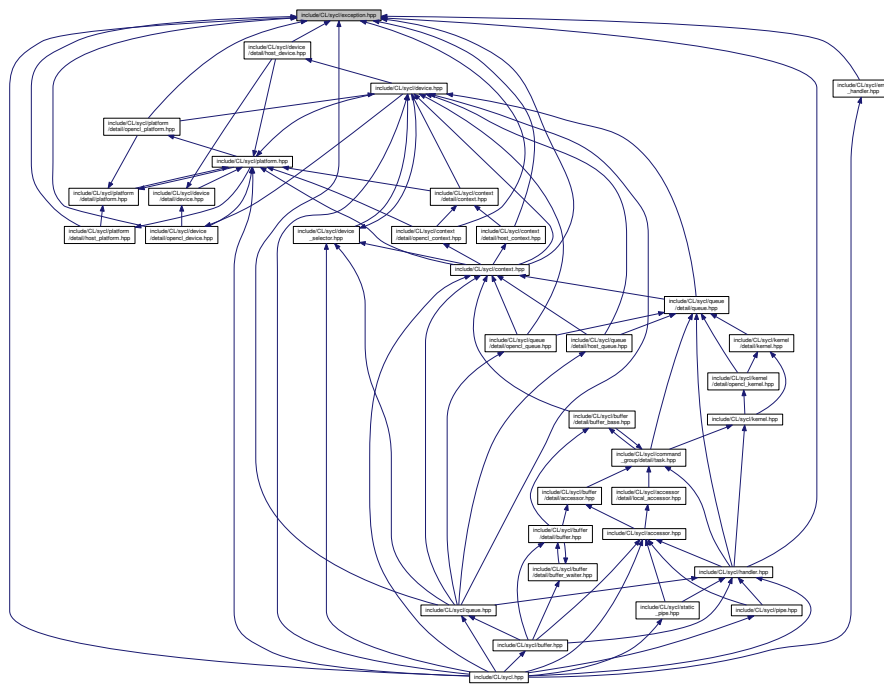
## 11.81 include/CL/sycl/exception.hpp File Reference

```
#include <exception>
```

Include dependency graph for exception.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [cl::sycl::exception\\_list](#)  
Exception list to store several exceptions. [More...](#)
- class [cl::sycl::exception](#)  
Encapsulate a SYCL error information. [More...](#)
- class [cl::sycl::cl\\_exception](#)  
Returns the OpenCL error code encapsulated in the exception. [More...](#)
- struct [cl::sycl::async\\_exception](#)  
An error stored in an [exception\\_list](#) for asynchronous errors. [More...](#)
- class [cl::sycl::runtime\\_error](#)

- class `cl::sycl::kernel_error`  
Error that occurred before or while 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 >`

## 11.82 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
00043     /// The error message to return
00044     string_class message;
00045
00046 public:
00047
00048     /// Construct an exception with a message for internal use
00049     exception(const string_class &message) : message { message } {}
00050
00051     /// Returns a descriptive string for the error, if available
00052     string_class what() const {
00053         return message;
00054     }
00055
00056
00057     /** Returns the context that caused the error
00058
00059         Returns nullptr if not a buffer error.
00060
00061         \todo Cannot return nullptr. Use optional? Use a specific exception type?
00062     */
00063     //context get_context()
00064
00065 };
00066
00067
00068 /// Returns the OpenCL error code encapsulated in the exception
00069 class cl_exception : public exception {
00070
00071 #ifndef TRISYCL_OPENCL
00072     /// The OpenCL error code to return
00073
00074     cl_int cl_code;
00075
00076 public:
00077
00078     /** Construct an exception with a message and OpenCL error code for
00079         internal use */
00080     cl_exception(const string_class &message, cl_int cl_code)
00081         : exception { message }, cl_code { cl_code } {}
00082
00083     // thrown as a result of an OpenCL API error code

```

```
00084     cl_int get_cl_code() const {
00085         return cl_code;
00086     }
00087 #endif
00088
00089 };
00090
00091
00092 /// An error stored in an exception_list for asynchronous errors
00093 struct async_exception : exception {
00094     using exception::exception;
00095 };
00096
00097
00098 class runtime_error : public exception {
00099     using exception::exception;
00100 };
00101
00102
00103 /// Error that occurred before or while enqueueing the SYCL kernel
00104 class kernel_error : public runtime_error {
00105     using runtime_error::runtime_error;
00106 };
00107
00108
00109 /// Error regarding the cl::sycl::accessor objects defined
00110 class accessor_error : public runtime_error {
00111     using runtime_error::runtime_error;
00112 };
00113
00114
00115 /// Error regarding the cl::sycl::nd_range specified for the SYCL kernel
00116 class nd_range_error : public runtime_error {
00117     using runtime_error::runtime_error;
00118 };
00119
00120
00121 /// Error regarding associated cl::sycl::event objects
00122 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

```

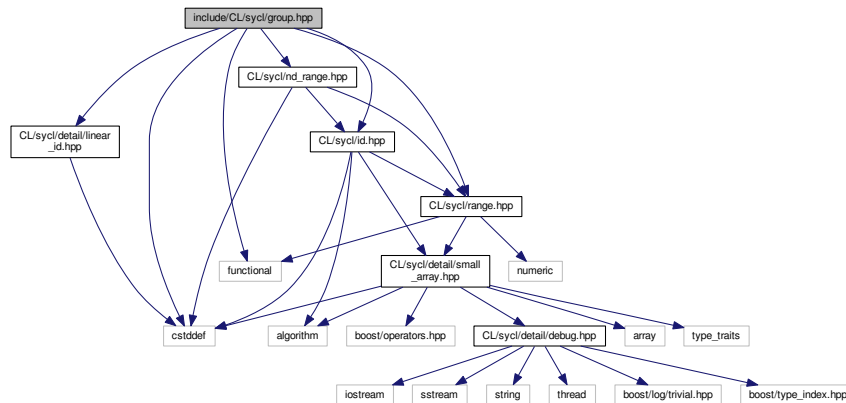
## 11.83 include/CL/sycl/group.hpp File Reference

```

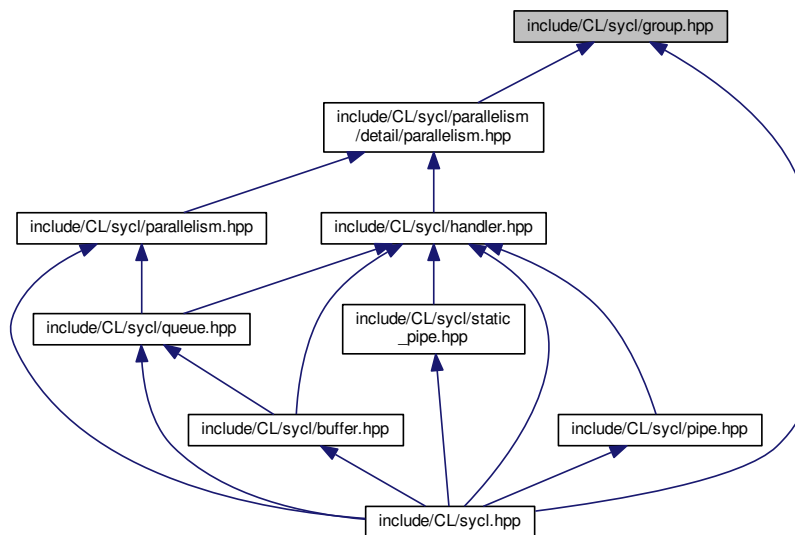
#include <cstddef>
#include <functional>
#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"

```

Include dependency graph for group.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- struct `cl::sycl::group< Dimensions >`  
A group index used in a `parallel_for_workitem` to specify a `work_group`. [More...](#)
- struct `cl::sycl::group< Dimensions >`  
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<int Dimensions = 1, typename ParallelForFuncor >`  
`void cl::sycl::detail::parallel\_for\_workitem (const group< Dimensions > &g, ParallelForFuncor f)`  
*Implement the loop on the work-items inside a work-group.*

## 11.84 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 <int Dimensions = 1>
00024 struct group;
00025
00026 namespace detail {
00027
00028 template <int 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 <int Dimensions>
00041 struct group {
00042     /// \todo add this Boost::multi_array or STL concept to the
00043     /// specification?
00044     static constexpr auto dimensionality = Dimensions;
00045
00046 private:
00047     /// The coordinate of the group item
00048     id<Dimensions> group_id;
00049
00050     /// Keep a reference on the nd_range to serve potential query on it
00051     nd_range<Dimensions> ndr;
00052
00053 public:
00054
00055     /** Create a group from an nd_range<> with a 0 id<>
00056
00057         \todo This should be private since it is only used by the triSYCL
00058         implementation
00059     */
00060     group(const nd_range<Dimensions> &ndr) : ndr { ndr } {}
00061
00062
00063     /** Create a group from an id and a nd_range<>
00064
00065         \todo This should be private somehow, but it is used by the
00066         validation infrastructure
00067     */
00068     group(const id<Dimensions> &i, const nd_range<Dimensions> &ndr) :
00069         group_id { i }, ndr { ndr } {}
00070
00071

```

```

00072
00073 /** To be able to copy and assign group, use default constructors too
00074
00075     \todo Make most of them protected, reserved to implementation
00076 */
00077 group() = default;
00078
00079
00080 /** Return an id representing the index of the group within the nd_range
00081     for every dimension
00082 */
00083 id<Dimensions> get() const { return group_id; }
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<Dimensions> 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<Dimensions> get_global_range() const {
00123     return get_nd_range().get_global();
00124 }
00125
00126
00127 /// Return element dimension from the constituent global range
00128 size_t get_global_range(int dimension) const {
00129     return get_global_range()[dimension];
00130 }
00131
00132
00133 /** Get the local range for this work_group
00134
00135     \todo Add to the specification
00136 */
00137 range<Dimensions> get_local_range() const {
00138     return get_nd_range().get_local();
00139 }
00140
00141
00142 /** Return element dimension from the constituent local range
00143
00144     \todo Add to the specification
00145 */
00146 size_t get_local_range(int dimension) const {
00147     return get_local_range()[dimension];
00148 }
00149
00150
00151 /** Get the offset of the NDRange
00152
00153     \todo Add to the specification
00154 */
00155 id<Dimensions> get_offset() const { return get_nd_range().get_offset(); }
00156
00157
00158 /** Get the offset of the NDRange

```

```

00159
00160     \todo Add to the specification
00161     */
00162     size_t get_offset(int dimension) const { return get_offset()[dimension]; }
00163
00164
00165     /// \todo Also provide this access to the current nd_range
00166     nd_range<Dimensions> get_nd_range() const { return ndr; }
00167
00168
00169     /** Get a linearized version of the group ID
00170
00171     */
00172     size_t get_linear() const {
00173         return detail::linear_id(get_group_range(), get());
00174     }
00175
00176
00177     /** Loop on the work-items inside a work-group
00178
00179     \todo Add this method in the specification
00180     */
00181     void parallel_for_work_item(std::function<void(
00182         nd_item<dimensionality>> f)
00183         const {
00184             detail::parallel_for_workitem(*this, f);
00185         }
00186
00187     /** Loop on the work-items inside a work-group
00188
00189     \todo Add this method in the specification
00190     */
00191     void parallel_for_work_item(std::function<void(
00192         item<dimensionality>> f)
00193         const {
00194             auto item_adapter = [=] (nd_item<dimensionality> ndi) {
00195                 item<dimensionality> i = ndi.get_item();
00196                 f(i);
00197             };
00198             detail::parallel_for_workitem(*this, item_adapter);
00199         }
00200 };
00201
00202 /// @} End the parallelism Doxygen group
00203 }
00204 }
00205 }
00206
00207 /*
00208     # Some Emacs stuff:
00209     ### Local Variables:
00210     ###   ispell-local-dictionary: "american"
00211     ###   eval: (flyspell-prog-mode)
00212     ### End:
00213 */
00214
00215 #endif // TRISYCL_SYCL_GROUP_HPP

```

## 11.85 include/CL/sycl/handler.hpp File Reference

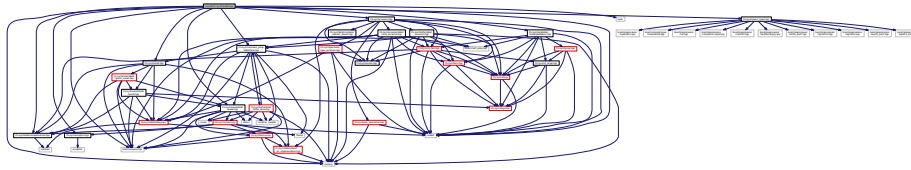
```

#include <cstddef>
#include <memory>
#include <tuple>
#include <boost/compute.hpp>
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/kernel.hpp"
#include "CL/sycl/opencl_types.hpp"
#include "CL/sycl/parallelism/detail/parallelism.hpp"

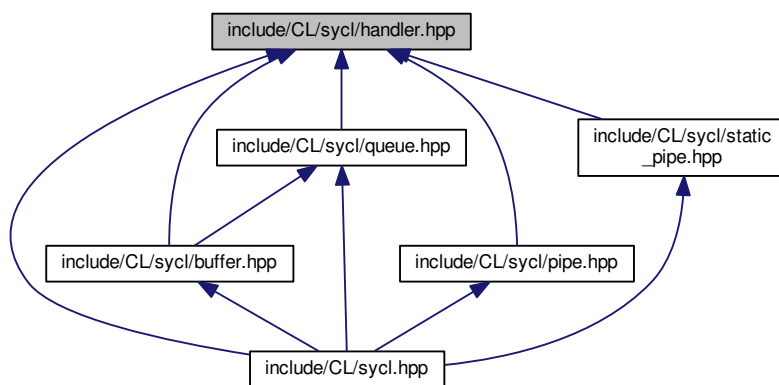
```

```
#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.*

### 11.85.1 Macro Definition Documentation

#### 11.85.1.1 TRISYCL\_parallel\_for\_functor\_GLOBAL

```
#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 range<>  |
| <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 206 of file [handler.hpp](#).

#### 11.85.1.2 TRISYCL\_ParallelForFunctor\_GLOBAL\_OFFSET

```
#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);
    }});
}

```

### 11.85.1.3 TRISYCL\_ParallelForKernel\_RANGE

```

#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 383 of file [handler.hpp](#).

## 11.86 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/opencl_types.hpp"
00026 #include "CL/sycl/parallelism/detail/parallelism.hpp"
00027 #include "CL/sycl/queue/detail/queue.hpp"
00028
00029 namespace cl {
00030 namespace sycl {
00031
00032 /** \addtogroup execution Platforms, contexts, devices and queues
00033     @{
00034 */
00035
00036 /** Command group handler class
00037
00038     A command group handler object can only be constructed by the SYCL runtime.
00039
00040     All of the accessors defined in the command group scope take as a
00041     parameter an instance of the command group handler and all the kernel
00042     invocation functions are methods of this class.
00043 */
00044 class handler : public detail::debug<handler> {
00045 public:
00046
00047     /** Attach the task and accessors to it.
00048     */
00050     std::shared_ptr<detail::task> task;
00051
00052
00053     /** Create a command group handler from the queue detail
00054
00055         The queue detail is used to track kernel completion.
00056
00057         Note that this is an implementation dependent constructor. Normal
00058         users cannot construct handler from scratch.
00059
00060         \todo Make this constructor private
00061     */
00062     handler(const std::shared_ptr<detail::queue> &q) {
00063         // Create a new task for this command_group
00064         task = std::make_shared<detail::task>(q);
00065     }
00066
00067
00068 #ifndef TRISYCL_OPENCL
00069     /** Set accessor kernel arg for an OpenCL kernel which is used through the
00070         SYCL/OpenCL interop interface
00071
00072         The index value specifies which parameter of the OpenCL kernel is
00073         being set and the accessor object, which OpenCL buffer or image is
00074         going to be given as kernel argument.
00075
00076         \todo Update the specification to use a ref && to the accessor instead?
00077
00078         \todo It is not that clean to have set_arg() associated to a
00079         command handler. Rethink the specification?
00080
00081         \todo It seems more logical to have these methods on kernel instead
00082     */
00083     template <typename DataType,
00084               int Dimensions,
00085               access::mode Mode,
00086               access::target Target = access::target::global_buffer>
00087     void set_arg(int arg_index,
00088                 accessor<DataType, Dimensions, Mode, Target> &&
00089                 acc_obj) {
00090         /** Think about setting the kernel argument before actually calling
00091             the kernel.
00092
00093             Explicitly capture task by copy instead of having this captured
00094             by reference and task by reference by side effect */
00094         task->add_prelude([=, task = task] {
00095             task->get_kernel().get_boost_compute()
00096                 .set_arg(arg_index, acc_obj.implementation->get_cl_buffer());
00097         });
00098     }
00099
00100
00101     /** Set kernel args for an OpenCL kernel which is used through the

```

```

00102     SYCL/OpenCL interoperability interface with a wrapper type
00103 */
00104 template <typename T, typename = std::enable_if_t<is_wrapper<T>::value> >
00105 void set_arg(int arg_index, T && scalar_value) {
00106     /* Explicitly capture task by copy instead of having this captured
00107        by reference and task by reference by side effect */
00108     task->add_prelude([=, task = task] {
00109         task->get_kernel().get_boost_compute()
00110             .set_arg(arg_index, scalar_value.unwrap());
00111     });
00112 }
00113
00114 /** Set kernel args for an OpenCL kernel which is used through the
00115     SYCL/OpenCL interoperability interface without a wrapper type
00116 */
00117 template <typename T>
00118 std::enable_if_t<!is_wrapper<T>::value>
00119 set_arg(int arg_index, T && scalar_value) {
00120     /* Explicitly capture task by copy instead of having this captured
00121        by reference and task by reference by side effect */
00122     task->add_prelude([=, task = task] {
00123         task->get_kernel().get_boost_compute()
00124             .set_arg(arg_index, scalar_value);
00125     });
00126 }
00127 }
00128
00129 private:
00130
00131     /// Helper to individually call set_arg() for each argument
00132     template <std::size_t... Is, typename... Ts>
00133     void dispatch_set_arg(std::index_sequence<Is...>, Ts&&... args) {
00134         // Use an intermediate tuple to ease individual argument access
00135         auto &&t = std::make_tuple(std::forward<Ts>(args)...);
00136         // Dispatch individual set_arg() for each argument
00137         auto just_to_evaluate = {
00138             0 /*< At least 1 element to deal with empty set_args() *//,
00139             ( set_arg(Is, std::forward<Ts>(std::get<Is>(t))), 0)...
00140         };
00141         // Remove the warning about unused variable
00142         static_cast<void>(just_to_evaluate);
00143     }
00144
00145 public:
00146
00147     /** Set all kernel args for an OpenCL kernel which is used through the
00148         SYCL/OpenCL interop interface
00149
00150         \todo Update the specification to add this function according to
00151             https://cvs.khronos.org/bugzilla/show\_bug.cgi?id=15978 proposal
00152     */
00153     template <typename... Ts>
00154     void set_args(Ts &&... args) {
00155         /* Construct a set of increasing argument index to be able to call
00156            the real set_arg */
00157         dispatch_set_arg(std::index_sequence_for<Ts...>{},
00158             std::forward<Ts>(args)...);
00159     }
00160 #endif
00161
00162
00163     /** Kernel invocation method of a kernel defined as a lambda or
00164         functor. If it is a lambda function or the functor type is globally
00165         visible there is no need for the developer to provide a kernel name type
00166         (typename KernelName) for it, as described in 3.5.3
00167
00168         SYCL single_task launches a computation without parallelism at
00169         launch time.
00170
00171         \param F specify the kernel to be launched as a single_task
00172
00173         \param KernelName is a class type that defines the name to be used for
00174             the underlying kernel
00175     */
00176     template <typename KernelName = std::nullptr_t>
00177     void single_task(std::function<void(void)> F) {
00178         task->schedule(detail::trace_kernel<KernelName>(F));
00179     }
00180
00181
00182     /** SYCL parallel_for launches a data parallel computation with
00183         parallelism specified at launch time by a range<>
00184
00185         Kernel invocation method of a kernel defined as a lambda or functor,
00186         for the specified range and given an id or item for indexing in the
00187         indexing space defined by range.
00188     */

```

```

00189     If it is a lambda function or the if the functor type is globally
00190     visible there is no need for the developer to provide a kernel name
00191     type (typename KernelName) for it, as described in detail in 3.5.3
00192
00193     \param global_size is the full size of the range<>
00194
00195     \param N dimensionality of the iteration space
00196
00197     \param f is the kernel functor to execute
00198
00199     \param KernelName is a class type that defines the name to be used
00200     for the underlying kernel
00201
00202     Unfortunately, to have implicit conversion to work on the range, the
00203     function can not be templated, so instantiate it for all the
00204     Dimensions
00205     */
00206 #define TRISYCL_parallel_for_functor_GLOBAL(N)
00207     template <typename KernelName = std::nullptr_t,
00208             typename ParallelForFunctor>
00209     void parallel_for(range<N> global_size,
00210                     ParallelForFunctor f) {
00211         task->schedule(detail::trace_kernel<KernelName>([=] {
00212             detail::parallel_for(global_size, f);
00213         }));
00214     }
00215
00216     TRISYCL_parallel_for_functor_GLOBAL(1)
00217     TRISYCL_parallel_for_functor_GLOBAL(2)
00218     TRISYCL_parallel_for_functor_GLOBAL(3)
00219
00220     /** Kernel invocation method of a kernel defined as a lambda or functor,
00221     for the specified range and offset and given an id or item for
00222     indexing in the indexing space defined by range
00223
00224     If it is a lambda function or the if the functor type is globally
00225     visible there is no need for the developer to provide a kernel name
00226     type (typename KernelName) for it, as described in detail in 3.5.3
00227
00228     \param global_size is the global size of the range<>
00229
00230
00231     \param offset is the offset to be add to the id<> during iteration
00232
00233     \param f is the kernel functor to execute
00234
00235     \param ParallelForFunctor is the kernel functor type
00236
00237     \param KernelName is a class type that defines the name to be used for
00238     the underlying kernel
00239
00240     Unfortunately, to have implicit conversion to work on the range, the
00241     function can not be templated, so instantiate it for all the
00242     dimensions
00243     */
00244 #define TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(N)
00245     template <typename KernelName = std::nullptr_t,
00246             typename ParallelForFunctor>
00247     void parallel_for(range<N> global_size,
00248                     id<N> offset,
00249                     ParallelForFunctor f) {
00250         task->schedule(detail::trace_kernel<KernelName>([=] {
00251             detail::parallel_for_global_offset(global_size,
00252                                             offset,
00253                                             f);
00254         }));
00255     }
00256
00257     TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(1)
00258     TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(2)
00259     TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(3)
00260
00261     /** Kernel invocation method of a kernel defined as a lambda or functor,
00262     for the specified nd_range and given an nd_item for indexing in the
00263     indexing space defined by the nd_range
00264
00265     If it is a lambda function or the if the functor type is globally
00266     visible there is no need for the developer to provide a kernel name
00267     type (typename KernelName) for it, as described in detail in 3.5.3
00268
00269     \param r defines the iteration space with the work-group layout and
00270     offset
00271
00272
00273     \param Dimensions dimensionality of the iteration space
00274
00275     \param f is the kernel functor to execute

```

```

00276
00277     \param ParallelForFunctor is the kernel functor type
00278
00279     \param KernelName is a class type that defines the name to be used for
00280     the underlying kernel
00281 */
00282 template <typename KernelName = std::nullptr_t,
00283           int Dimensions,
00284           typename ParallelForFunctor>
00285 void parallel_for(nd_range<Dimensions> r, ParallelForFunctor f) {
00286     task->schedule(detail::trace_kernel<KernelName>{[=] {
00287         detail::parallel_for(r, f);
00288     }});
00289 }
00290
00291
00292 /** Hierarchical kernel invocation method of a kernel defined as a
00293     lambda encoding the body of each work-group to launch
00294
00295     May contain multiple kernel built-in parallel_for_work_item
00296     functions representing the execution on each work-item.
00297
00298     Launch num_work_groups work-groups of runtime-defined
00299     size. Described in detail in 3.5.3.
00300
00301     \param r defines the iteration space with the work-group layout and
00302     offset
00303
00304     \param Dimensions dimensionality of the iteration space
00305
00306     \param f is the kernel functor to execute
00307
00308     \param ParallelForFunctor is the kernel functor type
00309
00310     \param KernelName is a class type that defines the name to be used for
00311     the underlying kernel
00312 */
00313 template <typename KernelName = std::nullptr_t,
00314           int Dimensions = 1,
00315           typename ParallelForFunctor>
00316 void parallel_for_work_group(nd_range<Dimensions> r,
00317                             ParallelForFunctor f) {
00318     task->schedule(detail::trace_kernel<KernelName>{[=] {
00319         detail::parallel_for_workgroup(r, f); }});
00320 }
00321
00322
00323 /** Hierarchical kernel invocation method of a kernel defined as a
00324     lambda encoding the body of each work-group to launch
00325
00326     May contain multiple kernel built-in parallel_for_work_item
00327     functions representing the execution on each work-item.
00328
00329     Launch num_work_groups work-groups of runtime-defined
00330     size. Described in detail in 3.5.3.
00331
00332     \param r defines the iteration space with the work-group layout and
00333     offset
00334
00335     \param Dimensions dimensionality of the iteration space
00336
00337     \param f is the kernel functor to execute
00338
00339     \param ParallelForFunctor is the kernel functor type
00340
00341     \param KernelName is a class type that defines the name to be used for
00342     the underlying kernel
00343 */
00344 template <typename KernelName = std::nullptr_t,
00345           int Dimensions = 1,
00346           typename ParallelForFunctor>
00347 void parallel_for_work_group(range<Dimensions> r1,
00348                             range<Dimensions> r2,
00349                             ParallelForFunctor f) {
00350     parallel_for_work_group(nd_range<Dimensions> { r1, r2 }, f);
00351 }
00352
00353 /** Kernel invocation method of a kernel defined as pointer to a kernel
00354     object, described in detail in 3.5.3
00355
00356     \todo Add in the spec a version taking a kernel and a functor,
00357     to have host fall-back
00358 */
00359 void single_task(kernel sycl_kernel) {
00360     /* For now just use the usual host task system to schedule      \
00361        manually the OpenCL kernels instead of using OpenCL event-based \

```

```

00362         scheduling
00363
00364         \todo Move the tracing inside the kernel implementation
00365
00366         \todo Simplify this 2 step ugly interface
00367     */
00368     task->set_kernel(sycl_kernel.implementation);
00369     task->schedule(detail::trace_kernel<kernel>([=, t = task] {
00370         sycl_kernel.implementation->single_task(t, t->get_queue());
00371     }));
00372 }
00373
00374
00375 /** Kernel invocation method of a kernel defined as a kernel object,
00376     for the specified range and given an id or item for indexing in
00377     the indexing space defined by range, described in detail in
00378     5.4.
00379
00380     \todo Add in the spec a version taking a kernel and a functor,
00381     to have host fall-back
00382 */
00383 #define TRISYCL_ParallelForKernel_RANGE(N)
00384 void parallel_for(range<N> num_work_items,
00385                 kernel sycl_kernel) {
00386     /* For now just use the usual host task system to schedule
00387        manually the OpenCL kernels instead of using OpenCL event-based
00388        scheduling
00389
00390        \todo Move the tracing inside the kernel implementation
00391
00392        \todo Simplify this 2 step ugly interface
00393    */
00394     task->set_kernel(sycl_kernel.implementation);
00395     /* Use an intermediate variable to capture task by copy because
00396        otherwise "this" is captured by reference and havoc with task
00397        just accessing the dead "this". Nasty bug to find... */
00398     task->schedule(detail::trace_kernel<kernel>([=, t = task] {
00399         sycl_kernel.implementation->parallel_for(t, t->get_queue(),
00400         num_work_items); })); \
00401 }
00402
00403
00404 /* Do not use a template parameter since otherwise the parallel_for
00405     functor is selected instead of this one
00406
00407     \todo Clean this
00408 */
00409 TRISYCL_ParallelForKernel_RANGE(1)
00410 TRISYCL_ParallelForKernel_RANGE(2)
00411 TRISYCL_ParallelForKernel_RANGE(3)
00412 #undef TRISYCL_ParallelForKernel_RANGE
00413
00414 /** Kernel invocation method of a kernel defined as pointer to a kernel
00415     object, for the specified nd_range and given an nd_item for indexing
00416     in the indexing space defined by the nd_range, described in detail
00417     in 3.5.3
00418
00419     \todo Add in the spec a version taking a kernel and a functor,
00420     to have host fall-back
00421
00422     \todo To be implemented
00423 */
00424 template <int Dimensions = 1>
00425 void parallel_for(nd_range<Dimensions>, kernel syclKernel) {
00426     detail::unimplemented();
00427 }
00428
00429 };
00430
00431 namespace detail {
00432
00433 /** Register a buffer as used by a task
00434
00435     This is a proxy function to avoid complicated type recursion.
00436 */
00437 static std::shared_ptr<detail::task>
00438 add_buffer_to_task(handler *command_group_handler,
00439                 std::shared_ptr<detail::buffer_base> b,
00440                 bool is_write_mode) {
00441     command_group_handler->task->add_buffer(b, is_write_mode);
00442     return command_group_handler->task;
00443 }
00444
00445 }
00446
00447 /// @} End the execution Doxygen group
00448

```

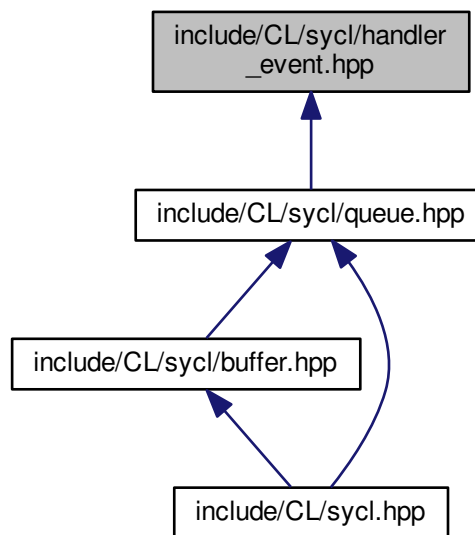
```

00449 }
00450 }
00451
00452 /*
00453  # Some Emacs stuff:
00454  ### Local Variables:
00455  ### ispell-local-dictionary: "american"
00456  ### eval: (flyspell-prog-mode)
00457  ### End:
00458 */
00459
00460 #endif // TRISYCL_SYCL_HANDLER_HPP

```

## 11.87 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.*

## 11.88 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

```

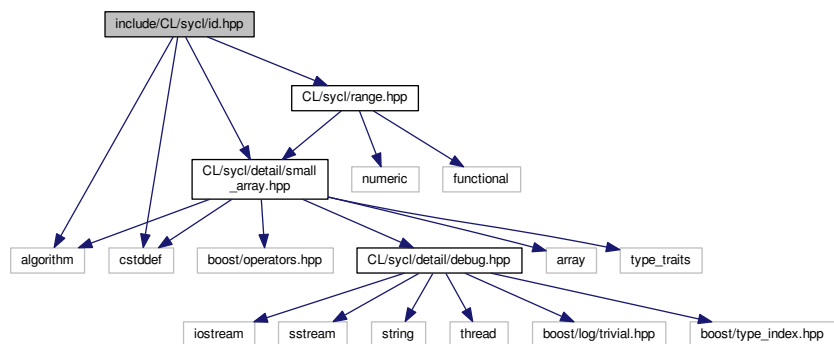
## 11.89 include/CL/sycl/id.hpp File Reference

```

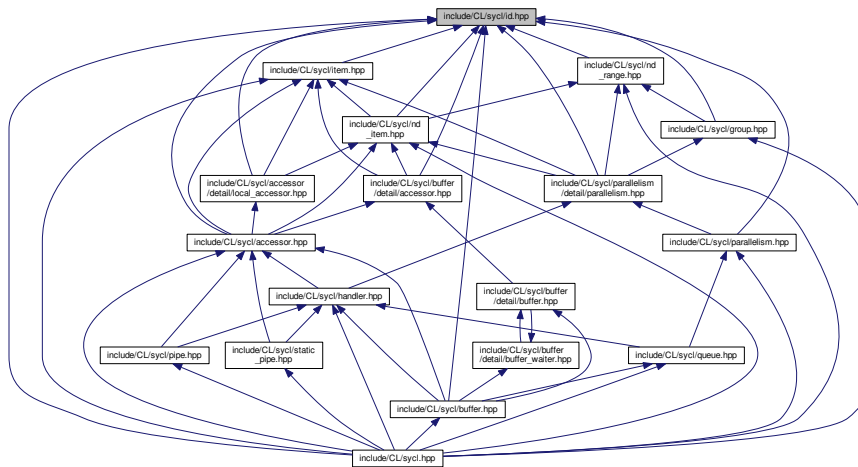
#include <algorithm>
#include <cstdint>
#include "CL/sycl/detail/small_array.hpp"
#include "CL/sycl/range.hpp"

```

Include dependency graph for id.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [cl::sycl::item< Dimensions >](#)

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< Dimensions >](#)

Define a multi-dimensional index, used for example to locate a work item. [More...](#)

## Namespaces

- [cl](#)

The vector type to be used as SYCL vector.

- [cl::sycl](#)

## Functions

- auto [cl::sycl::make\\_id](#) (id< 1 > i)

Implement a `make_id` to construct an `id<>` of the right dimension with implicit conversion from an initializer list for example.

- auto [cl::sycl::make\\_id](#) (id< 2 > i)

- auto [cl::sycl::make\\_id](#) (id< 3 > i)

- template<typename... BasicType>

auto [cl::sycl::make\\_id](#) (BasicType... Args)

Construct an `id<>` from a function call with arguments, like `make_id(1, 2, 3)`



## 11.90 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 <int Dimensions> 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 <int Dimensions = 1>
00031 class id : public detail::small_array_123<
00032     std::size_t,
00033     id<Dimensions>,
00034     Dimensions > {
00035
00036 public:
00037
00038     // Inherit from all the constructors
00039     using detail::small_array_123<std::size_t,
00040         id<Dimensions>,
00041         Dimensions>::small_array_123;
00042
00043
00044     /// Construct an id from the dimensions of a range
00045     id(const range<Dimensions> &range_size)
00046         /** Use the fact we have a constructor of a small_array from a another
00047         kind of small_array
00048         */
00049         : detail::small_array_123<std::size_t, id<Dimensions>, Dimensions>
00050         { range_size }
00051     {}
00052
00053
00054     /// Construct an id from an item global_id
00055     id(const item<Dimensions> &rhs)
00056         : detail::small_array_123<std::size_t, id<Dimensions>
00057         , Dimensions>
00058         { rhs.get() }
00059     {}
00060
00061     /// Keep other constructors
00062     id() = default;
00063 };
00064
00065
00066 /** Implement a make_id to construct an id<> of the right dimension with
00067     implicit conversion from an initializer list for example.
00068
00069     Cannot use a template on the number of dimensions because the implicit
00070     conversion would not be tried. */
00071 inline auto make_id(id<1> i) { return i; }
00072 inline auto make_id(id<2> i) { return i; }
00073 inline auto make_id(id<3> i) { return i; }
00074
00075
00076 /** Construct an id<> from a function call with arguments, like
00077     make_id(1, 2, 3) */
00078 template<typename... BasicType>
00079 auto make_id(BasicType... Args) {
00080     // Call constructor directly to allow narrowing
00081     return id<sizeof...(Args)>(Args...);
00082 }
00083

```

```

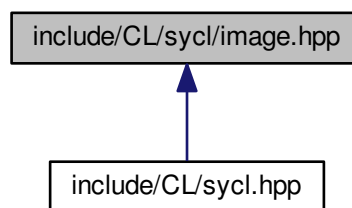
00084 /// @} End the parallelism Doxygen group
00085
00086 }
00087 }
00088
00089 /*
00090  # Some Emacs stuff:
00091  ### Local Variables:
00092  ### ispell-local-dictionary: "american"
00093  ### eval: (flyspell-prog-mode)
00094  ### End:
00095  */
00096
00097 #endif // TRISYCL_SYCL_ID_HPP

```

## 11.91 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](#)

#### 11.91.1 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.

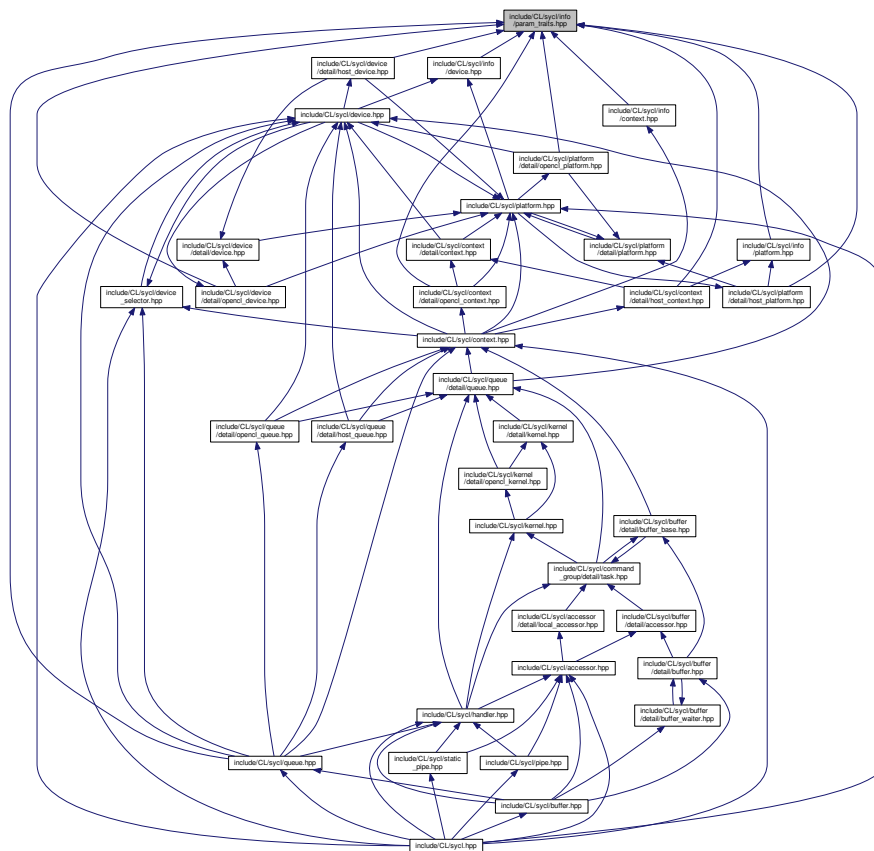
Definition in file [image.hpp](#).

## 11.92 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 <int 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
```

## 11.93 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  $(T, \text{value})$  to  $T'$  to express the return type value of an OpenCL function of kind  $(T, \text{value})$

### 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`.

## 11.93.1 Macro Definition Documentation

### 11.93.1.1 TRISYCL\_INFO\_PARAM\_TRAITS

```
#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](#).

### 11.93.1.2 TRISYCL\_INFO\_PARAM\_TRAITS\_ANY\_T

```
#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](#).

## 11.94 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

```

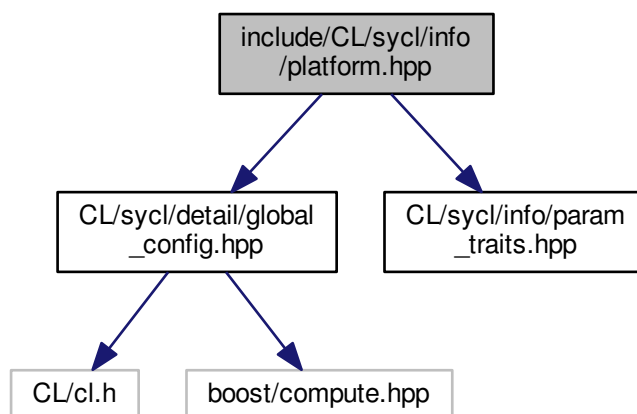
## 11.95 include/CL/sycl/info/platform.hpp File Reference

```

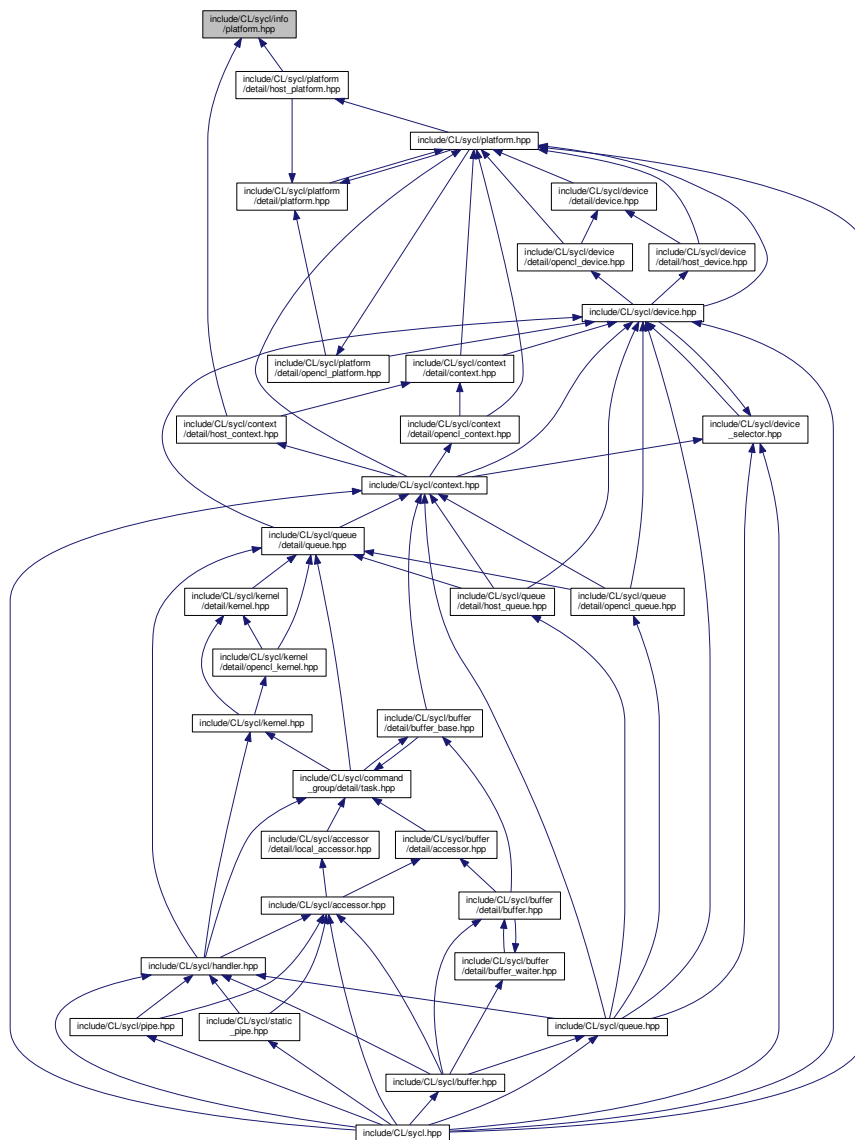
#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/info/param_traits.hpp"

```

Include dependency graph for platform.hpp:



This graph shows which files directly or indirectly include this file:



## Namespaces

- `cl`  
The vector type to be used as SYCL vector.
- `cl::sycl`
- `cl::sycl::info`

## Enumerations

- enum `cl::sycl::info::platform` : unsigned int {  
`cl::sycl::info::platform::TRISYCL_SKIP_OPENCL`  $\neq$  `CL_PLATFORM_PROFILE`), `cl::sycl::info::platform::TRISYCL_SKIP_OPENCL`  $\neq$  `CL_PLATFORM_VERSION`), `cl::sycl::info::platform::TRISYCL_SKIP_OPENCL`  $\neq$  `CL_PLATFORM_NAME`), `cl::sycl::info::platform::TRISYCL_SKIP_OPENCL`  $\neq$  `CL_PLATFORM_VENDOR`),  
`cl::sycl::info::platform::TRISYCL_SKIP_OPENCL`  $\neq$  `CL_PLATFORM_EXTENSIONS`) }  
Platform information descriptors.



## 11.96 platform.hpp

```

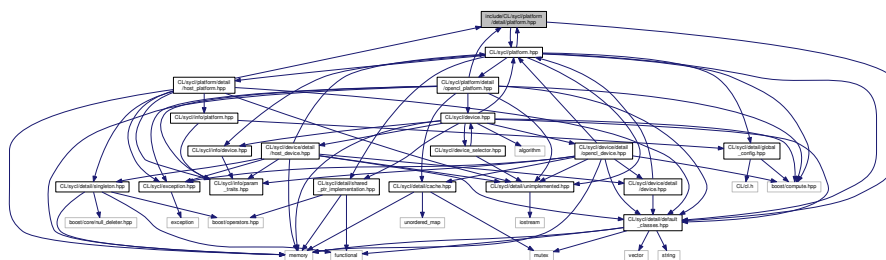
00001 #ifndef TRISYCL_SYCL_INFO_PLATFORM_HPP
00002 #define TRISYCL_SYCL_INFO_PLATFORM_HPP
00003
00004 /** \file The OpenCL SYCL platform information parameters
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. 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

```

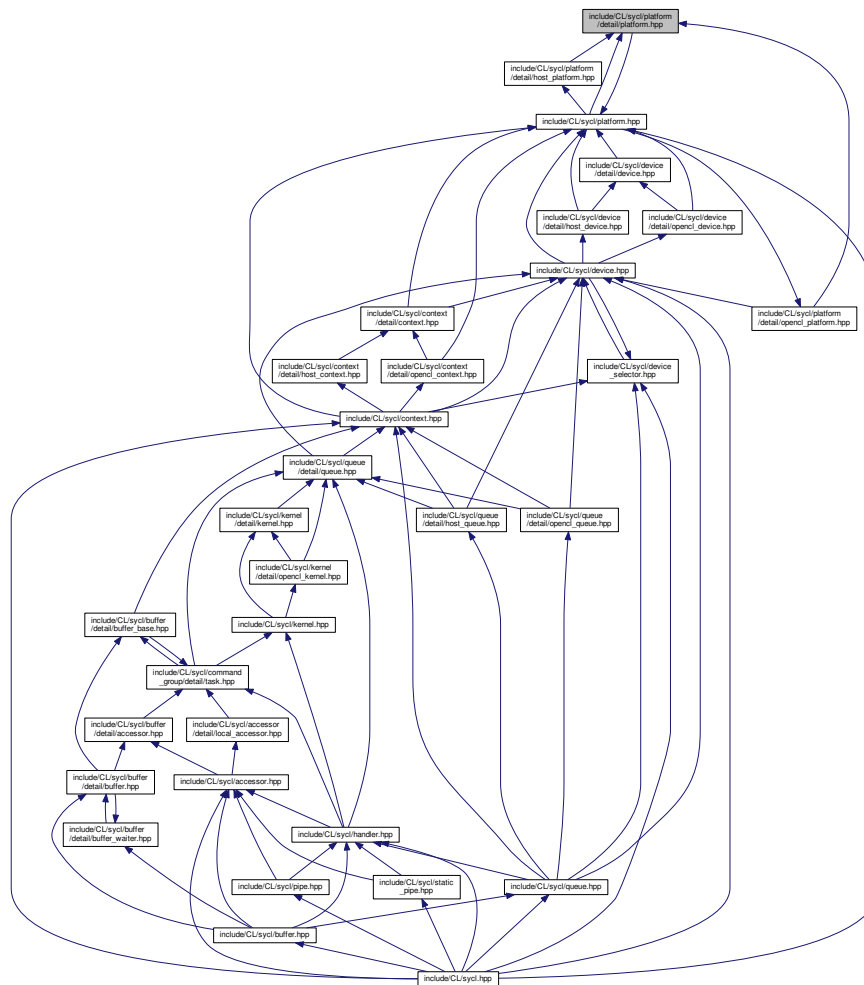
```
00084 TRISYCL_INFO_PARAM_TRAITS(info::platform::host_timer_resolution,
00085                             unsigned long int)
00086 #endif
00087 #endif
00088 }
00089 }
00090 }
00091
00092 /*
00093  # Some Emacs stuff:
00094  ### Local Variables:
00095  ### ispell-local-dictionary: "american"
00096  ### eval: (flyspell-prog-mode)
00097  ### End:
00098 */
00099
00100 #endif // TRISYCL_SYCL_INFO_PLATFORM_HPP
```

## 11.97 include/CL/sycl/platform/detail/platform.hpp File Reference

```
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/platform.hpp"
Include dependency graph for platform.hpp:
```



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](#)

## 11.98 platform.hpp

```
00001 #ifndef TRISYCL_SYCL_PLATFORM_DETAIL_PLATFORM_HPP
00002 #define TRISYCL_SYCL_PLATFORM_DETAIL_PLATFORM_HPP
00003
```

```

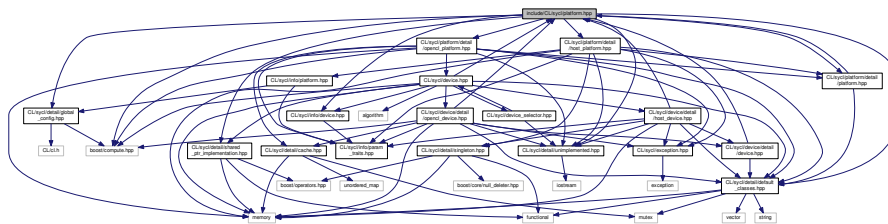
00004 /** \file The OpenCL SYCL abstract platform
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include "CL/sycl/detail/default_classes.hpp"
00013
00014 #include "CL/sycl/platform.hpp"
00015
00016 namespace cl {
00017 namespace sycl {
00018
00019     class device;
00020
00021     namespace detail {
00022
00023         /** \addtogroup execution Platforms, contexts, devices and queues
00024             @{
00025         */
00026
00027         /// An abstract class representing various models of SYCL platforms
00028         class platform {
00029         public:
00030
00031             #ifndef TRISYCL_OPENCL
00032             /// Return the cl_platform_id of the underlying OpenCL platform
00033             virtual cl_platform_id get() const = 0;
00034
00035             /// Return the underlying Boost.Compute platform, if any
00036             virtual const boost::compute::platform &
00037             get_boost_compute() const = 0;
00038             #endif
00039
00040             /// Return true if the platform is a SYCL host platform
00041             virtual bool is_host() const = 0;
00042
00043             /// Query the platform for OpenCL string info::platform info
00044             virtual string_class get_info_string(info::platform param) const
00045             = 0;
00046
00047             /// Specify whether a specific extension is supported on the platform.
00048             virtual bool has_extension(const string_class &extension) const = 0;
00049
00050             /** Get all the available devices for this platform
00051
00052                 \param[in] device_type is the device type to filter the selection
00053                 or \c info::device_type::all by default to return all the
00054                 devices
00055
00056                 \return the device list
00057             */
00058             virtual vector_class<device>
00059             get_devices(info::device_type device_type) const = 0;
00060
00061             // Virtual to call the real destructor
00062             virtual ~platform() {}
00063         };
00064     };
00065
00066     /// @} to end the execution Doxygen group
00067 }
00068
00069 }
00070
00071 }
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_PLATFORM_DETAIL_PLATFORM_HPP

```

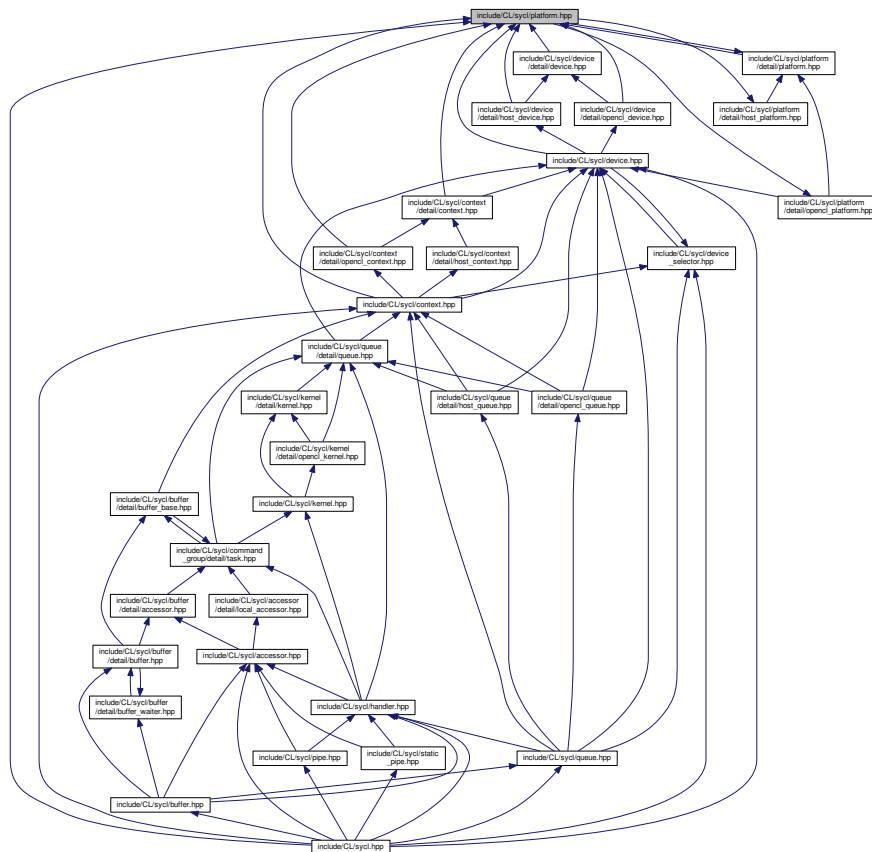
## 11.99 include/CL/sycl/platform.hpp File Reference

```
#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/info/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 dependency graph for platform.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [cl::sycl::platform](#)  
Abstract the OpenCL platform. [More...](#)
- struct [std::hash< cl::sycl::platform >](#)

## Namespaces

- [cl](#)  
The vector type to be used as SYCL vector.
- [cl::sycl](#)
- [std](#)

## 11.100 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/info/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
00028 namespace cl {
00029 namespace sycl {
00030
00031     class device_selector;
00032     class device;
00033
00034     /** \addtogroup execution Platforms, contexts, devices and queues
00035     @{
00036     */
00037
00038     /** Abstract the OpenCL platform
00039
00040         \todo triSYCL Implementation
00041     */
00042     class platform
00043     {
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     // Allows the comparison operation to access the implementation
00049     friend shared_ptr_implementation;
00050
00051     public:
00052
00053     // Make the implementation member directly accessible in this class
00054     using shared_ptr_implementation::implementation;
00055
00056     /** Default constructor for platform which is the host platform
00057
00058         Returns errors via the SYCL exception class.
00059     */

```

```

00059     platform() :
00060     {
00061         shared_ptr_implementation {
00062             detail::host_platform::instance() } {}
00063     }
00064 #ifndef TRISYCL_OPENCL
00065     /** Construct a platform class instance using cl_platform_id of the
00066         OpenCL device
00067
00068         Return synchronous errors via the SYCL exception class.
00069
00070         Retain a reference to the OpenCL platform.
00071     */
00072     platform(cl_platform_id platform_id)
00073     : platform { boost::compute::platform { platform_id } } {}
00074
00075     /** Construct a platform class instance using a boost::compute::platform
00076
00077         This is a triSYCL extension for boost::compute interoperability.
00078
00079         Return synchronous errors via the SYCL exception class.
00080     */
00081     platform(const boost::compute::platform &p)
00082     : shared_ptr_implementation {
00083         detail::opencl_platform::instance(p) } {}
00084 #endif
00085
00086     /** Construct a platform object from the device selected by a device
00087         selector of the user's choice
00088
00089         Returns errors via the SYCL exception class.
00090     */
00091     explicit platform(const device_selector &dev_selector) {
00092         detail::unimplemented();
00093     }
00094
00095 #ifndef TRISYCL_OPENCL
00096     /** Returns the cl_platform_id of the underlying OpenCL platform
00097
00098         If the platform is not a valid OpenCL platform, for example if it is
00099         the SYCL host, an exception is thrown
00100
00101         \todo Define a SYCL exception for this
00102     */
00103     cl_platform_id get() const {
00104         return implementation->get();
00105     }
00106
00107     /** Return the underlying Boost.Compute platform if it is an
00108         OpenCL platform
00109
00110         This is a triSYCL extension
00111     */
00112     const boost::compute::platform get_boost_compute() const {
00113         return implementation->get_boost_compute();
00114     }
00115 #endif
00116
00117     /// Get the list of all the platforms available to the application
00118     static vector_class<platform> get_platforms() {
00119         // Start with the default platform
00120         vector_class<platform> platforms { {} };
00121
00122 #ifndef TRISYCL_OPENCL
00123         // Then add all the OpenCL platforms
00124         for (const auto &d : boost::compute::system::platforms())
00125             platforms.emplace_back(d);
00126 #endif
00127
00128     return platforms;
00129 }
00130
00131 /** Get the OpenCL information about the requested parameter
00132
00133     \todo Add to the specification
00134 */
00135 template <typename ReturnT>
00136 ReturnT get_info(info::platform param) const {
00137     // Only strings are needed here
00138     return implementation->get_info_string(param);
00139 }

```

```

00144
00145
00146 /// Get the OpenCL information about the requested template parameter
00147 template <info::platform Param>
00148 typename info::param_traits<info::platform, Param>::type
00149 get_info() const {
00150     /* Forward to the implementation without using template parameter
00151        but with a parameter instead, since it is incompatible with
00152        virtual function and because fortunately only strings are
00153        needed here */
00154     return get_info<typename info::param_traits<
info::platform,
00155                                     Param>::type>(Param);
00156 }
00157
00158
00159 /// Test if an extension is available on the platform
00160 bool has_extension(const string_class &extension) const {
00161     return implementation->has_extension(extension);
00162 }
00163
00164
00165 /// Test if this platform is a host platform
00166 bool is_host() const {
00167     return implementation->is_host();
00168 }
00169
00170
00171 /** Get all the available devices for this platform
00172
00173     \param[in] device_type is the device type to filter the selection
00174     or \c info::device_type::all by default to return all the
00175     devices
00176
00177     \return the device list
00178 */
00179 vector_class<device>
00180 get_devices(info::device_type device_type =
info::device_type::all) const {
00181     return implementation->get_devices(device_type);
00182 }
00183
00184 };
00185
00186 /// @} to end the execution Doxygen group
00187
00188 }
00189 }
00190
00191
00192 /* Inject a custom specialization of std::hash to have the buffer
00193    usable into an unordered associative container
00194
00195    \todo Add this to the spec
00196 */
00197 namespace std {
00198
00199 template <> struct hash<cl::sycl::platform> {
00200
00201     auto operator()(const cl::sycl::platform &p) const {
00202         // Forward the hashing to the implementation
00203         return p.hash();
00204     }
00205
00206 };
00207
00208 }
00209
00210 /*
00211     # Some Emacs stuff:
00212     ### Local Variables:
00213     ### ispell-local-dictionary: "american"
00214     ### eval: (flyspell-prog-mode)
00215     ### End:
00216 */
00217
00218 #endif // TRISYCL_SYCL_PLATFORM_HPP

```

## 11.101 include/CL/sycl/item.hpp File Reference

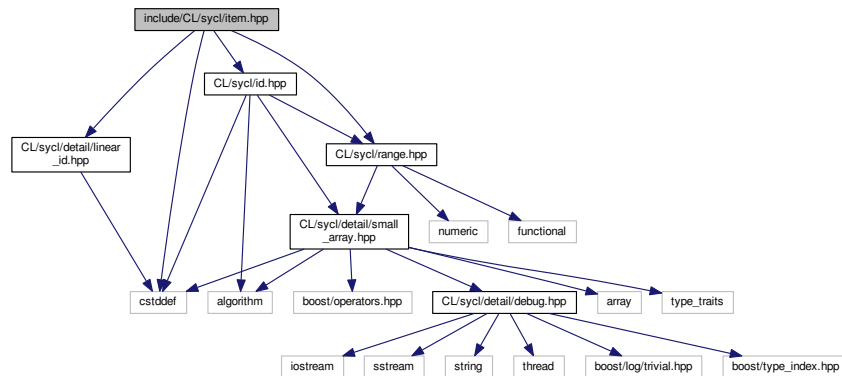
```

#include <cstdint>
#include "CL/sycl/detail/linear_id.hpp"

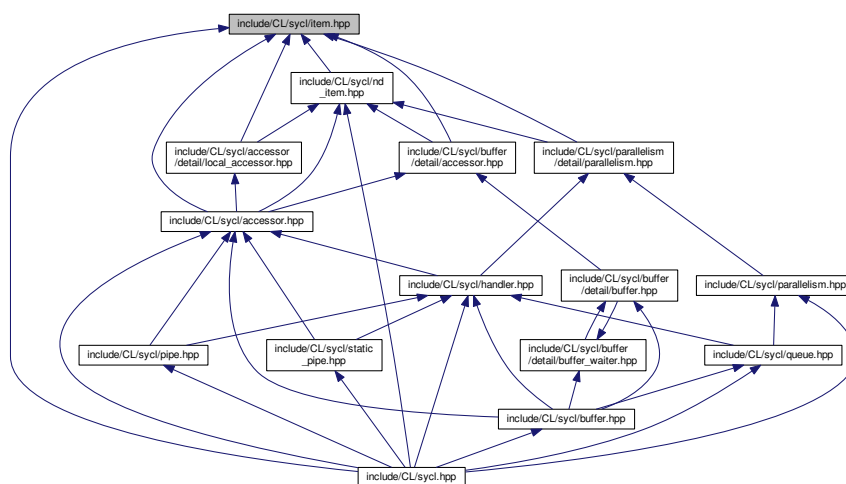
```



```
#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< Dimensions >](#)

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](#)

## 11.102 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 <int Dimensions = 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 = Dimensions;
00035
00036 private:
00037     range<Dimensions> global_range;
00038     id<Dimensions> global_index;
00039     id<Dimensions> 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<Dimensions> global_size,
00048          id<Dimensions> global_index,
00049          id<Dimensions> 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<Dimensions> 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<Dimensions> 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<Dimensions> 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(),
get_offset());
00106     }
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<Dimensions> 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

```

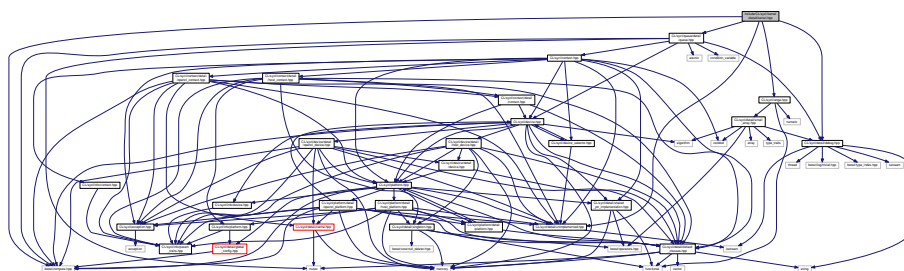
## 11.103 include/CL/sycl/kernel/detail/kernel.hpp File Reference

```

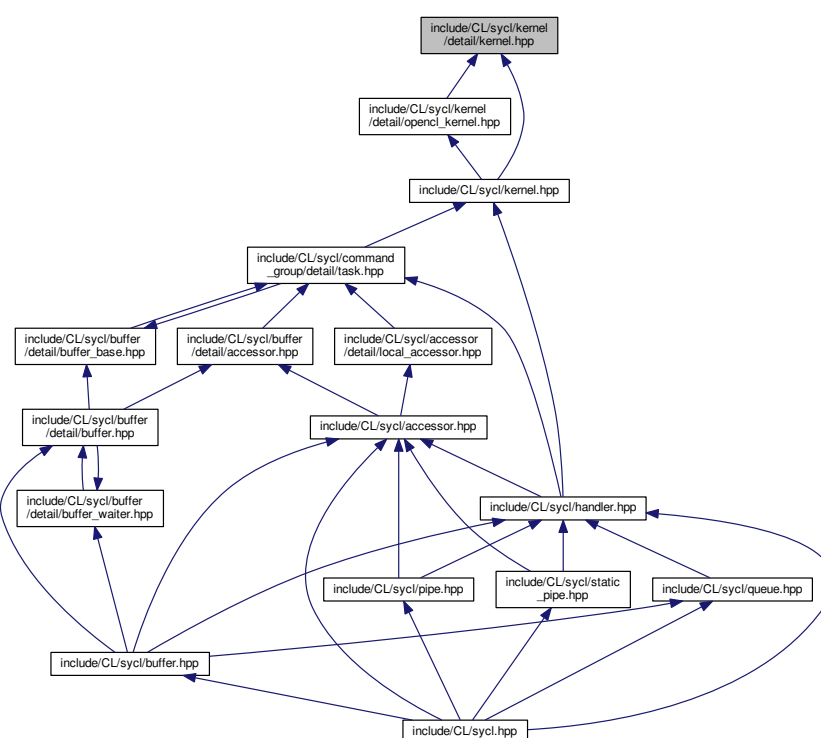
#include <boost/compute.hpp>
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/queue/detail/queue.hpp"
#include "CL/sycl/range.hpp"

```

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<>*

### 11.103.1 Macro Definition Documentation

#### 11.103.1.1 TRISYCL\_ParallelForKernel\_RANGE

```
#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 63 of file [kernel.hpp](#).

## 11.104 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 single task of the kernel
00053     virtual void single_task(std::shared_ptr<detail::task> task,
00054                             std::shared_ptr<detail::queue> q) = 0;
00055
00056
00057     /** Launch a kernel with a range<>
00058
00059         Do not use a template since it does not work with virtual functions
00060
00061         \todo Think to a cleaner solution
00062     */
00063 #define TRISYCL_ParallelForKernel_RANGE(N) \
00064     virtual void parallel_for(std::shared_ptr<detail::task> task, std::shared_ptr<detail::queue> q, \
00065                             const range<N> &num_work_items) = 0;
00066
00067     TRISYCL_ParallelForKernel_RANGE(1)
00068     TRISYCL_ParallelForKernel_RANGE(2)
00069     TRISYCL_ParallelForKernel_RANGE(3)
00070 #undef TRISYCL_ParallelForKernel_RANGE
00071
00072
00073     /// Return the context that this kernel is defined for
00074     //virtual context get_context() const;
00075
00076     /// Return the program that this kernel is part of
00077     //virtual program get_program() const;
00078
00079     // Virtual to call the real destructor
00080     virtual ~kernel() {}
00081
00082 };
00083
00084 /// @} End the execution 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_DETAIL_KERNEL_KERNEL_HPP

```

## 11.105 include/CL/sycl/kernel.hpp File Reference

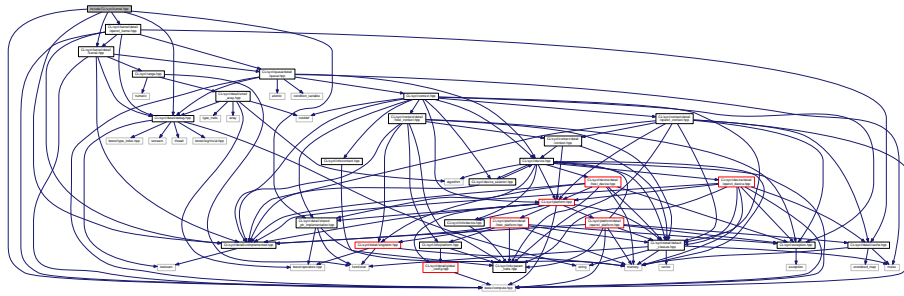
```

#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"

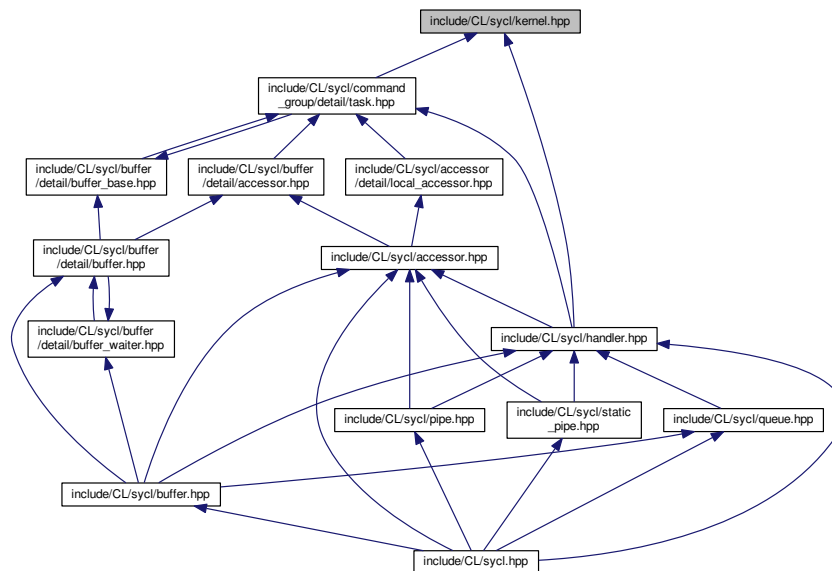
```

```
#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`

## 11.106 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/opengl_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 = typename
00045     kernel::shared_ptr_implementation;
00046
00047     // The handler class uses the implementation
00048     friend class handler;
00049
00050     // Allows the comparison operation to access the implementation
00051     friend implementation_t;
00052
00053     public:
00054
00055     // Make the implementation member directly accessible in this class
00056     using implementation_t::implementation;
00057
00058     /** The default object is not valid because there is no program or
00059     \code cl_kernel \endcode associated with it */
00060     kernel() = delete;
00061
00062 #ifdef TRISYCL_OPENCL
00063     /** Constructor for SYCL kernel class given an OpenGL kernel object
00064     with set arguments, valid for enqueueing
00065
00066     Retains a reference to the \p cl_kernel object. The Caller
00067     should release the passed cl_kernel object when it is no longer
00068     needed.
00069     */
00070     kernel(cl_kernel k) : kernel { boost::compute::kernel { k } } {}
00071
00072     /** Construct a kernel class instance using a boost::compute::kernel
00073
00074     This is a triSYCL extension for boost::compute interoperation.
00075
00076     Return synchronous errors via the SYCL exception class.
00077     */
00078     kernel(const boost::compute::kernel &k)
00079         : implementation_t { detail::opengl_kernel::instance(k) } {}
00080
00081
00082     /** Return the OpenGL kernel object for this kernel

```



```

00083
00084     Retains a reference to the returned cl_kernel object. Caller
00085     should release it when finished.
00086     */
00087     cl_kernel get() const {
00088         return implementation->get();
00089     }
00090 #endif
00091
00092
00093 #if 0
00094     /// Return the context that this kernel is defined for
00095     ///context get_context() const;
00096
00097     /// Return the program that this kernel is part of
00098     ///program get_program() const;
00099
00100     /** Query information from the kernel object using the
00101         info::kernel_info descriptor.
00102     */
00103     template <info::kernel param>
00104     typename info::param_traits<info::kernel, param>::type
00105         get_info() const {
00106         detail::unimplemented();
00107     }
00108 #endif
00109
00110 };
00111
00112 /// @} End the execution Doxygen group
00113 }
00114 }
00115 }
00116
00117
00118 /* Inject a custom specialization of std::hash to have the buffer
00119     usable into an unordered associative container
00120
00121     \todo Add this to the spec
00122 */
00123 namespace std {
00124
00125 template <> struct hash<cl::sycl::kernel> {
00126
00127     auto operator()(const cl::sycl::kernel &k) const {
00128         // Forward the hashing to the implementation
00129         return k.hash();
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_KERNEL_HPP

```

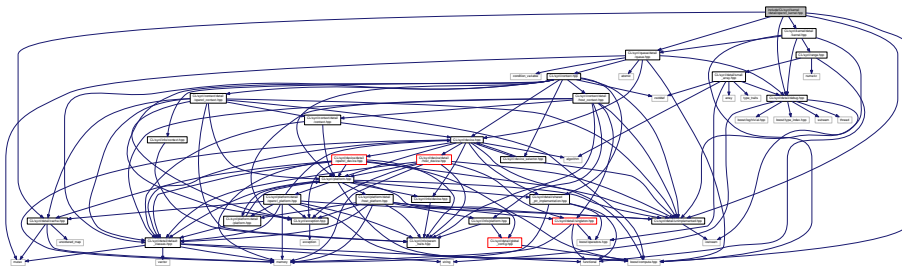
## 11.107 include/CL/sycl/kernel/detail/openccl\_kernel.hpp File Reference

```

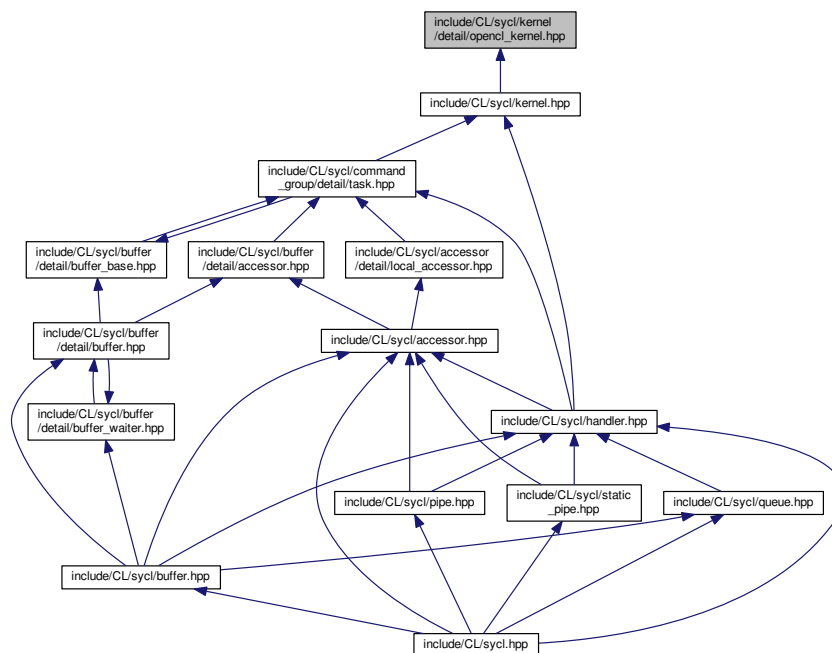
#include <boost/compute.hpp>
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/kernel/detail/kernel.hpp"
#include "CL/sycl/queue/detail/queue.hpp"

```

Include dependency graph for `openccl_kernel.hpp`:



This graph shows which files directly or indirectly include this file:



## Classes

- class `cl::sycl::detail::openccl_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<>*

### 11.107.1 Macro Definition Documentation

#### 11.107.1.1 TRISYCL\_ParallelForKernel\_RANGE

```
#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

**Todo** Remove either task or q

Definition at line 108 of file [opencil\\_kernel.hpp](#).

## 11.108 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
00024 namespace cl {
00025 namespace sycl {
00026 namespace detail {
00027
00028     /// An abstraction of the OpenCL kernel
00029     class opencil_kernel : public detail::kernel,
00030                           detail::debug<opencil_kernel> {
00031
00032     /// Use the Boost Compute abstraction of the OpenCL kernel
00033     boost::compute::kernel k;
00034
00035     /** A cache to always return the same alive kernel for a given
00036         OpenCL kernel
00037
00038         C++11 guaranties the static construction is thread-safe
00039     */
00040     static detail::cache<cl_kernel, detail::opencil_kernel>
00041     cache;
00042
00043     opencil_kernel(const boost::compute::kernel &k) : k { k } {}
00044
00045     public:
00046
00047     /// Get a singleton instance of the opencil_device
00048     static std::shared_ptr<opencil_kernel>
00049     instance(const boost::compute::kernel &k) {
00050         return cache.get_or_register(k.get(),
00051                                     [&] { return new opencil_kernel { k }; });
00052     }
00053
00054     /** Return the underlying OpenCL object
00055
00056         \todo Improve the spec to deprecate C OpenCL host API and move
00057         to C++ instead to avoid this ugly ownership management
00058     */
00059     cl_kernel get() const override {
00060         /// \todo Test error and throw. Externalize this feature in Boost.Compute?
00061         clRetainKernel(k);
00062         return k.get();
00063     }
00064
00065     /** Return the Boost.Compute OpenCL kernel object for this kernel
00066
00067         This is an extension.
00068     */
00069     boost::compute::kernel get_boost_compute() const override {
00070         return k;
00071     }
00072
00073     //context get_context() const override
00074
00075     //program get_program() const override
00076
00077
00078 #if 0
00079     template <info::kernel param>
00080     typename info::param_traits<info::kernel, param>::type
00081     get_info() const {
00082         detail::unimplemented();
00083     }
00084 #endif

```

```

00084 #endif
00085
00086
00087 /** Launch a single task of the OpenCL kernel
00088
00089     \todo Remove either task or q
00090 */
00091 void single_task(std::shared_ptr<detail::task> task,
00092                 std::shared_ptr<detail::queue> q) override {
00093     q->get_boost_compute().enqueue_task(k);
00094     /* For now use a crude synchronization mechanism to map directly a
00095        host task to an accelerator task */
00096     q->get_boost_compute().finish();
00097 }
00098
00099
00100 /** Launch an OpenCL kernel with a range<>
00101
00102     Do not use a template since it does not work with virtual functions
00103
00104     \todo Think to a cleaner solution
00105
00106     \todo Remove either task or q
00107 */
00108 #define TRISYCL_ParallelForKernel_RANGE(N)
00109 void parallel_for(std::shared_ptr<detail::task> task,
00110                  std::shared_ptr<detail::queue> q,
00111                  const range<N> &num_work_items) override {
00112     static_assert(sizeof(range<N>::value_type) == sizeof(size_t),
00113                   "num_work_items::value_type compatible with "
00114                   "Boost.Compute");
00115     q->get_boost_compute().enqueue_nd_range_kernel
00116         (k,
00117          static_cast<size_t>(N),
00118          NULL,
00119          static_cast<const size_t *>(num_work_items.data()),
00120          NULL);
00121     /* For now use a crude synchronization mechanism to map directly a
00122        host task to an accelerator task */
00123     q->get_boost_compute().finish();
00124 };
00125
00126 TRISYCL_ParallelForKernel_RANGE(1)
00127 TRISYCL_ParallelForKernel_RANGE(2)
00128 TRISYCL_ParallelForKernel_RANGE(3)
00129 #undef TRISYCL_ParallelForKernel_RANGE
00130
00131
00132 /// Unregister from the cache on destruction
00133 ~opencil_kernel() override {
00134     cache.remove(k.get());
00135 }
00136
00137 };
00138
00139 /* Allocate the cache here but since this is a pure-header library,
00140    use a weak symbol so that only one remains when SYCL headers are
00141    used in different compilation units of a program
00142 */
00143 TRISYCL_WEAK_ATTRIB_PREFIX
00144 detail::cache<cl_kernel, detail::opencil_kernel>
00145 opencil_kernel::cache
00146 TRISYCL_WEAK_ATTRIB_SUFFIX;
00147 }
00148 }
00149 }
00150
00151 /*
00152     # Some Emacs stuff:
00153     ### Local Variables:
00154     ### ispell-local-dictionary: "american"
00155     ### eval: (flyspell-prog-mode)
00156     ### End:
00157 */
00158
00159 #endif // TRISYCL_SYCL_KERNEL_DETAIL_OPENCL_KERNEL_HPP

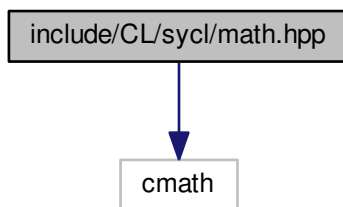
```

## 11.109 include/CL/sycl/math.hpp File Reference

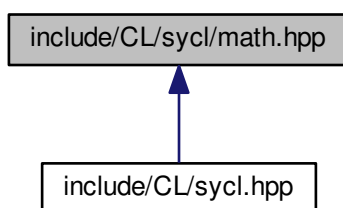
Implement a wrapper around OpenCL math operations Joan.Thibault AT ens-rennes POINT fr This file is distributed under the University of Illinois Open Source License.

```
#include <cmath>
```

Include dependency graph for math.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](#)

## Macros

- [#define TRISYCL\\_MATH\\_WRAP\(FUN\)](#)
- [#define TRISYCL\\_MATH\\_WRAP2\(FUN\)](#)
- [#define TRISYCL\\_MATH\\_WRAP2s\(FUN\)](#)
- [#define TRISYCL\\_MATH\\_WRAP3\(FUN\)](#)
- [#define TRISYCL\\_MATH\\_WRAP3s\(FUN\)](#)
- [#define TRISYCL\\_MATH\\_WRAP3ss\(FUN\)](#)

## Functions

- [cl::sycl::TRISYCL\\_MATH\\_WRAP](#) (abs) [TRISYCL\\_MATH\\_WRAP\(atan\)](#) [TRISYCL\\_MATH\\_WRAP2s\(fmax\)](#) [TRISYCL\\_MATH\\_WRAP2s\(fmin\)](#) [TRISYCL\\_MATH\\_WRAP2s\(frexp\)](#) [template< typename T > T max\(T x](#)
- [template<typename T >](#)  
[T cl::sycl::min](#) (T x, T y, T z)
- [cl::sycl::TRISYCL\\_MATH\\_WRAP2s](#) (modf) [TRISYCL\\_MATH\\_WRAP3s\(remquo\)](#) [TRISYCL\\_MATH\\_WRAP2s\(P2\(rotate\)](#) namespace native

## Variables

- T [cl::sycl::y](#)
- T T [cl::sycl::z](#)

### 11.109.1 Detailed Description

Implement a wrapper around OpenCL math operations Joan.Thibault AT ens-rennes POINT fr This file is distributed under the University of Illinois Open Source License.

See LICENSE.TXT for details.

Definition in file [math.hpp](#).

### 11.109.2 Macro Definition Documentation

#### 11.109.2.1 TRISYCL\_MATH\_WRAP

```
#define TRISYCL_MATH_WRAP(  
    FUN )
```

#### Value:

```
template<typename T> \
T FUN(T x) { \
    return std::FUN(x); \
}
```

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

### 11.109.2.2 TRISYCL\_MATH\_WRAP2

```
#define TRISYCL_MATH_WRAP2(  
    FUN )
```

#### Value:

```
template<typename T>  
    T FUN(T x, T y) {  
        return std::FUN(x, y);  
    }
```

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

### 11.109.2.3 TRISYCL\_MATH\_WRAP2s

```
#define TRISYCL_MATH_WRAP2s(  
    FUN )
```

#### Value:

```
template<typename T, typename U>  
    T FUN(T x, U y) {  
        return std::FUN(x, y);  
    }
```

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

### 11.109.2.4 TRISYCL\_MATH\_WRAP3

```
#define TRISYCL_MATH_WRAP3(  
    FUN )
```

#### Value:

```
template<typename T>  
    T FUN(T x, T y, T z) {  
        return std::FUN(x, y, z);  
    }
```

Definition at line 37 of file [math.hpp](#).



## 11.109.2.5 TRISYCL\_MATH\_WRAP3s

```
#define TRISYCL_MATH_WRAP3s(
    FUN )
```

**Value:**

```
template<typename T, typename U> \
    T FUN(T x, T y, U z) { \
        return std::FUN(x, y, z); \
    }
```

Definition at line 41 of file [math.hpp](#).

## 11.109.2.6 TRISYCL\_MATH\_WRAP3ss

```
#define TRISYCL_MATH_WRAP3ss(
    FUN )
```

**Value:**

```
template<typename T, typename U> \
    T FUN(T x, U y, U z) { \
        return std::FUN(x, y, z); \
    }
```

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

## 11.110 math.hpp

```
00001 #ifndef TRISYCL_SYCL_MATH_HPP
00002 #define TRISYCL_SYCL_MATH_HPP
00003
00004 /** \file
00005     Implement a wrapper around OpenCL math operations
00006     Joan.Thibault AT ens-rennes POINT fr
00007     This file is distributed under the University of Illinois Open Source
00008     License. See LICENSE.TXT for details.
00009 */
00010
00011 #include <cmath>
00012
00013 // Include order and configure insensitive treating of unwanted macros
00014 #ifdef _MSC_VER
00015     #ifdef min
00016         #undef min
00017     #endif
00018     #ifdef max
00019         #undef max
00020     #endif
00021 #endif
00022
00023 namespace cl {
00024     namespace sycl {
00025         #define TRISYCL_MATH_WRAP(FUN) template<typename T> \
00026             T FUN(T x) { \
00027                 return std::FUN(x); \
00028             }
00029         #define TRISYCL_MATH_WRAP2(FUN) template<typename T> \
00030             T FUN(T x, T y) { \
00031                 return std::FUN(x, y); \
00032             }
```

```

00033 #define TRISYCL_MATH_WRAP2s(FUN) template<typename T, typename U>
00034     T FUN(T x, U y) {
00035         return std::FUN(x, y);
00036     }
00037 #define TRISYCL_MATH_WRAP3(FUN) template<typename T>
00038     T FUN(T x, T y, T z) {
00039         return std::FUN(x, y, z);
00040     }
00041 #define TRISYCL_MATH_WRAP3s(FUN) template<typename T, typename U>
00042     T FUN(T x, T y, U z) {
00043         return std::FUN(x, y, z);
00044     }
00045 #define TRISYCL_MATH_WRAP3ss(FUN) template<typename T, typename U>
00046     T FUN(T x, U y, U z) {
00047         return std::FUN(x, y, z);
00048     }
00049
00050 TRISYCL_MATH_WRAP(abs) //I
00051 /*TRISYCL_MATH_WRAP2(abs_diff) //I
00052 /*TRISYCL_MATH_WRAP2(add_sat) //I
00053 TRISYCL_MATH_WRAP(acos)
00054 TRISYCL_MATH_WRAP(acosh)
00055 /*TRISYCL_MATH_WRAP(acospi)
00056 TRISYCL_MATH_WRAP(asin)
00057 TRISYCL_MATH_WRAP(asinh)
00058 /*TRISYCL_MATH_WRAP(asinpi)
00059 TRISYCL_MATH_WRAP(atan) // atan(y/x)
00060 TRISYCL_MATH_WRAP2(atan2)
00061 TRISYCL_MATH_WRAP(atanh)
00062 /*TRISYCL_MATH_WRAP(atanpi)
00063 /*TRISYCL_MATH_WRAP2(atan2pi)
00064 TRISYCL_MATH_WRAP(cbrt)
00065 TRISYCL_MATH_WRAP(ceil)
00066 /*TRISYCL_MATH_WRAP3ss(clamp) //I
00067 //geninteger clamp(geninteger, sgeninteger, sgeninteger)
00068 /*TRISYCL_MATH_WRAP(clz)
00069 TRISYCL_MATH_WRAP2(copysign)
00070 TRISYCL_MATH_WRAP(cos)
00071 TRISYCL_MATH_WRAP(cosh)
00072 /*TRISYCL_MATH_WRAP(cospi)
00073 TRISYCL_MATH_WRAP(erfc)
00074 TRISYCL_MATH_WRAP(erf)
00075 TRISYCL_MATH_WRAP(exp)
00076 TRISYCL_MATH_WRAP(exp2)
00077 /*TRISYCL_MATH_WRAP(exp10)
00078 TRISYCL_MATH_WRAP(expm1)
00079 TRISYCL_MATH_WRAP(fabs)
00080 TRISYCL_MATH_WRAP2(fdim)
00081 TRISYCL_MATH_WRAP(floor)
00082 TRISYCL_MATH_WRAP3(fma)
00083 /* genfloat fmax ( genfloat x, genfloat y)
00084 * genfloat fmax ( genfloat x, sgenfloat y)
00085 */
00086 TRISYCL_MATH_WRAP2s(fmax)
00087 TRISYCL_MATH_WRAP2s(fmin)
00088 TRISYCL_MATH_WRAP2(fmod)
00089 /*TRISYCL_MATH_WRAP2s(fract)
00090 TRISYCL_MATH_WRAP2s(frexp)
00091 /*TRISYCL_MATH_WRAP(hadd)
00092 TRISYCL_MATH_WRAP2(hypot)
00093 //log
00094 //ilogb
00095 //ldexp
00096 TRISYCL_MATH_WRAP(lgamma)
00097 /*TRISYCL_MATH_WRAP2s(lgamma_r)
00098 TRISYCL_MATH_WRAP(log)
00099 TRISYCL_MATH_WRAP(log2)
00100 TRISYCL_MATH_WRAP(log10)
00101 TRISYCL_MATH_WRAP(loglp)
00102 TRISYCL_MATH_WRAP(logb)
00103 /*TRISYCL_MATH_WRAP3(mad)
00104 /*TRISYCL_MATH_WRAP3(mad_hi) //I
00105 /*TRISYCL_MATH_WRAP3(mad_sat)
00106 //
00107 //TRISYCL_MATH_WRAP3s(max) //I
00108 template<typename T>
00109 T max(T x, T y, T z) {
00110     return std::max(x, std::max(y, z));
00111 }
00112 /* geninteger max (geninteger, geninteger)
00113 * geninteger max (geninteger, sgeninteger)
00114 */
00115
00116 /*TRISYCL_MATH_WRAP2(maxmag)
00117 //
00118 //TRISYCL_MATH_WRAP3s(min) //I
00119 template<typename T>

```

```

00120 T min(T x, T y, T z) {
00121     return std::min(x, std::min(y, z));
00122 }
00123 /* geninteger min (geninteger, geninteger)
00124 * geninteger min (geninteger, sgeninteger)
00125 */
00126
00127 // *TRISYCL_MATH_WRAP2 (minmag)
00128 TRISYCL_MATH_WRAP2s (modf)
00129 // *TRISYCL_MATH_WRAP2 (mul_hi) // I
00130 // nan
00131 TRISYCL_MATH_WRAP2 (pow)
00132 // *TRISYCL_MATH_WRAP2s (posn)
00133 // *TRISYCL_MATH_WRAP2 (powr)
00134 TRISYCL_MATH_WRAP2 (remainder)
00135 TRISYCL_MATH_WRAP3s (remquo)
00136 // *TRISYCL_MATH_WRAP (rhadd) // I
00137 TRISYCL_MATH_WRAP (rint)
00138 // *TRISYCL_MATH_WRAP3s (rootn)
00139 TRISYCL_MATH_WRAP2 (rotate) // I
00140 TRISYCL_MATH_WRAP (round)
00141 // *TRISYCL_MATH_WRAP (rsqrt)
00142 TRISYCL_MATH_WRAP (sin)
00143 // *TRISYCL_MATH_WRAP2s (sincos)
00144 TRISYCL_MATH_WRAP (sinh)
00145 // *TRISYCL_MATH_WRAP (sinpi)
00146 TRISYCL_MATH_WRAP (sqrt)
00147 // *TRISYCL_MATH_WRAP2 (sub_sat)
00148 TRISYCL_MATH_WRAP (tan)
00149 TRISYCL_MATH_WRAP (tanh)
00150 // *TRISYCL_MATH_WRAP (tanpi)
00151 TRISYCL_MATH_WRAP (tgamma)
00152 TRISYCL_MATH_WRAP (trunc)
00153 /* Integer concatenation
00154 * shortn upsample (charn hi, uchar n lo)
00155 * ushortn upsample (uchar n hi, uchar n lo)
00156 * intn upsample (shortn hi, ushortn lo)
00157 * uintn upsample (ushortn hi, ushortn lo)
00158 * longlongn upsample (intn hi, uintn lo)
00159 * ulonglongn upsample (uintn hi, uintn l)
00160 */
00161 // *TRISYCL_MATH_WRAP (popcount) // I
00162 // *TRISYCL_MATH_WRAP3 (mad24)
00163 // *TRISYCL_MATH_WRAP3 (mul24)
00164
00165 //
00166 namespace native {
00167     TRISYCL_MATH_WRAP (cos)
00168     // *TRISYCL_MATH_WRAP2 (divide)
00169     TRISYCL_MATH_WRAP (exp)
00170     TRISYCL_MATH_WRAP (exp2)
00171     // *TRISYCL_MATH_WRAP (exp10)
00172     TRISYCL_MATH_WRAP (log)
00173     TRISYCL_MATH_WRAP (log2)
00174     TRISYCL_MATH_WRAP (log10)
00175     // *TRISYCL_MATH_WRAP (powr)
00176     // *TRISYCL_MATH_WRAP (recip)
00177     // *TRISYCL_MATH_WRAP (rsqrt)
00178     TRISYCL_MATH_WRAP (sin)
00179     TRISYCL_MATH_WRAP (sqrt)
00180     TRISYCL_MATH_WRAP (tan)
00181 }
00182 #undef TRISYCL_MATH_WRAP
00183 #undef TRISYCL_MATH_WRAP2
00184 #undef TRISYCL_MATH_WRAP2s
00185 #undef TRISYCL_MATH_WRAP3
00186 #undef TRISYCL_MATH_WRAP3s
00187 #undef TRISYCL_MATH_WRAP3ss
00188
00189 }
00190 }
00191
00192 #endif

```

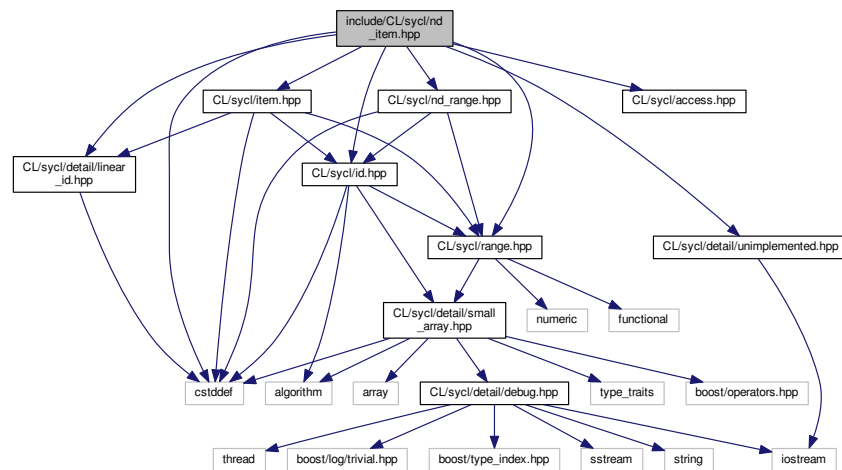
## 11.111 include/CL/sycl/nd\_item.hpp File Reference

```

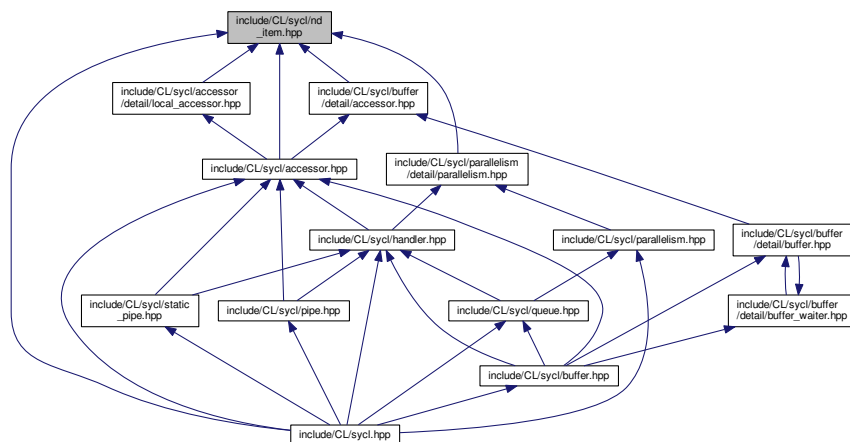
#include <cstdint>
#include "CL/sycl/access.hpp"
#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/detail/unimplemented.hpp"

```

```
#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:
```



This graph shows which files directly or indirectly include this file:



## Classes

- struct `cl::sycl::nd_item< Dimensions >`

A SYCL `nd_item` stores information on a work-item within a work-group, with some more context such as the definition ranges. [More...](#)

## Namespaces

- `cl`  
The vector type to be used as SYCL vector.
- `cl::sycl`

## 11.112 nd\_item.hpp

```

00001 #ifndef TRISYCL_SYCL_ND_ITEM_HPP
00002 #define TRISYCL_SYCL_ND_ITEM_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 <cstddef>
00013
00014 #include "CL/sycl/access.hpp"
00015 #include "CL/sycl/detail/linear_id.hpp"
00016 #include "CL/sycl/detail/unimplemented.hpp"
00017 #include "CL/sycl/id.hpp"
00018 #include "CL/sycl/item.hpp"
00019 #include "CL/sycl/nd_range.hpp"
00020 #include "CL/sycl/range.hpp"
00021
00022 namespace cl {
00023 namespace sycl {
00024
00025     /** \addtogroup parallelism Expressing parallelism through kernels
00026     @{
00027     */
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 <int Dimensions = 1>
00033     struct nd_item {
00034         /// \todo add this Boost::multi_array or STL concept to the
00035         /// specification?
00036         static constexpr auto dimensionality = Dimensions;
00037
00038     private:
00039         id<Dimensions> global_index;
00040         /* This is a cached value since it can be computed from global_index and
00041         ND_range */
00042         id<Dimensions> local_index;
00043         nd_range<Dimensions> ND_range;
00044
00045     public:
00046
00047         /** Create an empty nd_item<> from an nd_range<>
00048
00049         \todo This is for the triSYCL implementation which is expected to
00050         call set_global() and set_local() later. This should be hidden to
00051         the user.
00052         */
00053         nd_item(nd_range<Dimensions> ndr) : ND_range { ndr } {}
00054
00055
00056         /** Create a full nd_item
00057
00058         \todo This is for validation purpose. Hide this to the programmer
00059         somehow
00060         */
00061         nd_item(id<Dimensions> global_index,
00062                 nd_range<Dimensions> ndr) :
00063             global_index { global_index },
00064             // Compute the local index using the offset and the group size
00065             local_index
00066             { (global_index - ndr.get_offset())%id<Dimensions> { ndr.get_local() } },
00067             ND_range { ndr }
00068         {}
00069
00070
00071
00072         /** To be able to copy and assign nd_item, use default constructors too
00073
00074         \todo Make most of them protected, reserved to implementation
00075         */
00076         nd_item() = default;
00077
00078
00079         /** Return the constituent global id representing the work-item's
00080         position in the global iteration space
00081         */
00082         id<Dimensions> get_global() const { return
00083         global_index; }
00083

```

```

00084
00085 /** Return the constituent element of the global id representing the
00086     work-item's position in the global iteration space in the given
00087     dimension
00088 */
00089 size_t get_global(int dimension) const { return get_global()[dimension]; }
00090
00091
00092 /** Return the flattened id of the current work-item after subtracting
00093     the offset
00094 */
00095 size_t get_global_linear_id() const {
00096     return detail::linear_id(get_global_range(),
00097                             get_global(), get_offset());
00098 }
00099
00100 /** Return the constituent local id representing the work-item's
00101     position within the current work-group
00102 */
00103 id<Dimensions> get_local() const { return local_index; }
00104
00105
00106 /** Return the constituent element of the local id representing the
00107     work-item's position within the current work-group in the given
00108     dimension
00109 */
00110 size_t get_local(int dimension) const { return get_local()[dimension]; }
00111
00112
00113 /** Return the flattened id of the current work-item within the current
00114     work-group
00115 */
00116 size_t get_local_linear_id() const {
00117     return detail::linear_id(get_local_range(),
00118                             get_local());
00119 }
00120
00121 /** Return the constituent group group representing the work-group's
00122     position within the overall nd_range
00123 */
00124 id<Dimensions> get_group() const {
00125     /* Convert get_local_range() to an id<> to remove ambiguity into using
00126        implicit conversion either from range<> to id<> or the opposite */
00127     return get_global()/id<Dimensions> { get_local_range() };
00128 }
00129
00130
00131 /** Return the constituent element of the group id representing the
00132     work-group's position within the overall nd_range in the given
00133     dimension.
00134 */
00135 size_t get_group(int dimension) const {
00136     return get_group()[dimension];
00137 }
00138
00139
00140 /// Return the flattened id of the current work-group
00141 size_t get_group_linear_id() const {
00142     return detail::linear_id(get_num_groups(),
00143                             get_group());
00144 }
00145
00146 /// Return the number of groups in the nd_range
00147 id<Dimensions> get_num_groups() const {
00148     return get_nd_range().get_group();
00149 }
00150
00151 /// Return the number of groups for dimension in the nd_range
00152 size_t get_num_groups(int dimension) const {
00153     return get_num_groups()[dimension];
00154 }
00155
00156
00157 /// Return a range<> representing the dimensions of the nd_range<>
00158 range<Dimensions> get_global_range() const {
00159     return get_nd_range().get_global();
00160 }
00161
00162
00163 /// Return a range<> representing the dimensions of the current work-group
00164 range<Dimensions> get_local_range() const {
00165     return get_nd_range().get_local();
00166 }
00167

```

```

00168
00169 /** Return an id<> representing the n-dimensional offset provided to the
00170     constructor of the nd_range<> and that is added by the runtime to the
00171     global-ID of each work-item
00172     */
00173 id<Dimensions> get_offset() const { return
get_nd_range().get_offset(); }
00174
00175
00176 /// Return the nd_range<> of the current execution
00177 nd_range<Dimensions> get_nd_range() const { return
ND_range; }
00178
00179
00180 /** Allows projection down to an item
00181
00182     \todo Add to the specification
00183     */
00184 item<Dimensions> get_item() const {
00185     return { get_global_range(), get_global(),
get_offset() };
00186 }
00187
00188
00189 /** Execute a barrier with memory ordering on the local address space,
00190     global address space or both based on the value of flag
00191
00192     The current work-item will wait at the barrier until all work-items
00193     in the current work-group have reached the barrier.
00194
00195     In addition, the barrier performs a fence operation ensuring that all
00196     memory accesses in the specified address space issued before the
00197     barrier complete before those issued after the barrier
00198     */
00199 void barrier(access::fence_space flag =
00200             access::fence_space::global_and_local) const {
00201     #if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
00202         /* Use OpenMP barrier in the implementation with 1 OpenMP thread per
00203            work-item of the work-group */
00204     #pragma omp barrier
00205     #else
00206         // \todo To be implemented efficiently otherwise
00207         detail::unimplemented();
00208     #endif
00209 }
00210
00211
00212 // For the triSYCL implementation, need to set the local index
00213 void set_local(id<Dimensions> Index) { local_index = Index; }
00214
00215
00216 // For the triSYCL implementation, need to set the global index
00217 void set_global(id<Dimensions> Index) { global_index = Index; }
00218
00219 };
00220
00221 /// @} End the parallelism Doxygen group
00222
00223 }
00224 }
00225
00226 /*
00227     # Some Emacs stuff:
00228     ### Local Variables:
00229     ### ispell-local-dictionary: "american"
00230     ### eval: (flyspell-prog-mode)
00231     ### End:
00232     */
00233
00234 #endif // TRISYCL_SYCL_ND_ITEM_HPP

```

## 11.113 include/CL/sycl/nd\_range.hpp File Reference

```

#include <cstdint>
#include "CL/sycl/id.hpp"
#include "CL/sycl/range.hpp"

```





## 11.114 nd\_range.hpp

```

00001 #ifndef TRISYCL_SYCL_ND_RANGE_HPP
00002 #define TRISYCL_SYCL_ND_RANGE_HPP
00003
00004 /** \file The OpenCL SYCL nd_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 <cstdint>
00013
00014 #include "CL/sycl/id.hpp"
00015 #include "CL/sycl/range.hpp"
00016
00017 namespace cl {
00018 namespace sycl {
00019
00020 /** \addtogroup parallelism Expressing parallelism through kernels
00021     @{
00022 */
00023
00024 /** A ND-range, made by a global and local range, to specify work-group
00025     and work-item organization.
00026
00027     The local offset is used to translate the iteration space origin if
00028     needed.
00029
00030     \todo add copy constructors in the specification
00031 */
00032 template <int Dimensions = 1>
00033 struct nd_range {
00034     /// \todo add this Boost::multi_array or STL concept to the
00035     /// specification?
00036     static constexpr auto dimensionality = Dimensions;
00037
00038 private:
00039     range<dimensionality> global_range;
00040     range<dimensionality> local_range;
00041     id<dimensionality> offset;
00042
00043 public:
00044
00045     /** Construct a ND-range with all the details available in OpenCL
00046
00047         By default use a zero offset, that is iterations start at 0
00048     */
00049     nd_range(range<Dimensions> global_size,
00050             range<Dimensions> local_size,
00051             id<Dimensions> offset = {}) :
00052         global_range { global_size }, local_range { local_size }, offset { offset }
00053     {}
00054
00055
00056     /// Get the global iteration space range
00057     range<Dimensions> get_global() const { return
00058         global_range; }
00059
00060
00061     /// Get the local part of the iteration space range
00062     range<Dimensions> get_local() const { return
00063         local_range; }
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 range 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<Dimensions>{ 1 })/local_range;
00072     }
00073
00074
00075     /// \todo get_offset() is lacking in the specification
00076     id<Dimensions> 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

```

## 11.115 include/CL/sycl/opengl\_types.hpp File Reference

triSYCL wrapper for OpenCL types

```

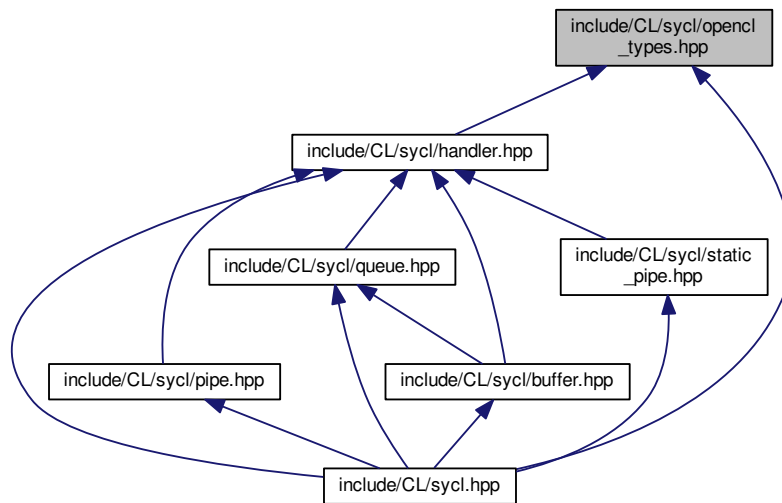
#include <boost/preprocessor/cat.hpp>
#include <boost/preprocessor/comparison/equal.hpp>
#include <boost/preprocessor/control/if.hpp>
#include <boost/preprocessor/facilities/empty.hpp>
#include <boost/preprocessor/list/for_each.hpp>
#include <boost/preprocessor/logical/not.hpp>
#include <boost/preprocessor/logical/or.hpp>
#include <boost/preprocessor/seq/for_each.hpp>
#include <boost/preprocessor/tuple/to_list.hpp>
#include <boost/preprocessor/tuple/elems.hpp>
#include <boost/compute/types/fundamental.hpp>

```

Include dependency graph for opengl\_types.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- struct `cl::sycl::is_wrapper< T >`

## Namespaces

- `cl`  
The vector type to be used as SYCL vector.
- `cl::sycl`

## Macros

- `#define TRISYCL_SCALAR_TYPES`
- `#define TRISYCL_TYPE_NAME(T) BOOST_PP_TUPLE_ELEM(3, 0, T)`
- `#define TRISYCL_TYPE_CL_NAME(T) BOOST_PP_TUPLE_ELEM(3, 1, T)`
- `#define TRISYCL_TYPE_ACTUAL_NAME(T) BOOST_PP_TUPLE_ELEM(3, 2, T)`
- `#define TRISYCL_SIZED_NAME(T, size)`
- `#define TRISYCL_IS_WRAPPER_TRAIT(type) template <> struct is_wrapper<type> : std::true_type {};`
- `#define TRISYCL_WRAPPER_CLASS_2(cl_type, boost_name, scalar_type)`
- `#define TRISYCL_WRAPPER_CLASS_3(cl_type, boost_name, scalar_type)`
- `#define TRISYCL_WRAPPER_CLASS_4(cl_type, boost_name, scalar_type)`
- `#define TRISYCL_BOOST_COMPUTE_NAME(scalar, size) TRISYCL_SIZED_NAME(boost::compute::TRISYCL_TYPE_NAME(scalar), size)`
- `#define TRISYCL_TYPEDEF_TYPE(cl_type, boost_name, scalar_type) using cl_type = boost_name##_;`
- `#define TRISYCL_H_DEFINE_TYPE(cl_type, boost_name, scalar_type, i)`
- `#define TRISYCL_DEFINE_TYPES(scalar, i)`
- `#define TRISYCL_DECLARE_CL_TYPES(r, data, scalar)`

### 11.115.1 Detailed Description

triSYCL wrapper for OpenCL types

Joan DOT Thibault AT ens-rennes DOT fr a.doumoulakis AT gmail DOT com

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file [openccl\\_types.hpp](#).

### 11.115.2 Macro Definition Documentation

#### 11.115.2.1 TRISYCL\_BOOST\_COMPUTE\_NAME

```
#define TRISYCL_BOOST_COMPUTE_NAME(  
    scalar,  
    size ) TRISYCL_SIZED_NAME(boost::compute::TRISYCL_TYPE_NAME(scalar), size)
```

Definition at line 137 of file [openccl\\_types.hpp](#).

#### 11.115.2.2 TRISYCL\_DECLARE\_CL\_TYPES

```
#define TRISYCL_DECLARE_CL_TYPES(  
    r,  
    data,  
    scalar )
```

**Value:**

```
TRISYCL_DEFINE_TYPES(scalar, 1)  
TRISYCL_DEFINE_TYPES(scalar, 2)  
TRISYCL_DEFINE_TYPES(scalar, 3)  
TRISYCL_DEFINE_TYPES(scalar, 4)  
TRISYCL_DEFINE_TYPES(scalar, 8)  
TRISYCL_DEFINE_TYPES(scalar, 16)
```

```
\\  
\\  
\\
```

Definition at line 168 of file [openccl\\_types.hpp](#).

#### 11.115.2.3 TRISYCL\_DEFINE\_TYPES

```
#define TRISYCL_DEFINE_TYPES(  
    scalar,  
    i )
```

**Value:**

```
TRISYCL_H_DEFINE_TYPE(TRISYCL_SIZED_NAME(  
    TRISYCL_TYPE_CL_NAME(scalar), i), \  
        TRISYCL_BOOST_COMPUTE_NAME(scalar, i), \  
        TRISYCL_TYPE_ACTUAL_NAME(scalar), i)
```

```
\
```

Definition at line 158 of file [openccl\\_types.hpp](#).

## 11.115.2.4 TRISYCL\_H\_DEFINE\_TYPE

```
#define TRISYCL_H_DEFINE_TYPE(
    cl_type,
    boost_name,
    scalar_type,
    i )
```

**Value:**

```
BOOST_PP_IF(BOOST_PP_EQUAL(i, 1), TRISYCL_TYPEDEF_TYPE,
    BOOST_PP_IF(BOOST_PP_EQUAL(i, 2), TRISYCL_WRAPPER_CLASS_2,
        BOOST_PP_IF(BOOST_PP_EQUAL(i, 3),
            TRISYCL_WRAPPER_CLASS_3,
            BOOST_PP_IF(BOOST_PP_EQUAL(i, 4),
                TRISYCL_WRAPPER_CLASS_4,
                TRISYCL_TYPEDEF_TYPE))))
    (cl_type, boost_name, scalar_type)
```

Definition at line 147 of file [opencil\\_types.hpp](#).

## 11.115.2.5 TRISYCL\_IS\_WRAPPER\_TRAIT

```
#define TRISYCL_IS_WRAPPER_TRAIT(
    type ) template <> struct is_wrapper<type> : std::true_type {};
```

Definition at line 83 of file [opencil\\_types.hpp](#).

## 11.115.2.6 TRISYCL\_SCALAR\_TYPES

```
#define TRISYCL_SCALAR_TYPES
```

**Value:**

```
BOOST_PP_TUPLE_TO_LIST(
    10,
    (
        ( char ,cl_char, char),
        ( uchar ,cl_uchar, unsigned char),
        ( short ,cl_short, short int),
        ( ushort ,cl_ushort, unsigned short int),
        ( int ,cl_int, int),
        ( uint ,cl_uint, unsigned int),
        ( long ,cl_long, long int),
        ( ulong ,cl_ulong, unsigned long int),
        ( float ,cl_float, float),
        ( double ,cl_double, double)
    )
)
```

Definition at line 36 of file [opencil\\_types.hpp](#).

#### 11.115.2.7 TRISYCL\_SIZED\_NAME

```
#define TRISYCL_SIZED_NAME(  
    T,  
    size )
```

**Value:**

```
BOOST_PP_IF(  
    BOOST_PP_EQUAL(size, 1), T,  
    BOOST_PP_CAT(T, size))
```

Definition at line 59 of file [opengl\\_types.hpp](#).

#### 11.115.2.8 TRISYCL\_TYPE\_ACTUAL\_NAME

```
#define TRISYCL_TYPE_ACTUAL_NAME(  
    T ) BOOST_PP_TUPLE_ELEM(3, 2, T)
```

Definition at line 56 of file [opengl\\_types.hpp](#).

#### 11.115.2.9 TRISYCL\_TYPE\_CL\_NAME

```
#define TRISYCL_TYPE_CL_NAME(  
    T ) BOOST_PP_TUPLE_ELEM(3, 1, T)
```

Definition at line 55 of file [opengl\\_types.hpp](#).

#### 11.115.2.10 TRISYCL\_TYPE\_NAME

```
#define TRISYCL_TYPE_NAME(  
    T ) BOOST_PP_TUPLE_ELEM(3, 0, T)
```

Definition at line 54 of file [opengl\\_types.hpp](#).

#### 11.115.2.11 TRISYCL\_TYPEDEF\_TYPE

```
#define TRISYCL_TYPEDEF_TYPE(  
    cl_type,  
    boost_name,  
    scalar_type ) using cl_type = boost_name##_;
```

Definition at line 143 of file [opengl\\_types.hpp](#).

## 11.115.2.12 TRISYCL\_WRAPPER\_CLASS\_2

```
#define TRISYCL_WRAPPER_CLASS_2(
    cl_type,
    boost_name,
    scalar_type )
```

**Value:**

```
class cl_type {
    ::cl_type self;

public:
    cl_type () = default;
    cl_type (::cl_type self_) : self { self_ } {}
    cl_type (scalar_type x, scalar_type y) : self { x, y } {}
    auto& x() { return self.s[0]; }
    auto& y() { return self.s[1]; }
    auto& unwrap() const { return self; };
    TRISYCL_IS_WRAPPER_TRAIT(cl_type)
```

Definition at line 91 of file [opengl\\_types.hpp](#).

## 11.115.2.13 TRISYCL\_WRAPPER\_CLASS\_3

```
#define TRISYCL_WRAPPER_CLASS_3(
    cl_type,
    boost_name,
    scalar_type )
```

**Value:**

```
class cl_type {
    ::cl_type self;

public:
    cl_type () = default;
    cl_type (::cl_type self_) : self { self_ } {}
    cl_type (scalar_type x, scalar_type y, scalar_type z) :
        self { x, y, z } {}
    auto& x() { return self.s[0]; }
    auto& y() { return self.s[1]; }
    auto& z() { return self.s[2]; }
    auto& unwrap() const { return self; };
    TRISYCL_IS_WRAPPER_TRAIT(cl_type)
```

Definition at line 104 of file [opengl\\_types.hpp](#).

## 11.115.2.14 TRISYCL\_WRAPPER\_CLASS\_4

```
#define TRISYCL_WRAPPER_CLASS_4(
    cl_type,
    boost_name,
    scalar_type )
```

**Value:**

```
class cl_type {
    ::cl_type self;

public:
    cl_type () = default;
    cl_type (::cl_type self_) : self { self_ } {}
    cl_type (scalar_type x, scalar_type y, scalar_type z, scalar_type w) :
        self { x, y, z, w } {}
    auto& x() { return self.s[0]; }
    auto& y() { return self.s[1]; }
    auto& z() { return self.s[2]; }
    auto& w() { return self.s[3]; }
    auto& unwrap() const { return self; };
    TRISYCL_IS_WRAPPER_TRAIT(cl_type)
```

Definition at line 119 of file [opengl\\_types.hpp](#).

## 11.116 opengl\_types.hpp

```
00001 #ifndef TRISYCL_SYCL_OPENGL_TYPES_HPP
00002 #define TRISYCL_SYCL_OPENGL_TYPES_HPP
00003
00004 /** \file
00005     triSYCL wrapper for OpenCL types
00006
00007     Joan DOT Thibault AT ens-rennes DOT fr
00008     a.doumoulakis AT gmail DOT com
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 #include <boost/preprocessor/cat.hpp>
00015 #include <boost/preprocessor/comparison/equal.hpp>
00016 #include <boost/preprocessor/control/if.hpp>
00017 #include <boost/preprocessor/facilities/empty.hpp>
00018 #include <boost/preprocessor/list/for_each.hpp>
00019 #include <boost/preprocessor/logical/not.hpp>
00020 #include <boost/preprocessor/logical/or.hpp>
00021 #include <boost/preprocessor/seq/for_each.hpp>
00022 #include <boost/preprocessor/tuple/to_list.hpp>
00023 #include <boost/preprocessor/tuple/elem.hpp>
00024
00025 #ifdef TRISYCL_OPENGL
00026 #include <boost/compute/types/fundamental.hpp>
00027 #else
00028 #include "CL/sycl/vec.hpp"
00029 #endif
00030
00031
00032 namespace cl {
00033 namespace sycl {
00034
00035     // List of the cl_types that will be iterated upon to generate the vector types
00036 #define TRISYCL_SCALAR_TYPES
00037     BOOST_PP_TUPLE_TO_LIST(
00038         10,
00039         (
00040             ( char ,cl_char,   char),
00041             ( uchar ,cl_uchar, unsigned char),
00042             ( short ,cl_short, short int),
00043             ( ushort ,cl_ushort, unsigned short int),
00044             ( int ,cl_int, int),
00045             ( uint ,cl_uint, unsigned int),
00046             ( long ,cl_long, long int),
```



```

00047     ( ulong ,cl_ulong,  unsigned long int),
00048     ( float ,cl_float,  float),
00049     ( double ,cl_double, double)
00050 )
00051 )
00052
00053 // Accessors to get the type name, the cl_type name and the actual type
00054 #define TRISYCL_TYPE_NAME(T)          BOOST_PP_TUPLE_ELEM(3, 0, T)
00055 #define TRISYCL_TYPE_CL_NAME(T)       BOOST_PP_TUPLE_ELEM(3, 1, T)
00056 #define TRISYCL_TYPE_ACTUAL_NAME(T)   BOOST_PP_TUPLE_ELEM(3, 2, T)
00057
00058 // Return the name of the type concatenated with the size unless the size is 1
00059 #define TRISYCL_SIZED_NAME(T, size)
00060     BOOST_PP_IF(
00061         BOOST_PP_EQUAL(size, 1), T,
00062         BOOST_PP_CAT(T, size))
00063
00064 #ifndef TRISYCL_OPENCL
00065
00066 /* If we are not using Boost Compute we just every cl_type with its \c vec
00067    equivalent
00068 */
00069 #define TRISYCL_DEFINE_TYPES(scalar, i)
00070     using TRISYCL_SIZED_NAME(TRISYCL_TYPE_CL_NAME(scalar), i) =
00071         BOOST_PP_CAT(TRISYCL_TYPE_NAME(scalar), i);
00072
00073 #else
00074
00075 /* When passing the arguments to Boost Compute, we need to know if we have to
00076    unwrap our type and give Boost Compute the actual OpenCL type. To do this
00077    we define this meta-function that returns true if \c T is a wrapper type
00078    to an OpenCL type. For example \c is_wrapper<cl::sycl::int3>::value == true
00079 */
00080 template<class T>
00081 struct is_wrapper : std::false_type {};
00082
00083 #define TRISYCL_IS_WRAPPER_TRAIT(type)
00084 template <> struct is_wrapper<type> : std::true_type {};
00085
00086 /* We define 3 different wrapper classes around OpenCL types.
00087    These classes allow us to use t.x() and t.y() for vector types of size 2.
00088    And in addition we can also use t.z() and t.w() for vector types of size
00089    3 and 4 similarly to OpenCL vector types
00090 */
00091 #define TRISYCL_WRAPPER_CLASS_2(cl_type, boost_name, scalar_type)
00092     class cl_type {
00093     public:
00094         cl_type () = default;
00095         cl_type (::cl_type self_) : self { self_ } {}
00096         cl_type (scalar_type x, scalar_type y) : self { x, y } {}
00097         auto& x() { return self.s[0]; }
00098         auto& y() { return self.s[1]; }
00099         auto& unwrap() const { return self; };
00100         TRISYCL_IS_WRAPPER_TRAIT(cl_type)
00101     };
00102
00103 #define TRISYCL_WRAPPER_CLASS_3(cl_type, boost_name, scalar_type)
00104     class cl_type {
00105     public:
00106         cl_type () = default;
00107         cl_type (::cl_type self_) : self { self_ } {}
00108         cl_type (scalar_type x, scalar_type y, scalar_type z) :
00109             self { x, y, z } {}
00110         auto& x() { return self.s[0]; }
00111         auto& y() { return self.s[1]; }
00112         auto& z() { return self.s[2]; }
00113         auto& unwrap() const { return self; };
00114         TRISYCL_IS_WRAPPER_TRAIT(cl_type)
00115     };
00116
00117 #define TRISYCL_WRAPPER_CLASS_4(cl_type, boost_name, scalar_type)
00118     class cl_type {
00119     public:
00120         cl_type () = default;
00121         cl_type (::cl_type self_) : self { self_ } {}
00122         cl_type (scalar_type x, scalar_type y, scalar_type z, scalar_type w) :
00123             self { x, y, z, w } {}
00124         auto& x() { return self.s[0]; }
00125         auto& y() { return self.s[1]; }
00126         auto& z() { return self.s[2]; }
00127         auto& w() { return self.s[3]; }
00128         auto& unwrap() const { return self; };
00129         TRISYCL_IS_WRAPPER_TRAIT(cl_type)
00130     };

```

```

00134
00135
00136 // Return the Boost Compute type for OpenCL vector types
00137 #define TRISYCL_BOOST_COMPUTE_NAME(scalar, size) \
00138     TRISYCL_SIZED_NAME(boost::compute::TRISYCL_TYPE_NAME(scalar), size)
00139
00140 /* For vector types of size above 4 and 1 we typedef the vector type with
00141     its Boost Compute equivalent
00142 */
00143 #define TRISYCL_TYPEDEF_TYPE(cl_type, boost_name, scalar_type) \
00144     using cl_type = boost_name##_;
00145
00146 // Helper macro to properly define each OpenCL type
00147 #define TRISYCL_H_DEFINE_TYPE(cl_type, boost_name, scalar_type, i) \
00148     BOOST_PP_IF(BOOST_PP_EQUAL(i, 1), TRISYCL_TYPEDEF_TYPE, \
00149         BOOST_PP_IF(BOOST_PP_EQUAL(i, 2), TRISYCL_WRAPPER_CLASS_2, \
00150             BOOST_PP_IF(BOOST_PP_EQUAL(i, 3), \
00151                 TRISYCL_WRAPPER_CLASS_3, \
00152                     BOOST_PP_IF(BOOST_PP_EQUAL(i, 4), \
00153                         TRISYCL_WRAPPER_CLASS_4, \
00154                             TRISYCL_TYPEDEF_TYPE)))) \
00155     (cl_type, boost_name, scalar_type)
00156
00157 // Helper macro to properly define each OpenCL type
00158 #define TRISYCL_DEFINE_TYPES(scalar, i) \
00159     TRISYCL_H_DEFINE_TYPE(TRISYCL_SIZED_NAME(TRISYCL_TYPE_CL_NAME(scalar), i), \
00160         TRISYCL_BOOST_COMPUTE_NAME(scalar, i), \
00161         TRISYCL_TYPE_ACTUAL_NAME(scalar), i)
00162
00163 #endif
00164
00165 /* We declare vector types of sizes 1,2,3,4,8 and 16 as per
00166     the SYCL specification
00167 */
00168 #define TRISYCL_DECLARE_CL_TYPES(r, data, scalar) \
00169     TRISYCL_DEFINE_TYPES(scalar, 1) \
00170     TRISYCL_DEFINE_TYPES(scalar, 2) \
00171     TRISYCL_DEFINE_TYPES(scalar, 3) \
00172     TRISYCL_DEFINE_TYPES(scalar, 4) \
00173     TRISYCL_DEFINE_TYPES(scalar, 8) \
00174     TRISYCL_DEFINE_TYPES(scalar, 16)
00175
00176 // Generate the vector types for all listed scalar types
00177 BOOST_PP_LIST_FOR_EACH(TRISYCL_DECLARE_CL_TYPES, _, \
00178     TRISYCL_SCALAR_TYPES)
00179
00178
00179 } // sycl
00180 } // cl
00181
00182
00183 // Undef macros to avoid name collision
00184 #undef TRISYCL_SCALAR_TYPES
00185 #undef TRISYCL_TYPE_NAME
00186 #undef TRISYCL_TYPE_CL_NAME
00187 #undef TRISYCL_TYPE_ACTUAL_NAME
00188 #undef TRISYCL_SIZED_NAME
00189 #undef TRISYCL_IS_WRAPPER_TRAIT
00190 #undef TRISYCL_WRAPPER_CLASS_2
00191 #undef TRISYCL_WRAPPER_CLASS_3
00192 #undef TRISYCL_WRAPPER_CLASS_4
00193 #undef TRISYCL_BOOST_COMPUTE_NAME
00194 #undef TRISYCL_TYPEDEF_TYPE
00195 #undef TRISYCL_DEFINE_TYPES
00196 #undef TRISYCL_DECLARE_CL_TYPES
00197
00198 /*
00199     # Some Emacs stuff:
00200     ### Local Variables:
00201     ###   ispell-local-dictionary: "american"
00202     ###   eval: (flyspell-prog-mode)
00203     ### End:
00204 */
00205
00206 #endif // TRISYCL_SYCL_OPENCL_TYPES_HPP

```

## 11.117 include/CL/sycl/parallelism/detail/parallelism.hpp File Reference

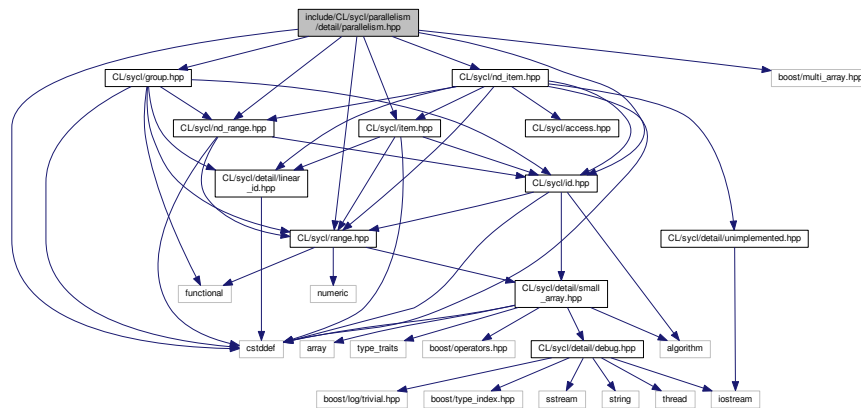
Implement the detail of the parallel constructions to launch kernels.

```

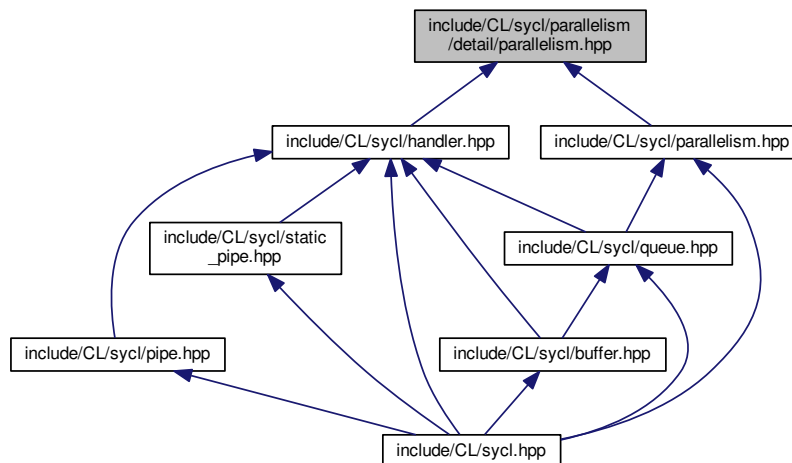
#include <cstdint>
#include <boost/multi_array.hpp>

```

Include 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<int Dimensions = 1, typename ParallelForFuncor , typename Id >`  
`void cl::sycl::detail::parallel\_for (range< Dimensions > r, ParallelForFuncor f, Id)`  
*Implementation of a data parallel computation with parallelism specified at launch time by a range<>.*
- `template<int Dimensions = 1, typename ParallelForFuncor >`  
`void cl::sycl::detail::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<int Dimensions = 1, typename ParallelForFuncor >`  
`void cl::sycl::detail::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<int Dimensions = 1, typename ParallelForFuncor >`  
`void cl::sycl::detail::parallel\_for\_global\_offset (range< Dimensions > global_size, id< Dimensions > offset, ParallelForFuncor f)`  
*Implementation of parallel\_for with a range<> and an offset.*
- `template<int Dimensions = 1, typename ParallelForFuncor >`  
`void cl::sycl::detail::parallel\_for (nd_range< Dimensions > r, ParallelForFuncor f)`  
*Implement a variation of parallel\_for to take into account a nd\_range<>*
- `template<int Dimensions = 1, typename ParallelForFuncor >`  
`void cl::sycl::detail::parallel\_for\_workgroup (nd_range< Dimensions > r, ParallelForFuncor f)`  
*Implement the loop on the work-groups.*
- `template<int Dimensions = 1, typename ParallelForFuncor >`  
`void cl::sycl::detail::parallel\_for\_workitem (const group< Dimensions > &g, ParallelForFuncor f)`  
*Implement the loop on the work-items inside a work-group.*

### 11.117.1 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](#).

## 11.118 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,
00047           typename Range,
00048           typename ParallelForFunc,
00049           typename Id>
00050 struct parallel_for_iterate {
00051     parallel_for_iterate(Range r, ParallelForFunc &f, Id &index) {
00052         for (boost::multi_array_types::index _sycl_index = 0,
00053              _sycl_end = r[Range::dimensionality - level];
00054              _sycl_index < _sycl_end;
00055              _sycl_index++) {
00056             // Set the current value of the index for this dimension
00057             index[Range::dimensionality - level] = _sycl_index;
00058             // Iterate further on lower dimensions
00059             parallel_for_iterate<level - 1,
00060                                Range,
00061                                ParallelForFunc,
00062                                Id> { r, f, index };
00063         }
00064     }
00065 };
00066
00067
00068 /** A top-level recursive multi-dimensional iterator variant using OpenMP
00069
00070     Only the top-level loop uses OpenMP and goes on with the normal
00071     recursive multi-dimensional.
00072 */
00073 template <std::size_t level,
00074           typename Range,
00075           typename ParallelForFunc,
00076           typename Id>
00077 struct parallel_OpenMP_for_iterate {
00078     parallel_OpenMP_for_iterate(Range r, ParallelForFunc &f) {
00079         // Create the OpenMP threads before the for-loop to avoid creating an
00080         // index in each iteration
00081         #pragma omp parallel
00082         {
00083             // Allocate an OpenMP thread-local index
00084             Id index;

```

```

00085     // Make a simple loop end condition for OpenMP
00086     boost::multi_array_types::index _sycl_end =
00087         r[Range::dimensionality - level];
00088     /* Distribute the iterations on the OpenMP threads. Some OpenMP
00089        "collapse" could be useful for small iteration space, but it
00090        would need some template specialization to have real contiguous
00091        loop nests */
00092     #pragma omp for
00093     for (boost::multi_array_types::index _sycl_index = 0;
00094          _sycl_index < _sycl_end;
00095          _sycl_index++) {
00096         // Set the current value of the index for this dimension
00097         index[Range::dimensionality - level] = _sycl_index;
00098         // Iterate further on lower dimensions
00099         parallel_for_iterate<level - 1,
00100                                Range,
00101                                ParallelForFuncor,
00102                                Id> { r, f, index };
00103     }
00104 }
00105 }
00106 };
00107
00108
00109 /** Stop the recursion when level reaches 0 by simply calling the
00110     kernel functor with the constructed id */
00111 template <typename Range, typename ParallelForFuncor, typename Id>
00112 struct parallel_for_iterate<0, Range, ParallelForFuncor, Id> {
00113     parallel_for_iterate(Range r, ParallelForFuncor &f, Id &index) {
00114         f(index);
00115     }
00116 };
00117
00118
00119 /** Implementation of a data parallel computation with parallelism
00120     specified at launch time by a range<>. Kernel index is id or int.
00121
00122     This implementation use OpenMP 3 if compiled with the right flag.
00123 */
00124 template <int Dimensions = 1, typename ParallelForFuncor, typename Id>
00125 void parallel_for(range<Dimensions> r,
00126                  ParallelForFuncor f,
00127                  Id) {
00128     #ifdef _OPENMP
00129         // Use OpenMP for the top loop level
00130         parallel_OpenMP_for_iterate<Dimensions,
00131                                     range<Dimensions>,
00132                                     ParallelForFuncor,
00133                                     id<Dimensions>> { r, f };
00134     #else
00135         // In a sequential execution there is only one index processed at a time
00136         id<Dimensions> index;
00137         parallel_for_iterate<Dimensions,
00138                             range<Dimensions>,
00139                             ParallelForFuncor,
00140                             id<Dimensions>> { r, f, index };
00141     #endif
00142 }
00143
00144
00145 /** Implementation of a data parallel computation with parallelism
00146     specified at launch time by a range<>. Kernel index is item.
00147
00148     This implementation use OpenMP 3 if compiled with the right flag.
00149 */
00150 template <int Dimensions = 1, typename ParallelForFuncor>
00151 void parallel_for(range<Dimensions> r,
00152                  ParallelForFuncor f,
00153                  item<Dimensions>) {
00154     auto reconstruct_item = [&] (id<Dimensions> l) {
00155         // Reconstruct the global item
00156         item<Dimensions> index { r, l };
00157         // Call the user kernel with the item<> instead of the id<>
00158         f(index);
00159     };
00160     #ifdef _OPENMP
00161         // Use OpenMP for the top loop level
00162         parallel_OpenMP_for_iterate<Dimensions,
00163                                     range<Dimensions>,
00164                                     decltype(reconstruct_item),
00165                                     id<Dimensions>> { r, reconstruct_item };
00166     #else
00167         // In a sequential execution there is only one index processed at a time
00168         id<Dimensions> index;
00169         parallel_for_iterate<Dimensions,
00170                             range<Dimensions>,
00171                             decltype(reconstruct_item),

```

```

00172         id<Dimensions>> { r, reconstruct_item, index };
00173 #endif
00174 }
00175
00176
00177 /** Calls the appropriate ternary parallel_for overload based on the
00178     index type of the kernel function object f
00179
00180 */
00181 template <int Dimensions = 1, typename ParallelForFuncor>
00182 void parallel_for(range<Dimensions> r, ParallelForFuncor f) {
00183     using mf_t = decltype(std::mem_fn(&ParallelForFuncor::operator()));
00184     using arg_t = typename mf_t::second_argument_type;
00185     parallel_for(r, f, arg_t{});
00186 }
00187
00188
00189 /** Implementation of parallel_for with a range<> and an offset */
00190 template <int Dimensions = 1, typename ParallelForFuncor>
00191 void parallel_for_global_offset(range<Dimensions> global_size,
00192                                id<Dimensions> offset,
00193                                ParallelForFuncor f) {
00194     // Reconstruct the item from its id<> and its offset
00195     auto reconstruct_item = [&] (id<Dimensions> l) {
00196         // Reconstruct the global item
00197         item<Dimensions> index { global_size, 1 + offset, offset };
00198         // Call the user kernel with the item<> instead of the id<>
00199         f(index);
00200     };
00201
00202     // First iterate on all the work-groups
00203     parallel_for(global_size, reconstruct_item);
00204 }
00205
00206
00207 /** Implement a variation of parallel_for to take into account a
00208     nd_range<>
00209
00210     \todo Add an OpenMP implementation
00211
00212     \todo Deal with incomplete work-groups
00213
00214     \todo Implement with parallel_for_workgroup()/parallel_for_workitem()
00215 */
00216 template <int Dimensions = 1, typename ParallelForFuncor>
00217 void parallel_for(nd_range<Dimensions> r,
00218                 ParallelForFuncor f) {
00219     // To iterate on the work-group
00220     id<Dimensions> group;
00221     range<Dimensions> group_range = r.get_group();
00222
00223 #ifdef _OPENMP
00224
00225     auto iterate_in_work_group = [&] (id<Dimensions> g) {
00226         //group.display();
00227
00228         // Then iterate on the local work-groups
00229         cl::sycl::group<Dimensions> wg {g, r};
00230         parallel_for_workitem<Dimensions>(
00231             decltype(f)>(wg, f);
00232     };
00233
00234 #else
00235
00236     // In a sequential execution there is only one index processed at a time
00237     nd_item<Dimensions> index { r };
00238
00239     // To iterate on the local work-item
00240     id<Dimensions> local;
00241     range<Dimensions> local_range = r.get_local();
00242
00243     // Reconstruct the nd_item from its group and local id
00244     auto reconstruct_item = [&] (id<Dimensions> l) {
00245         //local.display();
00246         // Reconstruct the global nd_item
00247         index.set_local(local);
00248         // Upgrade local_range to an id<> so that we can * with the group (an id<>)
00249         index.set_global(local + id<Dimensions>(local_range)*group);
00250         // Call the user kernel at last
00251         f(index);
00252     };
00253
00254     /* To recycle the parallel_for on range<>, wrap the ParallelForFuncor f
00255        into another functor that iterates inside the work-group and then
00256        calls f */
00257     auto iterate_in_work_group = [&] (id<Dimensions> g) {
00258         //group.display();

```

```

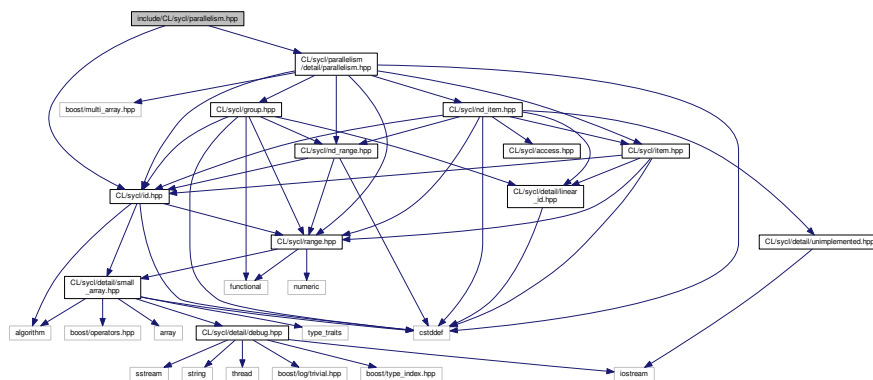
00259 // Then iterate on the local work-groups
00260 parallel_for_iterate<Dimensions,
00261                     range<Dimensions>,
00262                     decltype(reconstruct_item),
00263                     id<Dimensions>>> { local_range,
00264                                     reconstruct_item,
00265                                     local };
00266 };
00267
00268 #endif
00269
00270 // First iterate on all the work-groups
00271 parallel_for_iterate<Dimensions,
00272                     range<Dimensions>,
00273                     decltype(iterate_in_work_group),
00274                     id<Dimensions>>> { group_range,
00275                                     iterate_in_work_group,
00276                                     group };
00277 }
00278
00279
00280 /// Implement the loop on the work-groups
00281 template <int Dimensions = 1, typename ParallelForFuncor>
00282 void parallel_for_workgroup(nd_range<Dimensions> r,
00283                            ParallelForFuncor f) {
00284 // In a sequential execution there is only one index processed at a time
00285 group<Dimensions> g { r };
00286
00287 // First iterate on all the work-groups
00288 parallel_for_iterate<Dimensions,
00289                     range<Dimensions>,
00290                     ParallelForFuncor,
00291                     group<Dimensions>>> {
00292     r.get_group(),
00293     f,
00294     g };
00295 }
00296
00297
00298 /** Implement the loop on the work-items inside a work-group
00299
00300     \todo Better type the functor
00301 */
00302 template <int Dimensions, typename ParallelForFuncor>
00303 void parallel_for_workitem(const group<Dimensions> &g,
00304                           ParallelForFuncor f) {
00305 #if defined(_OPENMP) && (!defined(TRISYCL_NO_BARRIER) && !defined(_MSC_VER))
00306 /* To implement barriers With OpenMP, one thread is created for each
00307    work-item in the group and thus an OpenMP barrier has the same effect
00308    of an OpenCL barrier executed by the work-items in a workgroup
00309
00310    The issue is that the parallel_for_workitem() execution is slow even
00311    when nd_item::barrier() is not used
00312
00313    \todo Simplify by just using omp parallel for collapse
00314 */
00315
00316 range<Dimensions> l_r = g.get_nd_range().get_local();
00317 auto tot = l_r.get(0);
00318 for (int i = 1; i < (int) Dimensions; ++i){
00319     tot *= l_r.get(i);
00320 }
00321 #pragma omp parallel num_threads(tot)
00322 {
00323     nd_item<Dimensions> index { g.get_nd_range() };
00324     id<Dimensions> local; // to initialize correctly
00325 #pragma omp for nowait
00326     for (std::size_t th_id = 0; th_id < tot; ++th_id) {
00327         if (Dimensions == 1) {
00328             local[0] = th_id;
00329         } else if (Dimensions == 2) {
00330             local[0] = th_id / l_r.get(1);
00331             local[1] = th_id % l_r.get(1);
00332         } else if (Dimensions == 3) {
00333             local[0] = th_id / (l_r.get(1)*l_r.get(2));
00334             local[1] = (th_id / l_r.get(2)) % l_r.get(1);
00335             local[2] = th_id % l_r.get(2);
00336         }
00337         index.set_local(local);
00338         index.set_global(local + id<Dimensions>(l_r)*g.get());
00339         f(index);
00340     }
00341 }
00342 #else
00343 // In a sequential execution there is only one index processed at a time
00344 nd_item<Dimensions> index { g.get_nd_range() };
00345 // To iterate on the local work-item

```

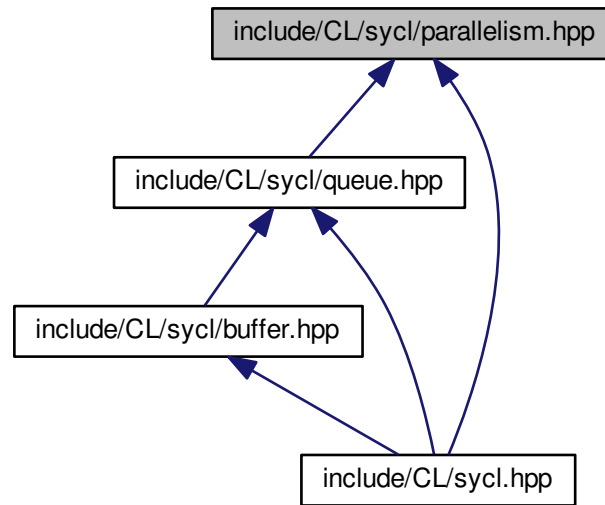


## 11.119 include/CL/sycl/parallelism.hpp File Reference

```
#include "CL/sycl/parallelism/detail/parallelism.hpp"
#include "CL/sycl/id.hpp"
Include dependency graph for parallelism.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](#)

## Functions

- `template<int Dimensions = 1, typename ParallelForFuncor >`  
`void cl::sycl::parallel\_for\_work\_item (const group< Dimensions > &g, ParallelForFuncor f)`  
*SYCL `parallel_for` version that allows a Program object to be specified.*

### 11.119.1 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](#).

## 11.120 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 #include "CL/sycl/id.hpp"
00016
00017 namespace cl {
00018 namespace sycl {
00019
00020 /** \addtogroup parallelism
00021     @{
00022 */
00023
00024 /// SYCL parallel_for version that allows a Program object to be specified
00025 /// \todo To be implemented
00026 /* template <typename Range, typename Program, typename ParallelForFunctor>
00027 void parallel_for(Range r, Program p, ParallelForFunctor f) {
00028     /// \todo deal with Program
00029     parallel_for(r, f);
00030 }
00031 */
00032
00033 /** Loop on the work-items inside a work-group
00034
00035     \todo Deprecate this function in the specification to use
00036     instead the group method
00037 */
00038 template <int Dimensions = 1, typename ParallelForFunctor>
00039 void parallel_for_work_item(const group<Dimensions> &g,
00040                             ParallelForFunctor f) {
00041     g.parallel_for_work_item(f);
00042 }
00043
00044
00045
00046 }
00047 }
00048
00049 /// @} End the parallelism Doxygen group
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_PARALLELISM_HPP

```

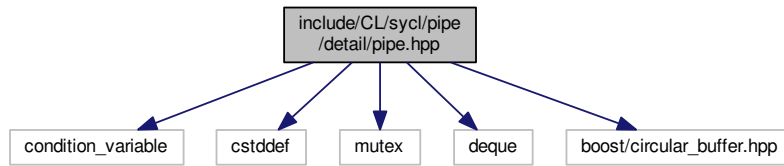
## 11.121 include/CL/sycl/pipe/detail/pipe.hpp File Reference

```

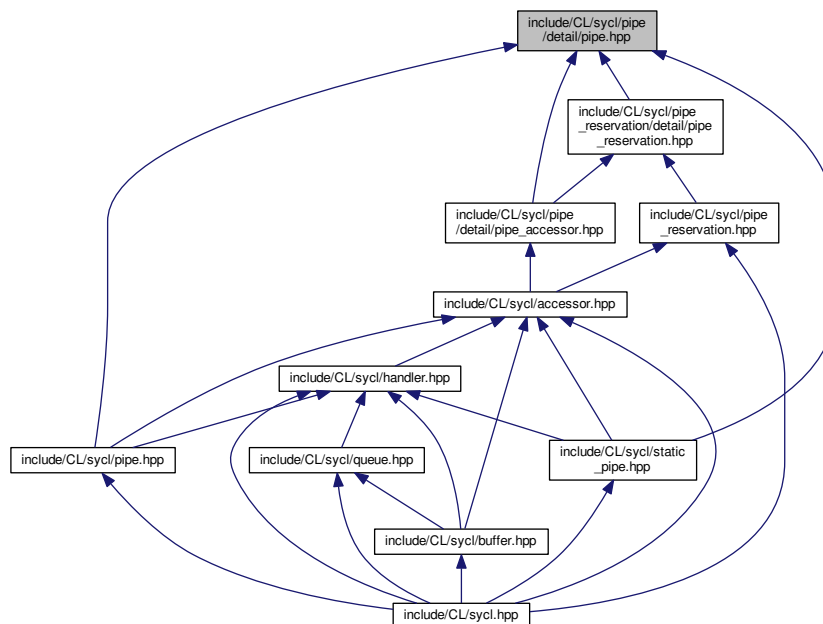
#include <condition_variable>
#include <cstdint>
#include <mutex>
#include <deque>
#include <boost/circular_buffer.hpp>

```

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`

## 11.122 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 TRISYCL_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 private:
00077     /// The circular buffer to store the elements
00078     boost::circular_buffer<value_type> cb;
00079
00080     /** 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 #ifndef _MSC_VER
00095     using rid_iterator = typename decltype(w_rid_q)::iterator;
00096 #else
00097     using rid_iterator = typename std::deque<reserve_id<value_type>>::iterator;
00098 #endif
00099
00100 private:
00101
00102     /// The queue of pending read reservations
00103     std::deque<reserve_id<value_type>> r_rid_q;
00104
00105     /// Track the number of frozen elements related to read reservations
00106     std::size_t read_reserved_frozen;
00107
00108     /// To signal that a read has been successful
00109     std::condition_variable read_done;
00110
00111     /// To signal that a write has been successful
00112     std::condition_variable write_done;
00113
00114     /// To control the debug mode, disabled by default
00115     bool debug_mode = false;
00116
00117 public:
00118
00119     /// True when the pipe is currently used for reading
00120     bool used_for_reading = false;
00121
00122     /// True when the pipe is currently used for writing
00123     bool used_for_writing = false;
00124
00125     /// Create a pipe as a circular buffer of the required capacity
00126     pipe(std::size_t capacity) : cb { capacity }, read_reserved_frozen { 0 } { }
00127
00128
00129     /** Return the maximum number of elements that can fit in the pipe
00130     */
00131     std::size_t capacity() const {
00132         // No lock required since it is fixed and set at construction time
00133         return cb.capacity();
00134     }
00135
00136 private:
00137
00138     /** Get the current number of elements in the pipe that can be read
00139
00140     This is obviously a volatile value which is constrained by the
00141     theory of restricted relativity.
00142
00143     Note that on some devices it may be costly to implement (for
00144     example on FPGA).
00145     */
00146     std::size_t size() const {
00147         TRISYCL_DUMP_T("size() cb.size() = " << cb.size()
00148             << " cb.end() = " << (void *)&cb.end()
00149             << " reserved_for_reading() = " << reserved_for_reading()
00150             << " reserved_for_writing() = " << reserved_for_writing());
00151         /* The actual number of available elements depends from the
00152            elements blocked by some reservations.
00153            This prevents a consumer to read into reserved area. */
00154         return cb.size() - reserved_for_reading() - reserved_for_writing();
00155     }
00156
00157
00158     /** Test if the pipe is empty
00159
00160     This is obviously a volatile value which is constrained by
00161     restricted relativity.
00162
00163     Note that on some devices it may be costly to implement on the
00164     write side (for example on FPGA).
00165     */
00166     bool empty() const {
00167         TRISYCL_DUMP_T("empty() cb.size() = " << cb.size()
00168             << " size() = " << size());
00169         // It is empty when the size is zero, taking into account reservations
00170         return size() == 0;
00171     }

```

```

00172
00173
00174  /** Test if the pipe is full
00175
00176      This is obviously a volatile value which is constrained by
00177      restricted relativity.
00178
00179      Note that on some devices it may be costly to implement on the
00180      read side (for example on FPGA).
00181  */
00182  bool full() const {
00183      return cb.full();
00184  }
00185
00186
00187 public:
00188
00189  /// The size() method used outside needs to lock the datastructure
00190  std::size_t size_with_lock() const {
00191      std::lock_guard<std::mutex> lg { cb_mutex };
00192      return size();
00193  }
00194
00195
00196  /// The empty() method used outside needs to lock the datastructure
00197  bool empty_with_lock() const {
00198      std::lock_guard<std::mutex> lg { cb_mutex };
00199      return empty();
00200  }
00201
00202
00203  // The full() method used outside needs to lock the datastructure
00204  bool full_with_lock() const {
00205      std::lock_guard<std::mutex> lg { cb_mutex };
00206      return full();
00207  }
00208
00209
00210  /** Try to write a value to the pipe
00211
00212      \param[in] value is what we want to write
00213
00214      \param[in] blocking specify if the call wait for the operation
00215      to succeed
00216
00217      \return true on success
00218
00219      \todo provide a && version
00220  */
00221  bool write(const T &value, bool blocking = false) {
00222      // Lock the pipe to avoid being disturbed
00223      std::unique_lock<std::mutex> ul { cb_mutex };
00224      TRISYCL_DUMP_T("Write pipe full = " << full()
00225          << " value = " << value);
00226
00227      if (blocking)
00228          /* If in blocking mode, wait for the not full condition, that
00229             may be changed when a read is done */
00230          read_done.wait(ul, [&] { return !full(); });
00231      else if (full())
00232          return false;
00233
00234      cb.push_back(value);
00235      TRISYCL_DUMP_T("Write pipe front = " << cb.front()
00236          << " back = " << cb.back()
00237          << " cb.begin() = " << (void *)&cb.begin()
00238          << " cb.size() = " << cb.size()
00239          << " cb.end() = " << (void *)&cb.end()
00240          << " reserved_for_reading() = " << reserved_for_reading()
00241          << " reserved_for_writing() = " << reserved_for_writing());
00242      // Notify the clients waiting to read something from the pipe
00243      write_done.notify_all();
00244      return true;
00245  }
00246
00247
00248  /** Try to read a value from the pipe
00249
00250      \param[out] value is the reference to where to store what is
00251      read
00252
00253      \param[in] blocking specify if the call wait for the operation
00254      to succeed
00255
00256      \return true on success
00257  */
00258  bool read(T &value, bool blocking = false) {

```

```

00259 // Lock the pipe to avoid being disturbed
00260 std::unique_lock<std::mutex> ul { cb_mutex };
00261 TRISYCL_DUMP_T("Read pipe empty = " << empty());
00262
00263 if (blocking)
00264     /* If in blocking mode, wait for the not empty condition, that
00265        may be changed when a write is done */
00266     write_done.wait(ul, [&] { return !empty(); });
00267 else if (empty())
00268     return false;
00269
00270 TRISYCL_DUMP_T("Read pipe front = " << cb.front()
00271                << " back = " << cb.back()
00272                << " reserved_for_reading() = " << reserved_for_reading());
00273 if (read_reserved_frozen)
00274     /** If there is a pending reservation, read the next element to
00275        be read and update the number of reserved elements */
00276     value = cb.begin()[read_reserved_frozen++];
00277 else {
00278     /* There is no pending read reservation, so pop the read value
00279        from the pipe */
00280     value = cb.front();
00281     cb.pop_front();
00282 }
00283
00284 TRISYCL_DUMP_T("Read pipe value = " << value);
00285 // Notify the clients waiting for some room to write in the pipe
00286 read_done.notify_all();
00287 return true;
00288 }
00289
00290 /** Compute the amount of elements blocked by read reservations, not yet
00291     committed
00292
00293     This includes some normal reads to pipes between/after
00294     un-committed reservations
00295
00296     This function assumes that the data structure is locked
00297 */
00298 std::size_t reserved_for_reading() const {
00299     return read_reserved_frozen;
00300 }
00301
00302 /** Compute the amount of elements blocked by write reservations, not yet
00303     committed
00304
00305     This includes some normal writes to pipes between/after
00306     un-committed reservations
00307
00308     This function assumes that the data structure is locked
00309 */
00310 std::size_t reserved_for_writing() const {
00311     if (w_rid_q.empty())
00312         // No on-going reservation
00313         return 0;
00314     else
00315         /* The reserved size is from the first element of the first
00316            on-going reservation up to the end of the pipe content */
00317         return cb.end() - w_rid_q.front().start;
00318 }
00319
00320 /** Reserve some part of the pipe for reading
00321
00322     \param[in] s is the number of element to reserve
00323
00324     \param[out] rid is an iterator to a description of the
00325     reservation that has been done if successful
00326
00327     \param[in] blocking specify if the call wait for the operation
00328     to succeed
00329
00330     \return true if the reservation was successful
00331 */
00332 bool reserve_read(std::size_t s,
00333                  rid_iterator &rid,
00334                  bool blocking = false) {
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;

```



```

00346
00347     if (blocking)
00348         /* If in blocking mode, wait for enough elements to read in the
00349            pipe for the reservation. This condition can change when a
00350            write is done */
00351         write_done.wait(ul, [&] { return s <= size(); });
00352     else if (s > size())
00353         // Not enough elements to read in the pipe for the reservation
00354         return false;
00355
00356     // Compute the location of the first element of the reservation
00357     auto first = cb.begin() + read_reserved_frozen;
00358     // Increment the number of frozen elements
00359     read_reserved_frozen += s;
00360     /* Add a description of the reservation at the end of the
00361        reservation queue */
00362     r_rid_q.emplace_back(first, s);
00363     // Return the iterator to the last reservation descriptor
00364     rid = r_rid_q.end() - 1;
00365     TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
00366                   << " size() = " << size());
00367     return true;
00368 }
00369
00370
00371 /** Reserve some part of the pipe for writing
00372
00373     \param[in] s is the number of element to reserve
00374
00375     \param[out] rid is an iterator to a description of the
00376         reservation that has been done if successful
00377
00378     \param[in] blocking specify if the call wait for the operation
00379         to succeed
00380
00381     \return true if the reservation was successful
00382 */
00383 bool reserve_write(std::size_t s,
00384                  rid_iterator &rid,
00385                  bool blocking = false) {
00386     // Lock the pipe to avoid being disturbed
00387     std::unique_lock<std::mutex> ul { cb_mutex };
00388
00389     TRISYCL_DUMP_T("Before write reservation cb.size() = " << cb.size()
00390                   << " size() = " << size());
00391     if (s == 0)
00392         // Empty reservation requested, so nothing to do
00393         return false;
00394
00395     if (blocking)
00396         /* If in blocking mode, wait for enough room in the pipe, that
00397            may be changed when a read is done. Do not use a difference
00398            here because it is only about unsigned values */
00399         read_done.wait(ul, [&] { return cb.size() + s <= capacity(); });
00400     else if (cb.size() + s > capacity())
00401         // Not enough room in the pipe for the reservation
00402         return false;
00403
00404     /* If there is enough room in the pipe, just create default values
00405        in it to do the reservation */
00406     for (std::size_t i = 0; i != s; ++i)
00407         cb.push_back();
00408     /* Compute the location of the first element a posteriori since it
00409        may not exist a priori if cb was empty before */
00410     auto first = cb.end() - s;
00411     /* Add a description of the reservation at the end of the
00412        reservation queue */
00413     w_rid_q.emplace_back(first, s);
00414     // Return the iterator to the last reservation descriptor
00415     rid = w_rid_q.end() - 1;
00416     TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
00417                   << " size() = " << size());
00418     return true;
00419 }
00420
00421
00422 /** Process the read reservations that are ready to be released in the
00423     reservation queue
00424 */
00425 void move_read_reservation_forward() {
00426     // Lock the pipe to avoid nuisance
00427     std::lock_guard<std::mutex> lg { cb_mutex };
00428
00429     for (;;) {
00430         if (r_rid_q.empty())
00431             // No pending reservation, so nothing to do
00432             break;

```

```

00433     if (!r_rid_q.front().ready)
00434         /* If the first reservation is not ready to be released, stop
00435            because it is blocking all the following in the queue
00436            anyway */
00437         break;
00438     // Remove the reservation to be released from the queue
00439     r_rid_q.pop_front();
00440     std::size_t n_to_pop;
00441     if (r_rid_q.empty())
00442         // If it was the last one, remove all the reservation
00443         n_to_pop = read_reserved_frozen;
00444     else
00445         // Else remove everything up to the next reservation
00446         n_to_pop = r_rid_q.front().start - cb.begin();
00447     // No longer take into account these reserved slots
00448     read_reserved_frozen -= n_to_pop;
00449     // Release the elements from the FIFO
00450     while (n_to_pop--)
00451         cb.pop_front();
00452     // Notify the clients waiting for some room to write in the pipe
00453     read_done.notify_all();
00454     /* ...and process the next reservation to see if it is ready to
00455        be released too */
00456 }
00457 }
00458
00459
00460 /** Process the write reservations that are ready to be released in the
00461     reservation queue
00462 */
00463 void move_write_reservation_forward() {
00464     // Lock the pipe to avoid nuisance
00465     std::lock_guard<std::mutex> lg { cb_mutex };
00466
00467     for (;;) {
00468         if (w_rid_q.empty())
00469             // No pending reservation, so nothing to do
00470             break;
00471         // Get the first reservation
00472         const auto &rid = w_rid_q.front();
00473         if (!rid.ready)
00474             /* If the reservation is not ready to be released, stop
00475                because it is blocking all the following in the queue
00476                anyway */
00477             break;
00478         // Remove the reservation to be released from the queue
00479         w_rid_q.pop_front();
00480         // Notify the clients waiting to read something from the pipe
00481         write_done.notify_all();
00482         /* ...and process the next reservation to see if it is ready to
00483            be released too */
00484     }
00485 }
00486
00487 };
00488
00489 /// @} End the execution Doxygen group
00490
00491 }
00492 }
00493 }
00494
00495 /*
00496  # Some Emacs stuff:
00497  ### Local Variables:
00498  ###  ispell-local-dictionary: "american"
00499  ###  eval: (flyspell-prog-mode)
00500  ###  End:
00501 */
00502
00503 #endif // TRISYCL_SYCL_PIPE_DETAIL_PIPE_HPP

```

## 11.123 include/CL/sycl/pipe.hpp File Reference

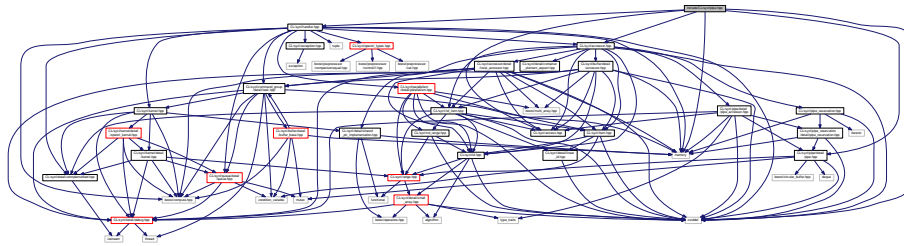
```

#include <cstdint>
#include <memory>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/handler.hpp"

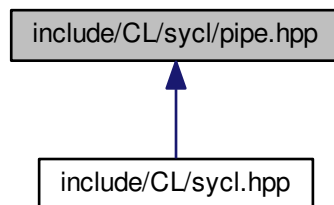
```

```
#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`

## 11.124 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 <cstdint>
```

```

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 = typename
00041     pipe::shared_ptr_implementation;
00042
00043     // Allows the comparison operation to access the implementation
00044     friend implementation_t;
00045
00046 public:
00047
00048     // Make the implementation member directly accessible in this class
00049     using implementation_t::implementation;
00050
00051     /// The STL-like types
00052     /* Since a pipe element cannot be directly addressed without
00053        accessor, only define value_type here */
00053     using value_type = T;
00054
00055
00056     /// Construct a pipe able to store up to capacity T objects
00057     pipe(std::size_t capacity)
00058         : implementation_t { new detail::pipe<T> { capacity } } {}
00059
00060
00061     /** Get an accessor to the pipe with the required mode
00062
00063         \param Mode is the requested access mode
00064
00065         \param Target is the type of pipe access required
00066
00067         \param[in] command_group_handler is the command group handler in
00068            which the kernel is to be executed
00069     */
00070     template <access::mode Mode,
00071             access::target Target = access::target::pipe>
00072     accessor<value_type, 1, Mode, Target>
00073     get_access(handler &command_group_handler) {
00074         static_assert(Target == access::target::pipe
00075             || Target == access::target::blocking_pipe,
00076             "get_access(handler) with pipes can only deal with "
00077             "access::pipe or access::blocking_pipe");
00078         return { implementation, command_group_handler };
00079     }
00080
00081
00082     /// Return the maximum number of elements that can fit in the pipe
00083     std::size_t capacity() const {
00084         return implementation->capacity();
00085     }
00086 };
00087
00088
00089 /// @} End the execution Doxygen group
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_PIPE_HPP

```

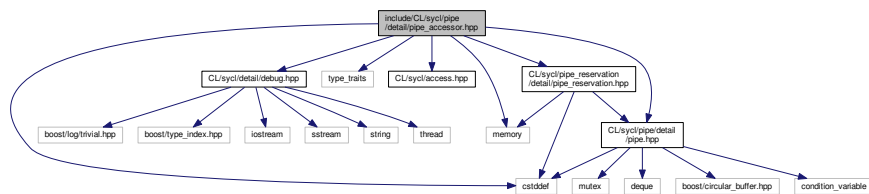
## 11.125 include/CL/sycl/pipe/detail/pipe\_accessor.hpp File Reference

```

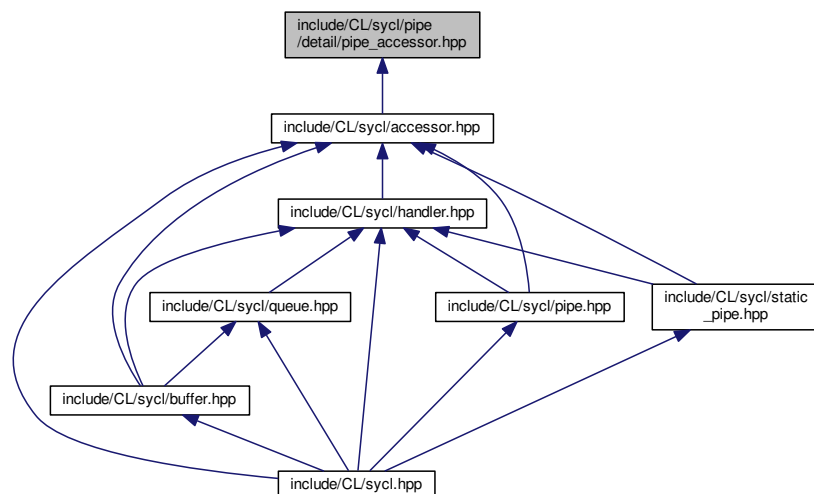
#include <cstdint>
#include <memory>
#include <type_traits>
#include "CL/sycl/access.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/pipe/detail/pipe.hpp"
#include "CL/sycl/pipe_reservation/detail/pipe_reservation.hpp"

```

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](#)

## 11.126 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 <cstdlib>
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                 int 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     pipe_accessor() = default;
00103
00104
00105     /// Return the maximum number of elements that can fit in the pipe
00106     std::size_t capacity() const {
00107         return implementation->capacity();
00108     }
00109
00110     /** Get the current number of elements in the pipe
00111
00112         This is obviously a volatile value which is constrained by
00113         restricted relativity.
00114
00115         Note that on some devices it may be costly to implement (for
00116         example on FPGA).
00117     */
00118     std::size_t size() const {
00119         return implementation->size_with_lock();
00120     }
00121
00122     /** Test if the pipe is empty
00123
00124         This is obviously a volatile value which is constrained by
00125         restricted relativity.
00126
00127         Note that on some devices it may be costly to implement on the
00128         write side (for example on FPGA).
00129     */
00130     bool empty() const {
00131         return implementation->empty_with_lock();
00132     }
00133
00134     /** Test if the pipe is full
00135
00136         This is obviously a volatile value which is constrained by
00137         restricted relativity.
00138
00139         Note that on some devices it may be costly to implement on the
00140         read side (for example on FPGA).
00141     */
00142     bool full() const {
00143         return implementation->full_with_lock();
00144     }
00145
00146     /** In an explicit bool context, the accessor gives the success
00147     status of the last access
00148
00149         It is not impacted by reservation success.
00150
00151         The explicitness is related to avoid \code some_pipe <<
00152         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

```

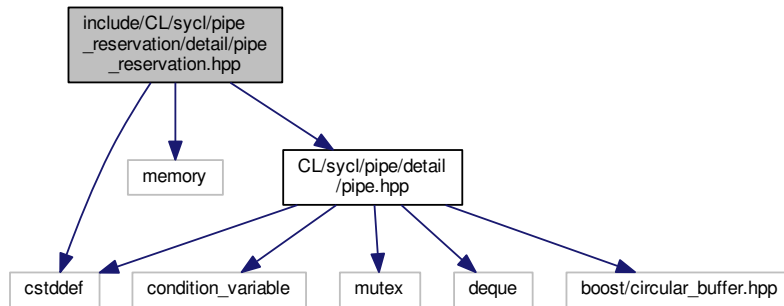
## 11.127 include/CL/sycl/pipe\_reservation/detail/pipe\_reservation.hpp File Reference

```

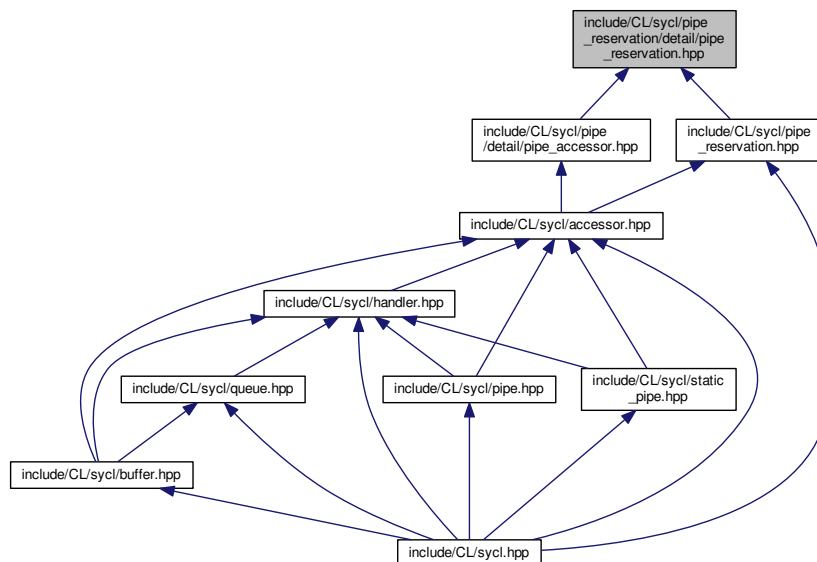
#include <cstdint>
#include <memory>
#include "CL/sycl/pipe/detail/pipe.hpp"

```

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`

## 11.128 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 <cstddef>
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               int 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
00160     return rid->start[index];
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

```

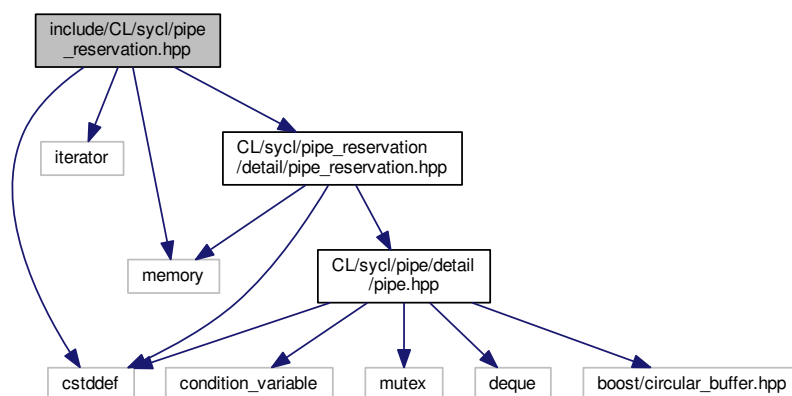
## 11.129 include/CL/sycl/pipe\_reservation.hpp File Reference

```

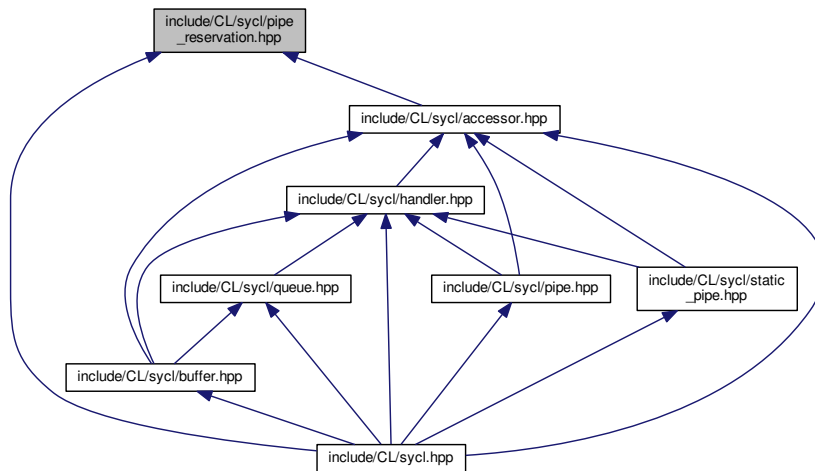
#include <cstddef>
#include <iterator>
#include <memory>
#include "CL/sycl/pipe_reservation/detail/pipe_reservation.hpp"

```

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`

## 11.130 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

```

## 11.131 include/CL/sycl/platform/detail/host\_platform.hpp File Reference

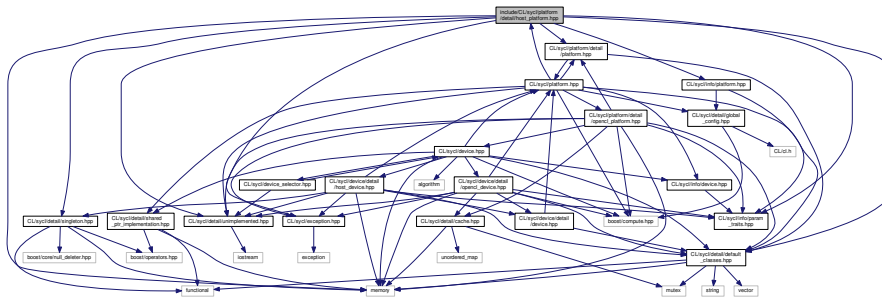
```

#include <memory>
#include "CL/sycl/detail/default_classes.hpp"

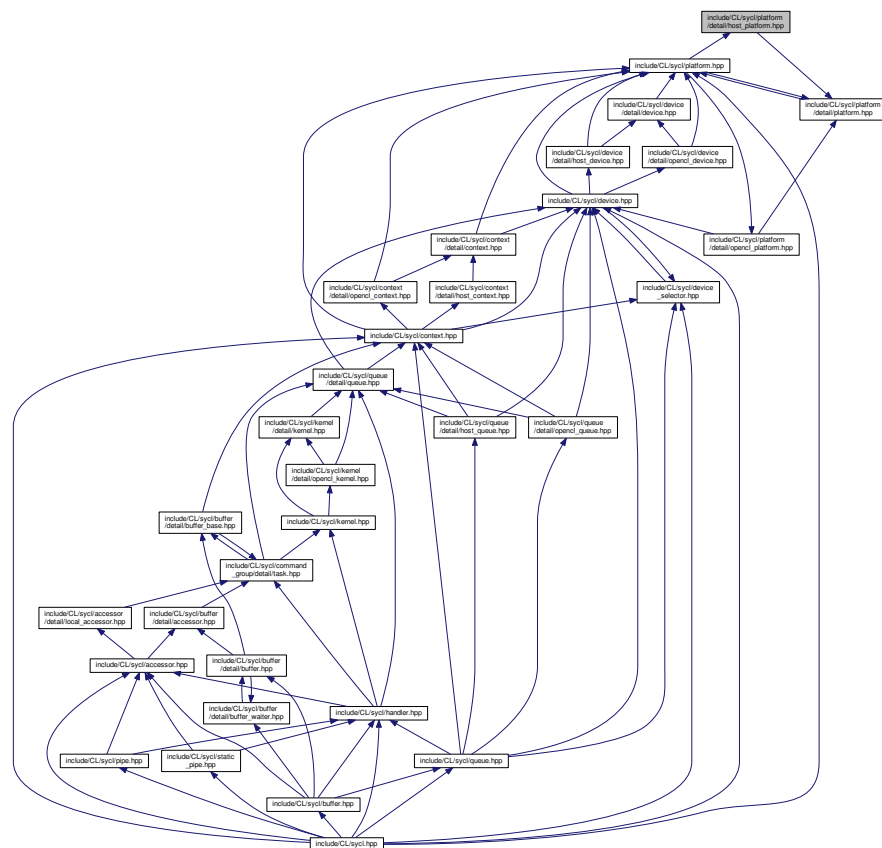
```



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](#)

## 11.132 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 #ifdef 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
00049     /** Return the underlying Boost.Compute platform
00050
00051         This throws an error since there is no Boost Compute platform associated
00052         to the host platform.
00053     */
00054     boost::compute::platform &get_boost_compute() const override {
00055         throw
00056             non_cl_error("The host device has no underlying Boost Compute platform");
00057     }
00058 #endif
00059
00060     /// Return true since this platform is the SYCL host platform
00061     bool is_host() const override {
00062         return true;
00063     }
00064
00065     /** Returning the information parameters for the host platform
00066         implementation
00067     */
00068
00069
00070

```

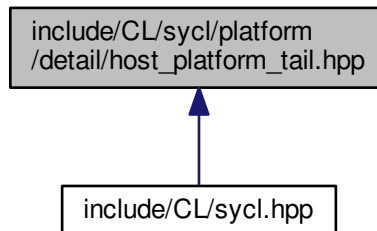
```

00071  string_class get_info_string(info::platform param) const
00072  override {
00073      switch (param) {
00074          case info::platform::profile:
00075              /* Well... Is the host platform really a full profile whereas it
00076               is not really OpenCL? */
00077              return "FULL_PROFILE";
00078          case info::platform::version:
00079              // \todo I guess it should include the software version too...
00080              return "2.2";
00081          case info::platform::name:
00082              return "triSYCL host platform";
00083          case info::platform::vendor:
00084              return "triSYCL Open Source project";
00085          case info::platform::extensions:
00086              return platform_extensions;
00087          default:
00088              // \todo Define some SYCL exception type for this type of errors
00089              throw std::invalid_argument {
00090                  "Unknown parameter value for SYCL platform information" };
00091      }
00092  }
00093  /** Specify whether a specific extension is supported on the platform
00094   \todo To be implemented
00095   */
00096  bool has_extension(const string_class &extension) const override {
00097      detail::unimplemented();
00098      return {};
00099  }
00100  /** Get all the available devices for the host platform
00101   \param[in] device_type is the device type to filter the selection
00102   or \c info::device_type::all by default to return all the
00103   devices
00104   \return the device list
00105   */
00106  vector_class<cl::sycl::device>
00107  get_devices(info::device_type device_type) const override;
00108  };
00109  /// @} to end the execution Doxygen group
00110  }
00111  }
00112  }
00113  /*
00114   # Some Emacs stuff:
00115   ### Local Variables:
00116   ### ispell-local-dictionary: "american"
00117   ### eval: (flyspell-prog-mode)
00118   ### End:
00119   */
00120  #endif // TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_HPP

```

## 11.133 include/CL/sycl/platform/detail/host\_platform\_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](#)
- [cl::sycl::detail](#)

## 11.134 host\_platform\_tail.hpp

```

00001 #ifndef TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_TAIL_HPP
00002 #define TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_TAIL_HPP
00003
00004 /** \file The ending part of the SYCL host platform
00005
00006     This is here to break a dependency between platform and device
00007
00008     a-doumoulakis at gmail dot com
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 execution Platforms, contexts, devices and queues
00019     @{
00020 */
00021
00022 /** Get all the available devices for this platform
00023
00024     \param[in] device_type is the device type to filter the selection
00025     or \c info::device_type::all by default to return all the
00026     devices
00027
00028     \return the device list
00029 */
00030 vector_class<cl::sycl::device>
00031 inline host_platform::get_devices(info::device_type device_type)
00032     const {
00033     /** If \c get_devices is called with the host platform
00034         and the right device type, returns the host_device.
00035     */
00036     if (device_type_selector { device_type } (cl::sycl::device {}) > 0)
  
```

```

00036     // Return 1 default device, i.e. the host device
00037     return { {} };
00038 else
00039     // No matching device
00040     return {};
00041 }
00042
00043 /// @} to end the Doxygen group
00044
00045 }
00046 }
00047 }
00048
00049 /*
00050 # Some Emacs stuff:
00051 ### Local Variables:
00052 ### ispell-local-dictionary: "american"
00053 ### eval: (flyspell-prog-mode)
00054 ### End:
00055 */
00056
00057 #endif // TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_TAIL_HPP

```

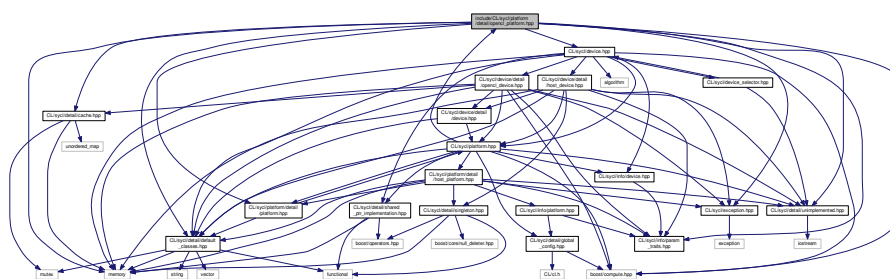
## 11.135 include/CL/sycl/platform/detail/opencil\_platform.hpp File Reference

```

#include <memory>
#include <boost/compute.hpp>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform/detail/platform.hpp"

```

Include dependency graph for opencil\_platform.hpp:





## 11.136 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>
                cache;
00047
00048         public:
00049
00050             /// Return the cl_platform_id of the underlying OpenCL platform
00051             cl_platform_id get() const override {
00052                 return p.id();
00053             }
00054
00055             /// Return the underlying Boost.Compute platform
00056             const boost::compute::platform &get_boost_compute() const
                override {
00057                 return p;
00058             }
00059
00060             /// Return false since an OpenCL platform is not the SYCL host platform
00061             bool is_host() const override {
00062                 return false;
00063             }
00064
00065             /// Returning the information string parameters for the OpenCL platform
00066             string_class get_info_string(info::platform param) const
                override {
00067                 /* Use the fact that the trisycl info values are the same as the
00068                     OpenCL ones used in Boost.Compute to just cast the enum class
00069                     to the int value */
00070                 return p.get_info<std::string>(static_cast<cl_platform_info>(param));
00071             }
00072
00073             /// Specify whether a specific extension is supported on the platform
00074             bool has_extension(const string_class &extension) const override {
00075                 return p.supports_extension(extension);
00076             }
00077
00078             bool is_host() const override {
00079                 return false;
00080             }
00081

```

```

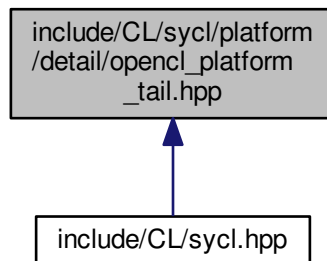
00082
00083     ///// Get a singleton instance of the opencil_platform
00084     static std::shared_ptr<opencil_platform>
00085     instance(const boost::compute::platform &p) {
00086         return cache.get_or_register(p.id(),
00087                                     [&] { return new opencil_platform { p }; });
00088     }
00089
00090
00091     /** Get all the available devices for this OpenCL platform
00092
00093         \param[in] device_type is the device type to filter the selection
00094         or \c info::device_type::all by default to return all the
00095         devices
00096
00097         \return the device list
00098     */
00099     vector_class<cl::sycl::device>
00100     get_devices(info::device_type device_type) const override;
00101
00102 private:
00103
00104     /// Only the instance factory can built it
00105     opencil_platform(const boost::compute::platform &p) : p { p } {}
00106
00107 public:
00108
00109     /// Unregister from the cache on destruction
00110     ~opencil_platform() override {
00111         cache.remove(p.id());
00112     }
00113
00114 };
00115
00116 /* Allocate the cache here but since this is a pure-header library,
00117    use a weak symbol so that only one remains when SYCL headers are
00118    used in different compilation units of a program
00119 */
00120 TRISYCL_WEAK_ATTRIB_PREFIX
00121 detail::cache<cl_platform_id, detail::opencil_platform>
00122     opencil_platform::cache
00123 TRISYCL_WEAK_ATTRIB_SUFFIX;
00124
00124 /// @} to end the execution Doxygen group
00125
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

```



## 11.137 include/CL/sycl/platform/detail/opencil\_platform\_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](#)
- [cl::sycl::detail](#)

## 11.138 opencil\_platform\_tail.hpp

```

00001 #ifndef TRISYCL_SYCL_PLATFORM_DETAIL_OPENCIL_PLATFORM_TAIL_HPP
00002 #define TRISYCL_SYCL_PLATFORM_DETAIL_OPENCIL_PLATFORM_TAIL_HPP
00003
00004 /** \file The ending part of the SYCL host platform
00005
00006     This is here to break a dependency between platform and device
00007
00008     a-doumoulakis at gmail dot com
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 execution Platforms, contexts, devices and queues
00019                 @{
00020             */
00021
00022             /** Returns a vector class containing all SYCL devices associated with
00023                 this OpenCL platform
00024
00025                 \param[in] device_type is the device type to filter the selection
00026                 or \c info::device_type::all by default to return all the
00027                 devices
00028
00029                 \return the device list
00030             */
00031             vector_class<cl::sycl::device>
00032             inline opencil_platform::get_devices(
00033                 info::device_type device_type) const {
00034                 vector_class<cl::sycl::device> devices;
00035                 device_type_selector ds { device_type };

```

```

00035 // Add the desired OpenCL devices
00036 for (const auto &d : get_boost_compute().devices()) {
00037     // Get the SYCL device from the Boost Compute device
00038     cl::sycl::device sycl_dev { d };
00039     /* Return the devices with the good criterion according to the selector.
00040      By calling devices on the \c boost::compute::platform we know that
00041      we iterate only over the device belonging to the current platform,
00042      */
00043     if (ds(sycl_dev) > 0)
00044         devices.push_back(sycl_dev);
00045 }
00046
00047 return devices;
00048 }
00049
00050 /// @} to end the Doxygen group
00051
00052 }
00053 }
00054 }
00055
00056 /*
00057     # Some Emacs stuff:
00058     ### Local Variables:
00059     ### ispell-local-dictionary: "american"
00060     ### eval: (flyspell-prog-mode)
00061     ### End:
00062 */
00063
00064 #endif // TRISYCL_SYCL_PLATFORM_DETAIL_OPENCL_PLATFORM_TAIL_HPP

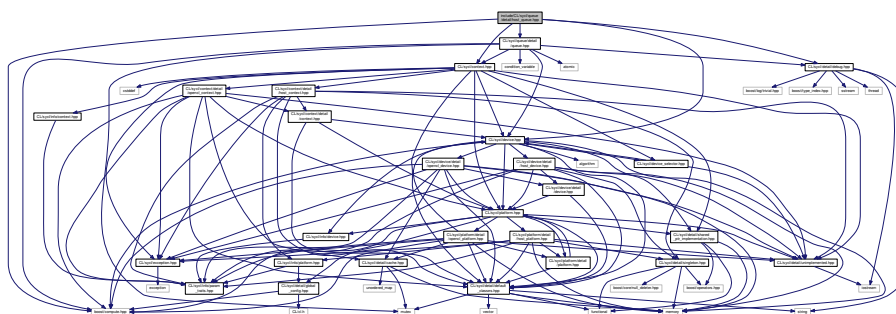
```

## 11.139 include/CL/sycl/queue/detail/host\_queue.hpp File Reference

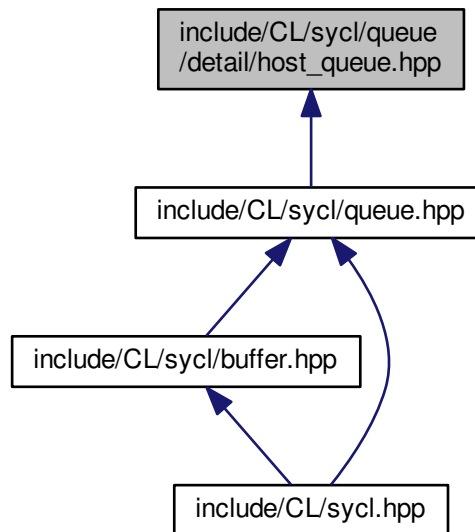
```

#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`

## 11.140 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 #ifdef 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     Note that a host queue is not a singleton, compared to host
00028     device or host platform, for example.
00029 */
00030 class host_queue : public detail::queue,
00031                   detail::debug<host_queue> {
00032
00033 #ifndef TRISYCL_OPENCL
00034     /** Return the cl_command_queue of the underlying OpenCL queue
00035
00036     This throws an error since there is no OpenCL queue associated
00037     to the host queue.
00038     */
00039     cl_command_queue get() const override {
00040         throw non_cl_error("The host queue has no OpenCL command queue");
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     */
00049     boost::compute::command_queue &get_boost_compute() override {
00050         throw non_cl_error("The host queue has no OpenCL command queue");
00051     }
00052 #endif
00053
00054     /// Return the SYCL host queue's host context
00056     cl::sycl::context get_context() const override {
00057         // Return the default context which is the host context
00058         return {};
00059     }
00060
00061     /// Return the SYCL host device the host queue is associated with
00063     cl::sycl::device get_device() const override {
00064         // Return the default device which is the host device
00065         return {};
00066     }
00067
00068     /// Claim proudly that the queue is executing on the SYCL host device
00070     bool is_host() const override {
00071         return true;
00072     }
00073
00074 };
00075
00076
00077 }
00078 }
00079 }
00080
00081 /*
00082     # Some Emacs stuff:
00083     ### Local Variables:
00084     ### ispell-local-dictionary: "american"
00085     ### eval: (flyspell-prog-mode)
00086     ### End:
00087 */
00088
00089 #endif // TRISYCL_SYCL_QUEUE_DETAIL_HOST_QUEUE_HPP

```

## 11.141 include/CL/sycl/queue/detail/opencv\_queue.hpp File Reference

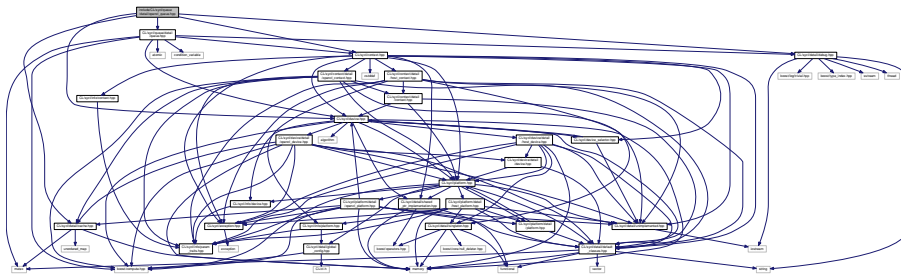
```

#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/device.hpp"

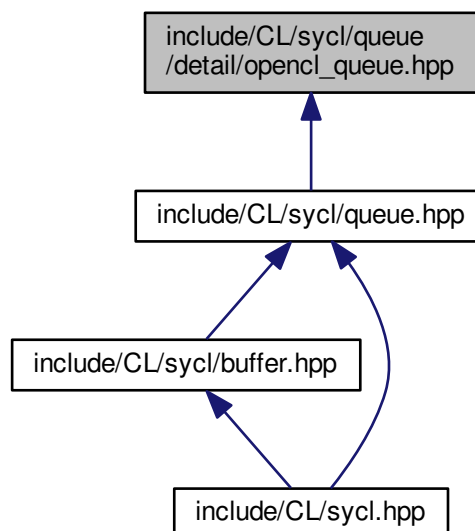
```

```
#include "CL/sycl/queue/detail/queue.hpp"
```

Include dependency graph for opencil\_queue.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [cl::sycl::detail::opencil\\_queue](#)  
*Some implementation details about the SYCL queue.*

## Namespaces

- [cl](#)  
*The vector type to be used as SYCL vector.*
- [cl::sycl](#)
- [cl::sycl::detail](#)

## 11.142 opencil\_queue.hpp

```

00001 #ifndef TRISYCL_SYCL_QUEUE_DETAIL_OPENCCL_QUEUE_HPP
00002 #define TRISYCL_SYCL_QUEUE_DETAIL_OPENCCL_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     cl::sycl::context get_context() const override {
00048         return q.get_context();
00049     }
00050
00051     // Return the SYCL device associated to the queue
00052     cl::sycl::device get_device() const override {
00053         return q.get_device();
00054     }
00055
00056     // Claim proudly that an OpenCL queue cannot be the SYCL host queue
00057     bool is_host() const override {
00058         return false;
00059     }
00060
00061 private:
00062     // Only the instance factory can build it
00063     opencil_queue(const boost::compute::command_queue &q) : q { q } {}
00064
00065 public:
00066     // Get a singleton instance of the opencil_queue
00067     static std::shared_ptr<opencil_queue>
00068     instance(const boost::compute::command_queue &q) {
00069         return cache.get_or_register(q.get(),
00070                                     [&] { return new opencil_queue { q }; });
00071     }
00072
00073     /** Create a new queue associated to this device
00074
00075         \todo Check with SYCL committee what is the expected behaviour
00076         here about the context. Is this a new context everytime, or
00077         always the same for a given device?

```

```

00084  */
00085  static std::shared_ptr<detail::queue>
00086  instance(const cl::sycl::device &d) {
00087      return instance (boost::compute::command_queue {
00088          // For now, create a new context every time
00089          boost::compute::context { d.get_boost_compute() },
00090          d.get_boost_compute()
00091      });
00092  }
00093
00094
00095  /// Unregister from the cache on destruction
00096  ~opencil_queue() override {
00097      cache.remove(q.get());
00098  }
00099
00100 };
00101
00102 /* Allocate the cache here but since this is a pure-header library,
00103    use a weak symbol so that only one remains when SYCL headers are
00104    used in different compilation units of a program
00105 */
00106 TRISYCL_WEAK_ATTRIB_PREFIX
00107 detail::cache<cl_command_queue, detail::opencil_queue>
00108 opencil_queue::cache
00109 TRISYCL_WEAK_ATTRIB_SUFFIX;
00110
00111 }
00112
00113
00114 /*
00115     # Some Emacs stuff:
00116     ### Local Variables:
00117     ### ispell-local-dictionary: "american"
00118     ### eval: (flyspell-prog-mode)
00119     ### End:
00120 */
00121
00122 #endif // TRISYCL_SYCL_QUEUE_DETAIL_OPENCIL_QUEUE_HPP

```

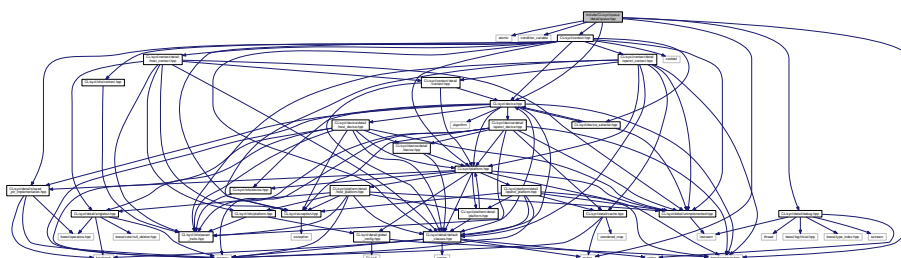
## 11.143 include/CL/sycl/queue/detail/queue.hpp File Reference

```

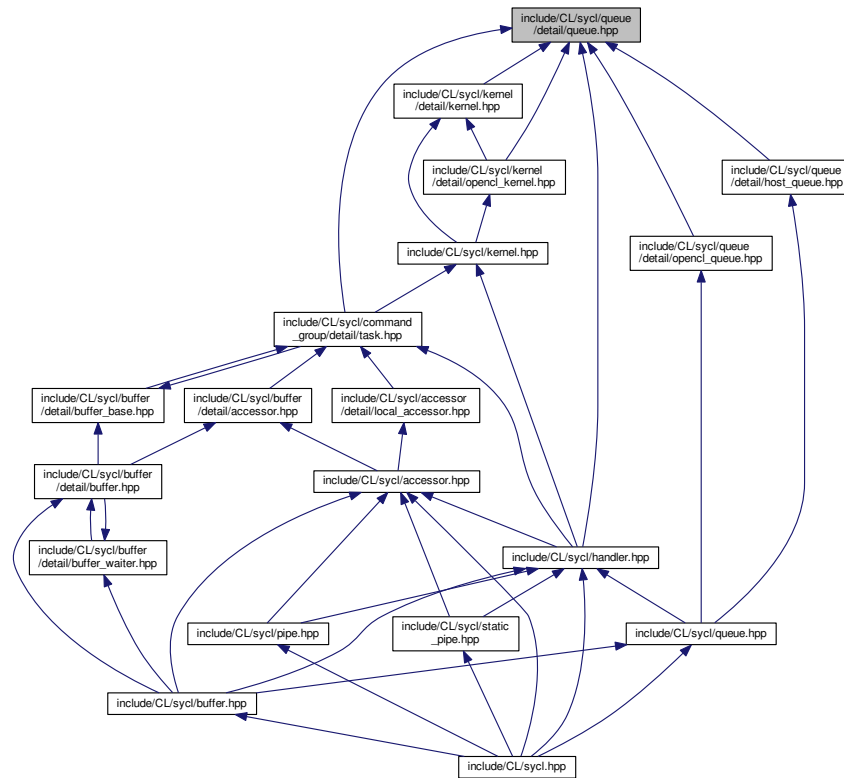
#include <atomic>
#include <condition_variable>
#include <mutex>
#include <boost/compute.hpp>
#include "CL/sycl/context.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/detail/debug.hpp"

```

Include dependency graph for queue.hpp:



This graph shows which files directly or indirectly include this file:



## 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`

## 11.144 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
00072                          In some cases several threads might want to wait for the
00073                          same queue, because of this \c notify_one is not be enough
00074                          and a \c notify_all is needed
00075                          */
00076                         finished.notify_all();
00077                     }
00078                 }
00079
00080
00081             #ifndef TRISYCL_OPENCL
00082                 /** Return the underlying OpenCL command queue after doing a retain
00083
00084                     This memory object is expected to be released by the developer.
00085
00086                     Retain a reference to the returned cl_command_queue object.
00087
00088                     Caller should release it when finished.
00089
00090                     If the queue is a SYCL host queue then an exception is thrown.
00091                 */
00092                 virtual cl_command_queue get() const = 0;
00093
00094                 /// Return the underlying Boost.Compute command queue
00095                 virtual boost::compute::command_queue &get_boost_compute() = 0;
00096             #endif
00097         }
00098     }
}

```

```

00099  /** Return the SYCL queue's context
00100
00101      Report errors using SYCL exception classes.
00102  */
00103  virtual cl::sycl::context get_context() const = 0;
00104
00105
00106  /** Return the SYCL device the queue is associated with
00107
00108      Report errors using SYCL exception classes.
00109  */
00110  virtual cl::sycl::device get_device() const = 0;
00111
00112
00113  /// Return whether the queue is executing on a SYCL host device
00114  virtual bool is_host() const = 0;
00115
00116
00117  /// Wait for all kernel completion before the queue destruction
00118  /// \todo Update according spec since queue destruction is non blocking
00119  virtual ~queue() {
00120      wait_for_kernel_execution();
00121  }
00122
00123 };
00124
00125 }
00126 }
00127 }
00128
00129 /*
00130     # Some Emacs stuff:
00131     ### Local Variables:
00132     ### ispell-local-dictionary: "american"
00133     ### eval: (flyspell-prog-mode)
00134     ### End:
00135 */
00136
00137 #endif // TRISYCL_SYCL_QUEUE_DETAIL_QUEUE_HPP

```

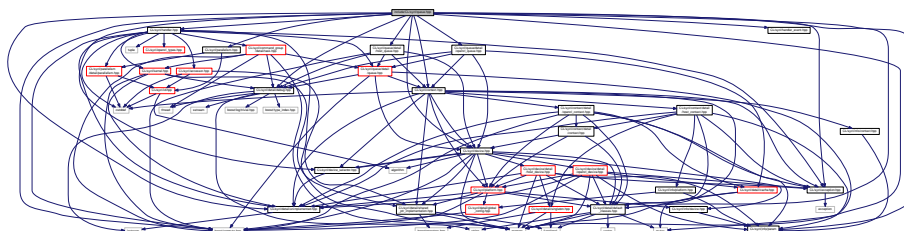
## 11.145 include/CL/sycl/queue.hpp File Reference

```

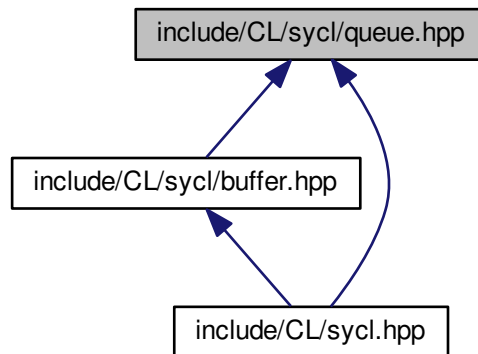
#include <memory>
#include <boost/compute.hpp>
#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/exception.hpp"
#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.

## 11.146 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/opencl_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     {
00081     public:
00082         /** Use the underlying queue implementation that can be shared in
00083             the SYCL model */
00084         : public detail::shared_ptr_implementation<queue, detail::queue>,

```

```

00084     detail::debug<queue> {
00085
00086     // The type encapsulating the implementation
00087     using implementation_t = typename
queue::shared_ptr_implementation;
00088
00089     /* Allows the comparison operation to sneak in
00090
00091     Required from Clang++ 3.9 and G++ 6
00092     */
00093     friend implementation_t;
00094
00095 public:
00096
00097     // Make the implementation member directly accessible in this class
00098     using implementation_t::implementation;
00099
00100     /** Default constructor for platform which is the host platform
00101
00102     Returns errors via the SYCL exception class.
00103
00104     \todo Check with the specification if it is the host queue or
00105     the one related to the default device selector.
00106     */
00107     queue() : implementation_t { new detail::host_queue } {}
00108
00109
00110     /** This constructor creates a SYCL queue from an OpenCL queue
00111
00112     At construction it does a retain on the queue memory object.
00113
00114     Retain a reference to the cl_command_queue object. Caller should
00115     release the passed cl_command_queue object when it is no longer
00116     needed.
00117
00118     Return synchronous errors regarding the creation of the queue and
00119     report asynchronous errors via the async_handler callback function
00120     in conjunction with the synchronization and throw methods.
00121
00122     Note that the default case asyncHandler = nullptr is handled by the
00123     default constructor.
00124
00125     */
00126     explicit queue(async_handler asyncHandler) : queue { } {
00127         detail::unimplemented();
00128     }
00129
00130
00131     /** Creates a queue for the device provided by the device selector
00132
00133     If no device is selected, an error is reported.
00134
00135     Return synchronous errors regarding the creation of the queue and
00136     report asynchronous errors via the \c async_handler callback
00137     function if and only if there is an \c async_handler provided.
00138     */
00139     queue(const device_selector &deviceSelector,
00140           async_handler asyncHandler = nullptr)
00141         // Just create the queue from the selected device
00142         : queue { device { deviceSelector }, asyncHandler } { }
00143
00144
00145     /** A queue is created for a SYCL device
00146
00147     Return asynchronous errors via the \c async_handler callback
00148     function.
00149     */
00150     queue(const device &d,
00151           async_handler asyncHandler = nullptr) : implementation_t {
00152 #ifdef TRISYCL_OPENCL
00153         d.is_host()
00154         ? std::shared_ptr<detail::queue>{ new detail::host_queue }
00155         : detail::opencl_queue::instance(d)
00156 #else
00157         new detail::host_queue
00158 #endif
00159     } { }
00160
00161
00162
00163     /** This constructor chooses a device based on the provided
00164     device_selector, which needs to be in the given context.
00165
00166     If no device is selected, an error is reported.
00167
00168     Return synchronous errors regarding the creation of the queue.
00169

```

```

00170         If and only if there is an asyncHandler provided, it reports
00171         asynchronous errors via the async_handler callback function in
00172         conjunction with the synchronization and throw methods.
00173     */
00174     queue(const context &syclContext,
00175           const device_selector &deviceSelector,
00176           async_handler asyncHandler = nullptr) : queue { } {
00177         detail::unimplemented();
00178     }
00179
00180
00181     /** Creates a command queue using clCreateCommandQueue from a context
00182         and a device
00183
00184         Return synchronous errors regarding the creation of the queue.
00185
00186         If 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           async_handler asyncHandler = nullptr) : queue { } {
00193         detail::unimplemented();
00194     }
00195
00196
00197     /** Creates a command queue using clCreateCommandQueue from a context
00198         and a device
00199
00200         It enables profiling on the queue if the profilingFlag is set to
00201         true.
00202
00203         Return synchronous errors regarding the creation of the queue. If
00204         and only if there is an asyncHandler provided, it reports
00205         asynchronous errors via the async_handler callback function in
00206         conjunction with the synchronization and throw methods.
00207     */
00208     queue(const context &syclContext,
00209           const device &syclDevice,
00210           info::queue_profiling profilingFlag,
00211           async_handler asyncHandler = nullptr) : queue { } {
00212         detail::unimplemented();
00213     }
00214
00215
00216 #ifndef TRISYCL_OPENCL
00217     /** This constructor creates a SYCL queue from an OpenCL queue
00218
00219         At construction it does a retain on the queue memory object.
00220
00221         Return synchronous errors regarding the creation of the queue. If
00222         and only if there is an async_handler provided, it reports
00223         asynchronous errors via the async_handler callback function in
00224         conjunction with the synchronization and throw methods.
00225     */
00226     queue(const cl_command_queue &q, async_handler ah = nullptr)
00227         : queue { boost::compute::command_queue { q }, ah } {}
00228
00229
00230     /** Construct a queue instance using a boost::compute::command_queue
00231
00232         This is a triSYCL extension for boost::compute interoperation.
00233
00234         Return synchronous errors via the SYCL exception class.
00235
00236         \todo Deal with handler
00237     */
00238     queue(const boost::compute::command_queue &q, async_handler ah = nullptr)
00239         : implementation_t { detail::openccl_queue::instance(q) }
00240     {}
00241 #endif
00242
00243 #ifndef TRISYCL_OPENCL
00244     /** Return the underlying OpenCL command queue after doing a retain
00245
00246         This memory object is expected to be released by the developer.
00247
00248         Retain a reference to the returned cl_command_queue object.
00249
00250         Caller should release it when finished.
00251
00252         If the queue is a SYCL host queue then an exception is thrown.
00253     */
00254     cl_command_queue get() const {
00255         return implementation->get();

```

```

00256     }
00257
00258
00259     /** Return the underlying Boost.Compute command queue if it is an
00260         OpenCL queue
00261
00262         This is a triSYCL extension
00263     */
00264     boost::compute::command_queue get_boost_compute() const {
00265         return implementation->get_boost_compute();
00266     }
00267 #endif
00268
00269
00270     /** Return the SYCL queue's context
00271
00272         Report errors using SYCL exception classes.
00273     */
00274     context get_context() const {
00275         return implementation->get_context();
00276     }
00277
00278
00279     /** Return the SYCL device the queue is associated with
00280
00281         Report errors using SYCL exception classes.
00282     */
00283     device get_device() const {
00284         return implementation->get_device();
00285     }
00286
00287
00288     /// Return whether the queue is executing on a SYCL host device
00289     bool is_host() const {
00290         return implementation->is_host();
00291     }
00292
00293
00294     /** Performs a blocking wait for the completion all enqueued tasks in
00295         the queue
00296
00297         Synchronous errors will be reported through SYCL exceptions.
00298     */
00299     void wait() {
00300         implementation->wait_for_kernel_execution();
00301     }
00302
00303
00304     /** Perform a blocking wait for the completion all enqueued tasks in the queue
00305
00306         Synchronous errors will be reported via SYCL exceptions.
00307
00308         Asynchronous errors will be passed to the async_handler passed to the
00309         queue on construction.
00310
00311         If no async_handler was provided then asynchronous exceptions will
00312         be lost.
00313     */
00314     void wait_and_throw() {
00315         // \todo Implement the throw part of wait_and_throw
00316         wait();
00317         detail::unimplemented();
00318     }
00319
00320
00321     /** Checks to see if any asynchronous errors have been produced by the
00322         queue and if so reports them by passing them to the async_handler
00323         passed to the queue on construction
00324
00325         If no async_handler was provided then asynchronous exceptions will
00326         be lost.
00327     */
00328     void throw_asynchronous() {
00329         detail::unimplemented();
00330     }
00331
00332
00333     /// Queries the platform for cl_command_queue info
00334     template <info::queue param>
00335     typename info::param_traits<info::queue, param>::type
00336     get_info() const {
00337         detail::unimplemented();
00338         return {};
00339     }
00340
00341     /** Submit a command group functor to the queue, in order to be

```

```

00342     scheduled for execution on the device
00343
00344     Use an explicit functor parameter taking a handler& so we can use
00345     "auto" in submit() lambda parameter.
00346
00347     \todo Add in the spec an implicit conversion of handler_event to
00348     queue& so it is possible to chain operations on the queue
00349
00350     \todo Update the spec to replace std::function by a templated
00351     type to avoid memory allocation
00352 */
00353 handler_event submit(std::function<void(handler &)> cgf) {
00354     handler command_group_handler { implementation };
00355     cgf(command_group_handler);
00356     return {};
00357 }
00358
00359
00360 /** Submit a command group functor to the queue, in order to be
00361     scheduled for execution on the device
00362
00363     On kernel error, this command group functor, then it is scheduled
00364     for execution on the secondary queue.
00365
00366     Return a command group functor event, which is corresponds to the
00367     queue the command group functor is being enqueued on.
00368 */
00369 handler_event submit(std::function<void(handler &)> cgf,
00370 queue &secondaryQueue) {
00371     detail::unimplemented();
00372     // Since it is not implemented, always submit on the main queue
00373     return submit(cgf);
00374 }
00375 };
00376
00377 /// @} to end the execution Doxygen group
00378
00379 }
00380 }
00381
00382 /* Inject a custom specialization of std::hash to have the buffer
00383     usable into an unordered associative container
00384
00385     \todo Add this to the spec
00386 */
00387 namespace std {
00388
00389 template <> struct hash<cl::sycl::queue> {
00390
00391     auto operator()(const cl::sycl::queue &q) const {
00392         // Forward the hashing to the implementation
00393         return q.hash();
00394     }
00395 };
00396 };
00397
00398 }
00399
00400 /*
00401     # Some Emacs stuff:
00402     ### Local Variables:
00403     ###  ispell-local-dictionary: "american"
00404     ###  eval: (flyspell-prog-mode)
00405     ###  End:
00406 */
00407
00408 #endif // TRISYCL_SYCL_QUEUE_HPP

```

## 11.147 include/CL/sycl/range.hpp File Reference

```

#include <functional>
#include <numeric>
#include "CL/sycl/detail/small_array.hpp"

```



```

graph TD
    A[include/CL/sycl/range.hpp] --> B[functional]
    A --> C[numeric]
    A --> D[CL/sycl/detail/smallest_array.hpp]
    D --> E[algorithm]
    D --> F[array]
    D --> G[cstddef]
    D --> H[type_traits]
    D --> I[boost/operators.hpp]
    D --> J[CL/sycl/detail/debug.hpp]
    J --> K[iostream]
    J --> L[sstream]
    J --> M[string]
    J --> N[thread]
    J --> O[boost/log/trivial.hpp]
    J --> P[boost/type_index.hpp]
  
```

[illegible]

- class `cl::sycl::range< Dimensions >`

## Namespaces

- ## Functions

- Generated by Doxygen

## 11.148 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 Dimensions instead of int Dimensions in the
00027     specification?
00028
00029     \todo add to the specification this default parameter value?
00030
00031     \todo add to the specification some way to specify an offset?
00032 */
00033 template <int Dimensions = 1>
00034 class range : public detail::small_array_123<
00035     std::size_t,
00036     range<Dimensions>,
00037     Dimensions > {
00038
00039 public:
00040
00041     // Inherit of all the constructors
00042     using detail::small_array_123<std::size_t,
00043         range<Dimensions>,
00044         Dimensions>::small_array_123;
00045
00046
00047     /** Return the number of elements in the range
00048
00049         \todo Give back size() its real meaning in the specification
00050
00051         \todo add this method to the specification
00052     */
00053     size_t get_count() const {
00054         // Return the product of the sizes in each dimension
00055         return std::accumulate(this->cbegin(),
00056             this->cend(),
00057             1,
00058             std::multiplies<size_t> {});
00059     }
00060 };
00061
00062
00063 /** Implement a make_range to construct a range<> of the right dimension
00064     with implicit conversion from an initializer list for example.
00065
00066     Cannot use a template on the number of dimensions because the implicit
00067     conversion would not be tried.
00068 */
00069 inline auto make_range(range<1> r) { return r; }
00070 inline auto make_range(range<2> r) { return r; }
00071 inline auto make_range(range<3> r) { return r; }
00072
00073
00074 /** Construct a range<> from a function call with arguments, like
00075     make_range(1, 2, 3)
00076 */
00077 template<typename... BasicType>
00078 auto make_range(BasicType... Args) {
00079     // Call constructor directly to allow narrowing
00080     return range<sizeof...(Args)>(Args...);
00081 }
00082
00083 /// @} End the parallelism Doxygen group
00084

```

```

00085 }
00086 }
00087
00088 /*
00089  # Some Emacs stuff:
00090  ### Local Variables:
00091  ### ispell-local-dictionary: "american"
00092  ### eval: (flyspell-prog-mode)
00093  ### End:
00094 */
00095
00096 #endif // TRISYCL_SYCL_RANGE_HPP

```

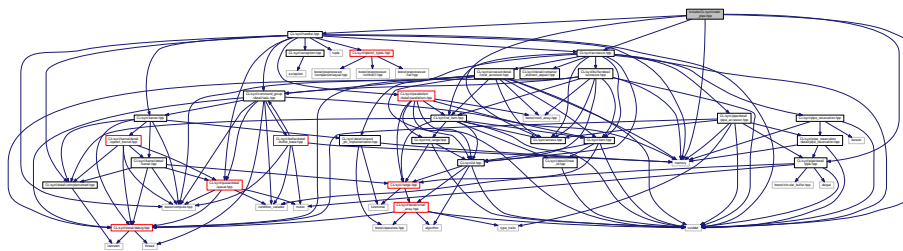
## 11.149 include/CL/sycl/static\_pipe.hpp File Reference

```

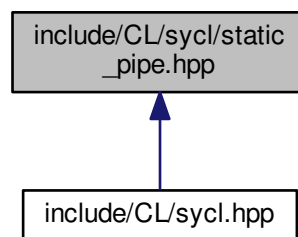
#include <cstddef>
#include <memory>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/pipe/detail/pipe.hpp"

```

Include dependency graph for static\_pipe.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class `cl::sycl::static_pipe< T, Capacity >`

A SYCL static-scoped pipe equivalent to an OpenCL program-scoped pipe. [More...](#)

## Namespaces

- [cl](#)  
*The vector type to be used as SYCL vector.*
- [cl::sycl](#)

## 11.150 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 <cstdint>
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     /* Use the underlying pipe implementation that can be shared in
00052     the SYCL model */
00053     : public detail::shared_ptr_implementation<static_pipe<T, Capacity>,
00054         detail::pipe<T>>,
00055         detail::debug<static_pipe<T, Capacity>> {
00056
00057     // The type encapsulating the implementation
00058     using implementation_t = typename
00059         static_pipe::shared_ptr_implementation;
00060
00061     // Make the implementation member directly accessible in this class
00062     using implementation_t::implementation;
00063
00064     // Allows the comparison operation to access the implementation
00065     friend implementation_t;
00066
00067 public:
00068
00069     /// The STL-like types
00070     using value_type = T;
00071

```

```

00071
00072 /// Construct a static-scoped pipe able to store up to Capacity T objects
00073 static_pipe()
00074 : implementation_t { new detail::pipe<T> { Capacity } } { }
00075
00076
00077 /** Get an accessor to the pipe with the required mode
00078
00079     \param Mode is the requested access mode
00080
00081     \param Target is the type of pipe access required
00082
00083     \param[in] command_group_handler is the command group handler in
00084         which the kernel is to be executed
00085 */
00086 template <access::mode Mode,
00087           access::target Target = access::target::pipe>
00088 accessor<value_type, 1, Mode, Target>
00089 get_access(handler &command_group_handler) {
00090     static_assert(Target == access::target::pipe
00091                 || Target == access::target::blocking_pipe,
00092                 "get_access(handler) with pipes can only deal with "
00093                 "access::pipe or access::blocking_pipe");
00094     return { implementation, command_group_handler };
00095 }
00096
00097
00098 /** Return the maximum number of elements that can fit in the pipe
00099
00100     This is a constexpr since the capacity is in the type.
00101 */
00102 std::size_t constexpr capacity() const {
00103     return Capacity;
00104 }
00105
00106 };
00107
00108 /// @} End the execution Doxygen group
00109
00110 }
00111 }
00112
00113 /*
00114     # Some Emacs stuff:
00115     ### Local Variables:
00116     ### ispell-local-dictionary: "american"
00117     ### eval: (flyspell-prog-mode)
00118     ### End:
00119 */
00120
00121 #endif // TRISYCL_SYCL_STATIC_PIPE_HPP

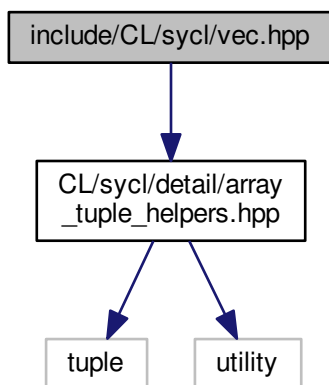
```

## 11.151 include/CL/sycl/vec.hpp File Reference

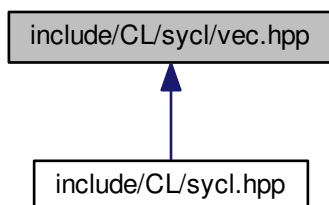
Implement the small OpenCL vector class.

```
#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 `actual_type=unsigned char`, `uchar4` is equivalent to `vec<unsigned char, 4>`*
- `#define TRISYCL_DEFINE_VEC_TYPE(type, actual_type)`  
*Declare the vector types of a type for all the sizes.*

### 11.151.1 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](#).

## 11.152 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 basic_type::basic_type;
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
00156     /** A macro to define type alias, such as for type=uchar, size=4 and
00157         actual_type=unsigned char, uchar4 is equivalent to vec<unsigned char, 4>
00158     */
00159 #define TRISYCL_DEFINE_VEC_TYPE_SIZE(type, size, actual_type) \
00160     using type##size = vec<actual_type, size>;
00161
00162     /// Declare the vector types of a type for all the sizes
00163 #define TRISYCL_DEFINE_VEC_TYPE(type, actual_type) \
00164     TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 1, actual_type) \
00165     TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 2, actual_type) \
00166     TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 3, actual_type) \
00167     TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 4, actual_type) \
00168     TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 8, actual_type) \
00169     TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 16, actual_type)
00170
00171     /// Declare all the possible vector type aliases
00172     TRISYCL_DEFINE_VEC_TYPE(char, char)
00173     TRISYCL_DEFINE_VEC_TYPE(uchar, unsigned char)
00174     TRISYCL_DEFINE_VEC_TYPE(short, short int)
00175     TRISYCL_DEFINE_VEC_TYPE(ushort, unsigned short int)
00176     TRISYCL_DEFINE_VEC_TYPE(int, int)
00177     TRISYCL_DEFINE_VEC_TYPE(uint, unsigned int)
00178     TRISYCL_DEFINE_VEC_TYPE(long, long int)
00179     TRISYCL_DEFINE_VEC_TYPE(ulong, unsigned long int)
00180     TRISYCL_DEFINE_VEC_TYPE(float, float)
00181     TRISYCL_DEFINE_VEC_TYPE(double, double)
00182
00183     /// @} End the vector Doxygen group
00184
00185
00186 }
00187 }
00188
00189 /*
00190     # Some Emacs stuff:
00191     ### Local Variables:
00192     ### ispell-local-dictionary: "american"
00193     ### eval: (flyspell-prog-mode)
00194     ### End:
00195 */
00196
00197 #endif // TRISYCL_SYCL_VEC_HPP

```



# Index

- `__SYCL_SINGLE_SOURCE__`
  - Manage default configuration and types, [366](#)
- `~accessor`
  - `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >`, [35](#)
- `~buffer`
  - `cl::sycl::detail::buffer`, [98](#)
- `~buffer_base`
  - `cl::sycl::detail::buffer_base`, [111](#)
- `~buffer_waiter`
  - `cl::sycl::detail::buffer_waiter`, [122](#)
- `~context`
  - `cl::sycl::detail::context`, [236](#)
- `~device`
  - `cl::sycl::detail::device`, [253](#)
- `~device_selector`
  - `cl::sycl::device_selector`, [271](#)
- `~error_handler`
  - `cl::sycl::error_handler`, [370](#)
- `~opencl_context`
  - `cl::sycl::detail::opencl_context`, [504](#)
- `~opencl_device`
  - `cl::sycl::detail::opencl_device`, [511](#)
- `~opencl_platform`
  - `cl::sycl::detail::opencl_platform`, [294](#)
- `~opencl_queue`
  - `cl::sycl::detail::opencl_queue`, [525](#)
- `~pipe_accessor`
  - `cl::sycl::detail::pipe_accessor`, [161](#)
- `~pipe_reservation`
  - `cl::sycl::detail::pipe_reservation`, [176](#)
- `~platform`
  - `cl::sycl::detail::platform`, [298](#)
- `~queue`
  - `cl::sycl::detail::queue`, [531](#)
- `accelerator`
  - Platforms, contexts, devices and queues, [327](#)
- `access`
  - `cl::sycl::detail::buffer`, [107](#)
- `accessor`
  - `cl::sycl::accessor`, [51–53](#)
  - `cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >`, [68](#)
  - `cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >`, [65](#)
  - `cl::sycl::detail::accessor`, [75, 76](#)
  - `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >`, [35](#)
- `accessor_detail`
  - `cl::sycl::accessor`, [50](#)
  - `cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >`, [68](#)
  - `cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >`, [65](#)
  - `cl::sycl::pipe_reservation`, [185](#)
- `accessor_type`
  - `cl::sycl::detail::pipe_reservation`, [174](#)
  - `cl::sycl::pipe_reservation`, [185](#)
- `add_buffer`
  - `cl::sycl::detail::task`, [546](#)
- `add_buffer_to_task`
  - `cl::sycl::detail`, [475](#)
  - Data access and storage in SYCL, [200](#)
- `add_postlude`
  - `cl::sycl::detail::task`, [546](#)
- `add_prelude`
  - `cl::sycl::detail::task`, [547](#)
- `add_to_task`
  - `cl::sycl::detail::buffer_base`, [112](#)
- `addr_space`
  - Dealing with OpenCL address spaces, [226](#)
- `address_space`
  - `cl::sycl::detail::address_space_base`, [222](#)
  - Dealing with OpenCL address spaces, [230](#)
- `address_space_array`
  - `cl::sycl::detail::address_space_array`, [210](#)
- `address_space_fundamental`
  - `cl::sycl::detail::address_space_fundamental`, [213](#)
- `address_space_object`
  - `cl::sycl::detail::address_space_object`, [215](#)
- `address_space_ptr`
  - `cl::sycl::detail::address_space_ptr`, [219](#)
- `address_space_variable`
  - `cl::sycl::detail::address_space_variable`, [224, 225](#)
- `all`
  - Platforms, contexts, devices and queues, [327](#)
- `alloc`
  - `cl::sycl::detail::buffer`, [108](#)
- `allocate_accessor`
  - `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >`, [36](#)
- `allocate_buffer`
  - `cl::sycl::detail::buffer`, [99](#)
- `allocation`
  - `cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >`, [47](#)
  - `cl::sycl::detail::buffer`, [108](#)
- `allocator_type`

- cl::sycl::buffer, 126
- array
  - cl::sycl::detail::accessor, 90
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 47
- array\_type
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 33
- array\_view\_type
  - cl::sycl::detail::accessor, 73
- assume\_validity
  - cl::sycl::detail::pipe\_reservation, 177
- async\_handler
  - Error handling, 394
- barrier
  - cl::sycl::nd\_item, 419
- basic\_type
  - cl::sycl::vec, 457
- begin
  - cl::sycl::accessor, 53
  - cl::sycl::detail::accessor, 77
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 36
  - cl::sycl::detail::pipe\_reservation, 178
  - cl::sycl::pipe\_reservation, 188
- blocking
  - cl::sycl::detail::pipe\_accessor, 167
  - cl::sycl::detail::pipe\_reservation, 182
  - cl::sycl::pipe\_reservation, 194
- buf
  - cl::sycl::detail::accessor, 90
- buffer
  - cl::sycl::buffer, 127–133
  - cl::sycl::detail::buffer, 95–97
- buffer\_add\_to\_task
  - Data access and storage in SYCL, 200
- buffer\_allocator
  - Data access and storage in SYCL, 199
- buffer\_base
  - cl::sycl::detail::buffer\_base, 111
- buffer\_cache
  - cl::sycl::detail::buffer\_base, 119
- buffer\_waiter
  - cl::sycl::detail::buffer\_waiter, 122
- buffers\_in\_use
  - cl::sycl::detail::task, 553
- c
  - cl::sycl::detail::cache, 483
  - cl::sycl::detail::opencl\_context, 508
- CL\_SYCL\_LANGUAGE\_VERSION
  - Manage default configuration and types, 366
- cache
  - cl::sycl::detail::opencl\_context, 509
  - cl::sycl::detail::opencl\_device, 516
  - cl::sycl::detail::opencl\_kernel, 522
  - cl::sycl::detail::opencl\_platform, 297
  - cl::sycl::detail::opencl\_queue, 528
- call\_update\_buffer\_state
  - cl::sycl::detail::buffer, 100, 101
- capacity
  - cl::sycl::detail::pipe, 147
  - cl::sycl::detail::pipe\_accessor, 161
  - cl::sycl::pipe, 171
  - cl::sycl::static\_pipe, 197
- cb
  - cl::sycl::detail::pipe, 156
- cb\_mutex
  - cl::sycl::detail::pipe, 156
- cbegin
  - cl::sycl::accessor, 54
  - cl::sycl::detail::accessor, 77
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 37
  - cl::sycl::pipe\_reservation, 189
- cend
  - cl::sycl::accessor, 54
  - cl::sycl::detail::accessor, 78
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 38
  - cl::sycl::pipe\_reservation, 189
- cl, 461
- cl::sycl, 461
  - function\_class, 466
  - min, 467
  - mutex\_class, 466
  - shared\_ptr\_class, 466
  - string\_class, 466
  - TRISYCL\_MATH\_WRAP2s, 468
  - TRISYCL\_MATH\_WRAP, 467
  - unique\_ptr\_class, 467
  - vector\_class, 467
  - weak\_ptr\_class, 467
  - y, 469
  - z, 469
- cl::sycl::access, 469
  - fence\_space, 469
  - mode, 470
  - target, 470
- cl::sycl::accessor, 48
  - accessor, 51–53
  - accessor\_detail, 50
  - begin, 53
  - cbegin, 54
  - cend, 54
  - crbegin, 55
  - crend, 55
  - dimensionality, 63
  - end, 55
  - get\_count, 56
  - get\_pointer, 56
  - get\_range, 56
  - get\_size, 57
  - implementation\_t, 50, 63
  - operator\*, 57, 58
  - operator[], 58–61

- rbegin, 62
  - rend, 62
- cl::sycl::accessor< DataType, 1, AccessMode, access\_mode::target::blocking\_pipe >, 66
  - accessor, 68
  - accessor\_detail, 68
  - get\_pipe\_detail, 68
  - reserve, 69
- cl::sycl::accessor< DataType, 1, AccessMode, access\_mode::target::pipe >, 63
  - accessor, 65
  - accessor\_detail, 65
  - get\_pipe\_detail, 65
  - reserve, 66
- cl::sycl::accessor\_error, 379
- cl::sycl::async\_exception, 376
- cl::sycl::buffer, 123
  - allocator\_type, 126
  - buffer, 127–133
  - const\_reference, 126
  - get\_access, 134, 135
  - get\_count, 136
  - get\_range, 136
  - get\_size, 136
  - implementation\_t, 126, 141
  - is\_cached, 137
  - is\_data\_up\_to\_date, 137
  - is\_read\_only, 137
  - mark\_as\_written, 138
  - reference, 126
  - set\_final\_data, 138–140
  - use\_count, 140
  - value\_type, 127
- cl::sycl::cl\_exception, 374
  - cl\_code, 376
  - cl\_exception, 375
  - get\_cl\_code, 375
- cl::sycl::compile\_program\_error, 384
- cl::sycl::context, 241
  - context, 244–248
  - get, 249
  - get\_boost\_compute, 249
  - get\_boost\_queue, 249
  - get\_devices, 250
  - get\_info, 250
  - get\_platform, 251
  - implementation\_t, 244
  - is\_host, 251
- cl::sycl::detail, 471
  - add\_buffer\_to\_task, 475
- cl::sycl::detail::accessor, 69
  - accessor, 75, 76
  - array, 90
  - array\_view\_type, 73
  - begin, 77
  - buf, 90
  - cbegin, 77
  - cend, 78
  - const\_iterator, 73
  - const\_reference, 73
  - const\_reverse\_iterator, 73
  - copy\_back\_cl\_buffer, 78
  - copy\_in\_cl\_buffer, 78
  - crbegin, 79
  - crend, 79
  - dimensionality, 91
  - element, 74
  - end, 79
  - get\_buffer, 80
  - get\_cl\_buffer, 80
  - get\_count, 81
  - get\_pointer, 81
  - get\_range, 82
  - get\_size, 82
  - handler, 91
  - is\_read\_access, 83
  - is\_write\_access, 84
  - iterator, 74
  - operator\*, 84, 85
  - operator[], 85–88
  - rbegin, 88
  - reference, 74
  - register\_accessor, 89
  - rend, 89
  - reverse\_iterator, 74
  - task, 91
  - value\_type, 74
  - writable\_array\_view\_type, 75
- cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 30
  - ~accessor, 35
  - accessor, 35
  - allocate\_accessor, 36
  - allocation, 47
  - array, 47
  - array\_type, 33
  - begin, 36
  - cbegin, 37
  - cend, 38
  - const\_iterator, 33
  - const\_reference, 33
  - const\_reverse\_iterator, 34
  - crbegin, 38
  - crend, 38
  - deallocate\_accessor, 38
  - dimensionality, 47
  - element, 34
  - end, 39
  - get\_count, 39
  - get\_range, 39
  - get\_size, 40
  - handler, 48
  - is\_read\_access, 40
  - is\_write\_access, 41
  - iterator, 34
  - operator\*, 41, 42

- operator[], 42–45
- rbegin, 46
- reference, 34
- rend, 46
- reverse\_iterator, 34
- value\_type, 35
- writable\_array\_type, 35
- cl::sycl::detail::address\_space\_array, 207
  - address\_space\_array, 210
  - super, 209
- cl::sycl::detail::address\_space\_base, 219
  - address\_space, 222
  - opengl\_type, 222
  - type, 222
- cl::sycl::detail::address\_space\_fundamental, 210
  - address\_space\_fundamental, 213
  - super, 212
- cl::sycl::detail::address\_space\_object, 213
  - address\_space\_object, 215
  - opengl\_type, 215
  - operator opengl\_type &, 216
- cl::sycl::detail::address\_space\_ptr, 216
  - address\_space\_ptr, 219
  - pointer\_t, 218
  - reference\_t, 219
  - super, 219
- cl::sycl::detail::address\_space\_variable, 222
  - address\_space\_variable, 224, 225
  - get\_address, 225
  - opengl\_type, 224
  - operator opengl\_type &, 225
  - super, 224
  - variable, 226
- cl::sycl::detail::buffer, 91
  - ~buffer, 98
  - access, 107
  - alloc, 108
  - allocate\_buffer, 99
  - allocation, 108
  - buffer, 95–97
  - call\_update\_buffer\_state, 100, 101
  - copy\_if\_modified, 108
  - data\_host, 108
  - deallocate\_buffer, 101
  - detail::accessor, 107
  - element, 95
  - final\_write\_back, 109
  - get\_count, 102
  - get\_destructor\_future, 102
  - get\_range, 103
  - get\_size, 104
  - input\_shared\_pointer, 109
  - mark\_as\_written, 104
  - modified, 109
  - non\_const\_value\_type, 95
  - set\_final\_data, 105, 106
  - track\_access\_mode, 106
  - value\_type, 95
- cl::sycl::detail::buffer\_base, 109
  - ~buffer\_base, 111
  - add\_to\_task, 112
  - buffer\_base, 111
  - buffer\_cache, 119
  - create\_in\_cache, 112
  - fresh\_ctx, 119
  - get\_cl\_buffer, 113
  - get\_latest\_producer, 113
  - is\_cached, 114
  - is\_data\_up\_to\_date, 114
  - latest\_producer, 119
  - latest\_producer\_mutex, 120
  - notify\_buffer\_destructor, 120
  - number\_of\_users, 120
  - ready, 120
  - ready\_mutex, 120
  - release, 114
  - set\_latest\_producer, 115
  - sync\_with\_host, 115
  - update\_buffer\_state, 116
  - use, 118
  - wait, 118
- cl::sycl::detail::buffer\_waiter, 121
  - ~buffer\_waiter, 122
  - buffer\_waiter, 122
  - implementation\_t, 122, 123
- cl::sycl::detail::cache
  - c, 483
  - get\_or\_register, 481
  - key\_type, 480
  - m, 483
  - remove, 482
  - value\_type, 480
- cl::sycl::detail::cache< Key, Value >, 479
- cl::sycl::detail::container\_element\_aspect, 342
  - const\_pointer, 343
  - const\_reference, 343
  - pointer, 343
  - reference, 343
  - value\_type, 343
- cl::sycl::detail::context, 235
  - ~context, 236
  - get, 236
  - get\_boost\_compute, 236
  - get\_boost\_queue, 236
  - get\_devices, 237
  - get\_platform, 237
  - is\_host, 237
- cl::sycl::detail::debug, 363
- cl::sycl::detail::device, 252
  - ~device, 253
  - get, 253
  - get\_boost\_compute, 253
  - get\_platform, 253
  - has\_extension, 254
  - is\_accelerator, 254
  - is\_cpu, 254

- is\_gpu, 254
- is\_host, 254
- cl::sycl::detail::display\_vector, 363
  - display, 364
- cl::sycl::detail::expand\_to\_vector, 337
- cl::sycl::detail::expand\_to\_vector< V, Tuple, true >, 337
- cl::sycl::detail::host\_context, 237
  - get, 239
  - get\_boost\_compute, 239
  - get\_boost\_queue, 239
  - get\_devices, 240
  - get\_platform, 240
  - is\_host, 241
- cl::sycl::detail::host\_device, 493
  - get, 494
  - get\_boost\_compute, 494
  - get\_platform, 495
  - has\_extension, 495
  - is\_accelerator, 496
  - is\_cpu, 496
  - is\_gpu, 497
  - is\_host, 497
- cl::sycl::detail::host\_platform, 288
  - get, 290
  - get\_boost\_compute, 290
  - get\_info\_string, 290
  - has\_extension, 291
  - is\_host, 292
  - platform\_extensions, 292
- cl::sycl::detail::host\_queue, 498
  - get, 499
  - get\_boost\_compute, 499
  - get\_context, 500
  - get\_device, 500
  - is\_host, 500
- cl::sycl::detail::kernel, 282
  - get, 284
  - get\_boost\_compute, 284
  - single\_task, 284
  - TRISYCL\_ParallelForKernel\_RANGE, 284
- cl::sycl::detail::ocl\_type, 205
  - type, 205
- cl::sycl::detail::ocl\_type< T, constant\_address\_space >, 205
  - type, 205
- cl::sycl::detail::ocl\_type< T, generic\_address\_space >, 206
  - type, 206
- cl::sycl::detail::ocl\_type< T, global\_address\_space >, 206
  - type, 206
- cl::sycl::detail::ocl\_type< T, local\_address\_space >, 207
  - type, 207
- cl::sycl::detail::ocl\_type< T, private\_address\_space >, 207
  - type, 207
- cl::sycl::detail::opencl\_context, 502
  - ~opencl\_context, 504
  - c, 508
  - cache, 509
  - get, 504
  - get\_boost\_compute, 505
  - get\_boost\_queue, 505
  - get\_devices, 505
  - get\_platform, 506
  - instance, 507
  - is\_host, 508
  - opencl\_context, 504
  - q, 509
- cl::sycl::detail::opencl\_device, 510
  - ~opencl\_device, 511
  - cache, 516
  - d, 516
  - get, 512
  - get\_boost\_compute, 512
  - get\_platform, 513
  - has\_extension, 513
  - instance, 514
  - is\_accelerator, 515
  - is\_cpu, 515
  - is\_gpu, 515
  - is\_host, 516
  - opencl\_device, 511
- cl::sycl::detail::opencl\_kernel, 517
  - cache, 522
  - get, 519
  - get\_boost\_compute, 519
  - instance, 520
  - k, 522
  - opencl\_kernel, 519
  - single\_task, 521
  - TRISYCL\_ParallelForKernel\_RANGE, 521
- cl::sycl::detail::opencl\_platform, 292
  - ~opencl\_platform, 294
  - cache, 297
  - get, 295
  - get\_boost\_compute, 295
  - get\_info\_string, 295
  - has\_extension, 296
  - instance, 296
  - is\_host, 297
  - opencl\_platform, 294
  - p, 297
- cl::sycl::detail::opencl\_queue, 523
  - ~opencl\_queue, 525
  - cache, 528
  - get, 526
  - get\_boost\_compute, 526
  - get\_context, 526
  - get\_device, 526
  - instance, 527
  - is\_host, 528
  - opencl\_queue, 525
  - q, 529
- cl::sycl::detail::parallel\_OpenMP\_for\_iterate, 439

- parallel\_OpenMP\_for\_iterate, 439
- cl::sycl::detail::parallel\_for\_iterate, 438
  - parallel\_for\_iterate, 439
- cl::sycl::detail::parallel\_for\_iterate< 0, Range, Parallel←
  - ForFunctor, Id >, 440
  - parallel\_for\_iterate, 440
- cl::sycl::detail::pipe, 143
  - capacity, 147
  - cb, 156
  - cb\_mutex, 156
  - debug\_mode, 156
  - empty, 147
  - empty\_with\_lock, 147
  - full, 148
  - full\_with\_lock, 148
  - implementation\_t, 146
  - move\_read\_reservation\_forward, 148
  - move\_write\_reservation\_forward, 149
  - pipe, 146
  - r\_rid\_q, 156
  - read, 150
  - read\_done, 156
  - read\_reserved\_frozen, 157
  - reserve\_read, 151
  - reserve\_write, 152
  - reserved\_for\_reading, 153
  - reserved\_for\_writing, 153
  - rid\_iterator, 146
  - size, 154
  - size\_with\_lock, 154
  - used\_for\_reading, 157
  - used\_for\_writing, 157
  - value\_type, 146
  - w\_rid\_q, 157
  - write, 155
  - write\_done, 157
- cl::sycl::detail::pipe\_accessor, 158
  - ~pipe\_accessor, 161
  - blocking, 167
  - capacity, 161
  - const\_reference, 160
  - empty, 161
  - full, 162
  - get\_pipe\_detail, 162
  - implementation, 167
  - mode, 167
  - ok, 168
  - operator bool, 162
  - operator<<, 163
  - operator>>, 163
  - pipe\_accessor, 160
  - rank, 168
  - read, 164
  - reference, 160
  - reserve, 165
  - set\_debug, 165
  - size, 166
  - target, 168
  - value\_type, 160
  - write, 166
- cl::sycl::detail::pipe\_reservation, 172
  - ~pipe\_reservation, 176
  - accessor\_type, 174
  - assume\_validity, 177
  - begin, 178
  - blocking, 182
  - commit, 178
  - const\_iterator, 174
  - end, 179
  - iterator, 174
  - mode, 182
  - ok, 182
  - operator bool, 180
  - operator[], 180
  - p, 182
  - pipe\_reservation, 175, 176
  - reference, 174
  - rid, 182
  - size, 181
  - target, 183
  - value\_type, 174
- cl::sycl::detail::platform, 298
  - ~platform, 298
  - get, 299
  - get\_boost\_compute, 299
  - get\_devices, 299
  - get\_info\_string, 299
  - has\_extension, 300
  - is\_host, 300
- cl::sycl::detail::queue, 530
  - ~queue, 531
  - finished, 536
  - finished\_mutex, 536
  - get, 532
  - get\_boost\_compute, 532
  - get\_context, 533
  - get\_device, 533
  - is\_host, 534
  - kernel\_end, 534
  - kernel\_start, 535
  - queue, 531
  - running\_kernels, 536
  - wait\_for\_kernel\_execution, 535
- cl::sycl::detail::reserve\_id, 141
  - ready, 143
  - reserve\_id, 142
  - size, 143
  - start, 143
- cl::sycl::detail::shared\_ptr\_implementation
  - hash, 540
  - implementation, 541
  - operator<, 540
  - operator==, 541
  - shared\_ptr\_implementation, 539
- cl::sycl::detail::shared\_ptr\_implementation< Parent, Im-
  - plementation >, 537



- cl::sycl::detail::singleton
  - instance, [542](#)
- cl::sycl::detail::singleton< T >, [542](#)
- cl::sycl::detail::small\_array, [343](#)
  - dimension, [351](#)
  - dimensionality, [351](#)
  - element\_type, [346](#)
  - get, [348](#)
  - operator FinalType, [348](#)
  - small\_array, [346](#), [347](#)
  - x, [348](#)
  - y, [349](#)
  - z, [350](#)
- cl::sycl::detail::small\_array\_123, [351](#)
- cl::sycl::detail::small\_array\_123< BasicType, FinalType, 1 >, [352](#)
  - operator BasicType, [354](#)
  - small\_array\_123, [353](#), [354](#)
- cl::sycl::detail::small\_array\_123< BasicType, FinalType, 2 >, [354](#)
  - small\_array\_123, [355](#), [356](#)
- cl::sycl::detail::small\_array\_123< BasicType, FinalType, 3 >, [357](#)
  - small\_array\_123, [358](#), [359](#)
- cl::sycl::detail::task, [543](#)
  - add\_buffer, [546](#)
  - add\_postlude, [546](#)
  - add\_prelude, [547](#)
  - buffers\_in\_use, [553](#)
  - epilogues, [553](#)
  - execution\_ended, [554](#)
  - get\_kernel, [547](#)
  - get\_queue, [547](#)
  - kernel, [554](#)
  - notify\_consumers, [548](#)
  - owner\_queue, [554](#)
  - postlude, [548](#)
  - prelude, [549](#)
  - producer\_tasks, [554](#)
  - prologues, [554](#)
  - ready, [555](#)
  - ready\_mutex, [555](#)
  - release\_buffers, [549](#)
  - schedule, [550](#)
  - set\_kernel, [552](#)
  - task, [546](#)
  - wait, [552](#)
  - wait\_for\_producers, [553](#)
- cl::sycl::device, [255](#)
  - device, [257](#), [258](#)
  - get, [259](#)
  - get\_boost\_compute, [260](#)
  - get\_info, [260](#), [261](#)
  - get\_platform, [261](#)
  - has\_extension, [262](#)
  - implementation\_t, [257](#)
  - is\_accelerator, [262](#)
  - is\_cpu, [263](#)
  - is\_gpu, [263](#)
  - is\_host, [264](#)
  - type, [264](#)
- cl::sycl::device\_error, [383](#)
- cl::sycl::device\_selector, [270](#)
  - ~device\_selector, [271](#)
  - operator(), [272](#)
  - select\_device, [272](#)
- cl::sycl::device\_type\_selector, [265](#)
  - default\_device, [268](#)
  - device\_type, [268](#)
  - device\_type\_selector, [267](#)
  - operator(), [267](#)
- cl::sycl::device\_typename\_selector, [269](#)
  - device\_typename\_selector, [270](#)
- cl::sycl::error\_handler, [369](#)
  - ~error\_handler, [370](#)
  - default\_handler, [370](#)
  - report\_error, [370](#)
- cl::sycl::event, [485](#)
  - event, [485](#)
- cl::sycl::event\_error, [381](#)
- cl::sycl::exception, [371](#)
  - exception, [373](#)
  - message, [374](#)
  - what, [373](#)
- cl::sycl::exception\_list, [370](#)
- cl::sycl::feature\_not\_supported, [392](#)
- cl::sycl::group, [396](#)
  - dimensionality, [407](#)
  - get, [399](#)
  - get\_global\_range, [399](#), [400](#)
  - get\_group\_range, [400](#)
  - get\_linear, [401](#)
  - get\_local\_range, [401](#), [402](#)
  - get\_nd\_range, [402](#)
  - get\_offset, [403](#)
  - group, [398](#)
  - group\_id, [407](#)
  - ndr, [407](#)
  - operator[], [404](#)
  - parallel\_for\_work\_item, [405](#), [406](#)
- cl::sycl::handler, [272](#)
  - Dimensions, [282](#)
  - dispatch\_set\_arg, [275](#)
  - handler, [274](#)
  - parallel\_for, [275](#)
  - parallel\_for\_work\_group, [276](#), [277](#)
  - set\_arg, [278](#), [279](#)
  - set\_args, [279](#)
  - single\_task, [280](#), [281](#)
  - TRISYCL\_parallel\_for\_functor\_GLOBAL, [281](#)
  - task, [282](#)
- cl::sycl::id, [407](#)
  - id, [408](#)
- cl::sycl::image, [141](#)
- cl::sycl::info, [475](#)
  - queue, [477](#)

- queue\_profiling, 477
- cl::sycl::info::param\_traits< T, Param >, 529
- cl::sycl::invalid\_object\_error, 387
- cl::sycl::invalid\_parameter\_error, 382
- cl::sycl::is\_wrapper< T >, 501
- cl::sycl::item, 409
  - dimensionality, 415
  - display, 411
  - get, 411, 412
  - get\_linear\_id, 412
  - get\_offset, 413
  - get\_range, 414
  - global\_index, 415
  - global\_range, 415
  - item, 410
  - offset, 416
  - operator[], 414
  - set, 415
- cl::sycl::kernel, 284
  - get, 287
  - handler, 288
  - implementation\_t, 286, 288
  - kernel, 286, 287
- cl::sycl::kernel\_error, 378
- cl::sycl::link\_program\_error, 385
- cl::sycl::memory\_allocation\_error, 388
- cl::sycl::nd\_item, 416
  - barrier, 419
  - dimensionality, 433
  - get\_global, 420
  - get\_global\_linear\_id, 421
  - get\_global\_range, 422
  - get\_group, 423, 424
  - get\_group\_linear\_id, 425
  - get\_item, 425
  - get\_local, 426, 427
  - get\_local\_linear\_id, 428
  - get\_local\_range, 428
  - get\_nd\_range, 429
  - get\_num\_groups, 430, 431
  - get\_offset, 431
  - global\_index, 433
  - local\_index, 433
  - ND\_range, 433
  - nd\_item, 418
  - set\_global, 432
  - set\_local, 432
- cl::sycl::nd\_range, 433
  - dimensionality, 437
  - display, 435
  - get\_global, 435
  - get\_group, 436
  - get\_local, 436
  - get\_offset, 437
  - global\_range, 438
  - local\_range, 438
  - nd\_range, 435
  - offset, 438
- cl::sycl::nd\_range\_error, 380
- cl::sycl::non\_cl\_error, 393
- cl::sycl::pipe, 168
  - capacity, 171
  - get\_access, 171
  - implementation\_t, 170, 171
  - pipe, 170
  - value\_type, 170
- cl::sycl::pipe\_error, 389
- cl::sycl::pipe\_reservation, 183
  - accessor\_detail, 185
  - accessor\_type, 185
  - begin, 188
  - blocking, 194
  - cbegin, 189
  - cend, 189
  - commit, 190
  - const\_iterator, 185
  - const\_pointer, 185
  - const\_reference, 185
  - const\_reverse\_iterator, 186
  - crbegin, 190
  - crend, 191
  - difference\_type, 186
  - end, 191
  - implementation, 195
  - iterator, 186
  - operator bool, 192
  - operator[], 192
  - pipe\_reservation, 187, 188
  - pointer, 186
  - rbegin, 193
  - reference, 186
  - rend, 193
  - reverse\_iterator, 186
  - size, 194
  - size\_type, 187
  - value\_type, 187
- cl::sycl::platform, 300
  - get, 304
  - get\_boost\_compute, 304
  - get\_devices, 305
  - get\_info, 305, 306
  - get\_platforms, 306
  - has\_extension, 306
  - is\_host, 307
  - platform, 302, 303
  - shared\_ptr\_implementation, 307
- cl::sycl::platform\_error, 390
- cl::sycl::profiling\_error, 391
- cl::sycl::queue, 307
  - get, 315
  - get\_boost\_compute, 315
  - get\_context, 315
  - get\_device, 315
  - get\_info, 316
  - implementation\_t, 309, 319
  - is\_host, 316

- queue, 310–314
- submit, 316, 317
- throw\_asynchronous, 317
- wait, 318
- wait\_and\_throw, 318
- cl::sycl::range, 441
  - get\_count, 442
- cl::sycl::runtime\_error, 377
- cl::sycl::static\_pipe, 195
  - capacity, 197
  - get\_access, 198
  - implementation\_t, 197, 199
  - static\_pipe, 197
  - value\_type, 197
- cl::sycl::trisycl, 478
- cl::sycl::trisycl::default\_error\_handler, 484
  - report\_error, 485
- cl::sycl::vec, 455
  - basic\_type, 457
  - flatten, 458
  - flatten\_to\_tuple, 459
  - vec, 457, 458
- cl\_code
  - cl::sycl::cl\_exception, 376
- cl\_exception
  - cl::sycl::cl\_exception, 375
- commit
  - cl::sycl::detail::pipe\_reservation, 178
  - cl::sycl::pipe\_reservation, 190
- const\_iterator
  - cl::sycl::detail::accessor, 73
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 33
  - cl::sycl::detail::pipe\_reservation, 174
  - cl::sycl::pipe\_reservation, 185
- const\_pointer
  - cl::sycl::detail::container\_element\_aspect, 343
  - cl::sycl::pipe\_reservation, 185
- const\_reference
  - cl::sycl::buffer, 126
  - cl::sycl::detail::accessor, 73
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 33
  - cl::sycl::detail::container\_element\_aspect, 343
  - cl::sycl::detail::pipe\_accessor, 160
  - cl::sycl::pipe\_reservation, 185
- const\_reverse\_iterator
  - cl::sycl::detail::accessor, 73
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 34
  - cl::sycl::pipe\_reservation, 186
- constant
  - Dealing with OpenCL address spaces, 226
- constant\_ptr
  - Dealing with OpenCL address spaces, 227
- context
  - cl::sycl::context, 244–248
  - Platforms, contexts, devices and queues, 322
- copy\_back\_cl\_buffer
  - cl::sycl::detail::accessor, 78
- copy\_if\_modified
  - cl::sycl::detail::buffer, 108
- copy\_in\_cl\_buffer
  - cl::sycl::detail::accessor, 78
- cpu
  - Platforms, contexts, devices and queues, 327
- cpu\_selector
  - Platforms, contexts, devices and queues, 319
- crbegin
  - cl::sycl::accessor, 55
  - cl::sycl::detail::accessor, 79
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 38
  - cl::sycl::pipe\_reservation, 190
- create\_in\_cache
  - cl::sycl::detail::buffer\_base, 112
- crend
  - cl::sycl::accessor, 55
  - cl::sycl::detail::accessor, 79
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 38
  - cl::sycl::pipe\_reservation, 191
- custom
  - Platforms, contexts, devices and queues, 327
- d
  - cl::sycl::detail::opencl\_device, 516
- Data access and storage in SYCL, 29
  - add\_buffer\_to\_task, 200
  - buffer\_add\_to\_task, 200
  - buffer\_allocator, 199
  - get\_pipe\_detail, 201
  - image\_allocator, 199
  - map\_allocator, 200
  - waiter, 202
- data\_host
  - cl::sycl::detail::buffer, 108
- Dealing with OpenCL address spaces, 203
  - addr\_space, 226
  - address\_space, 230
  - constant, 226
  - constant\_ptr, 227
  - generic, 227
  - global, 227
  - global\_ptr, 228
  - local, 228
  - local\_ptr, 228
  - make\_multi, 231
  - multi\_ptr, 229
  - priv, 229
  - private\_ptr, 230
- deallocate\_accessor
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 38
- deallocate\_buffer
  - cl::sycl::detail::buffer, 101
- debug.hpp

- TRISYCL\_DUMP\_T, [648](#)
- TRISYCL\_DUMP, [648](#)
- TRISYCL\_INTERNAL\_DUMP, [648](#)
- debug\_mode
  - cl::sycl::detail::pipe, [156](#)
- Debugging and tracing support, [363](#)
  - trace\_kernel, [364](#)
- default\_device
  - cl::sycl::device\_type\_selector, [268](#)
- default\_handler
  - cl::sycl::error\_handler, [370](#)
- default\_selector
  - Platforms, contexts, devices and queues, [320](#)
- defaults
  - Platforms, contexts, devices and queues, [327](#)
- denorm
  - Platforms, contexts, devices and queues, [328](#)
- detail::accessor
  - cl::sycl::detail::buffer, [107](#)
- device
  - cl::sycl::device, [257](#), [258](#)
  - Platforms, contexts, devices and queues, [322](#)
- device::get\_info< info::device::device\_type >
  - Platforms, contexts, devices and queues, [330](#)
- device::get\_info< info::device::local\_mem\_size >
  - Platforms, contexts, devices and queues, [330](#)
- device::get\_info< info::device::max\_compute\_units >
  - Platforms, contexts, devices and queues, [331](#)
- device::get\_info< info::device::max\_mem\_alloc\_size >
  - Platforms, contexts, devices and queues, [331](#)
- device::get\_info< info::device::max\_work\_group\_size >
  - Platforms, contexts, devices and queues, [331](#)
- device::get\_info< info::device::name >
  - Platforms, contexts, devices and queues, [332](#)
- device::get\_info< info::device::profile >
  - Platforms, contexts, devices and queues, [332](#)
- device::get\_info< info::device::vendor >
  - Platforms, contexts, devices and queues, [332](#)
- device\_affinity\_domain
  - Platforms, contexts, devices and queues, [325](#)
- device\_exec\_capabilities
  - Platforms, contexts, devices and queues, [320](#)
- device\_execution\_capabilities
  - Platforms, contexts, devices and queues, [325](#)
- device\_fp\_config
  - Platforms, contexts, devices and queues, [320](#)
- device\_partition\_property
  - Platforms, contexts, devices and queues, [326](#)
- device\_partition\_type
  - Platforms, contexts, devices and queues, [326](#)
- device\_queue\_properties
  - Platforms, contexts, devices and queues, [321](#)
- device\_type
  - cl::sycl::device\_type\_selector, [268](#)
  - Platforms, contexts, devices and queues, [327](#)
- device\_type\_selector
  - cl::sycl::device\_type\_selector, [267](#)
- device\_type\_name\_selector
  - cl::sycl::device\_type\_name\_selector, [270](#)
- devices
  - Platforms, contexts, devices and queues, [322](#)
- difference\_type
  - cl::sycl::pipe\_reservation, [186](#)
- dimension
  - cl::sycl::detail::small\_array, [351](#)
- dimensionality
  - cl::sycl::accessor, [63](#)
  - cl::sycl::detail::accessor, [91](#)
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, [47](#)
  - cl::sycl::detail::small\_array, [351](#)
  - cl::sycl::group, [407](#)
  - cl::sycl::item, [415](#)
  - cl::sycl::nd\_item, [433](#)
  - cl::sycl::nd\_range, [437](#)
- Dimensions
  - cl::sycl::handler, [282](#)
- dispatch\_set\_arg
  - cl::sycl::handler, [275](#)
- display
  - cl::sycl::detail::display\_vector, [364](#)
  - cl::sycl::item, [411](#)
  - cl::sycl::nd\_range, [435](#)
- element
  - cl::sycl::detail::accessor, [74](#)
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, [34](#)
  - cl::sycl::detail::buffer, [95](#)
- element\_type
  - cl::sycl::detail::small\_array, [346](#)
- empty
  - cl::sycl::detail::pipe, [147](#)
  - cl::sycl::detail::pipe\_accessor, [161](#)
- empty\_with\_lock
  - cl::sycl::detail::pipe, [147](#)
- end
  - cl::sycl::accessor, [55](#)
  - cl::sycl::detail::accessor, [79](#)
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, [39](#)
  - cl::sycl::detail::pipe\_reservation, [179](#)
  - cl::sycl::pipe\_reservation, [191](#)
- epilogues
  - cl::sycl::detail::task, [553](#)
- Error handling, [368](#)
  - async\_handler, [394](#)
  - exception\_ptr, [394](#)
- event
  - cl::sycl::event, [485](#)
- exception
  - cl::sycl::exception, [373](#)
- exception\_ptr
  - Error handling, [394](#)
- execution\_ended
  - cl::sycl::detail::task, [554](#)
- expand

- Helpers to do array and tuple conversion, 338, 339
- Expressing parallelism through kernels, 395
  - make\_id, 443, 444
  - make\_range, 444, 445
  - parallel\_for, 445–448
  - parallel\_for\_global\_offset, 449
  - parallel\_for\_work\_item, 450
  - parallel\_for\_workgroup, 451
  - parallel\_for\_workitem, 452
- extensions
  - Platforms, contexts, devices and queues, 324
- fence\_space
  - cl::sycl::access, 469
- fill\_tuple
  - Helpers to do array and tuple conversion, 339
- final\_write\_back
  - cl::sycl::detail::buffer, 109
- finished
  - cl::sycl::detail::queue, 536
- finished\_mutex
  - cl::sycl::detail::queue, 536
- flatten
  - cl::sycl::vec, 458
- flatten\_to\_tuple
  - cl::sycl::vec, 459
- fma
  - Platforms, contexts, devices and queues, 328
- fp\_config
  - Platforms, contexts, devices and queues, 328
- fresh\_ctx
  - cl::sycl::detail::buffer\_base, 119
- full
  - cl::sycl::detail::pipe, 148
  - cl::sycl::detail::pipe\_accessor, 162
- full\_with\_lock
  - cl::sycl::detail::pipe, 148
- function\_class
  - cl::sycl, 466
- generic
  - Dealing with OpenCL address spaces, 227
- get
  - cl::sycl::context, 249
  - cl::sycl::detail::context, 236
  - cl::sycl::detail::device, 253
  - cl::sycl::detail::host\_context, 239
  - cl::sycl::detail::host\_device, 494
  - cl::sycl::detail::host\_platform, 290
  - cl::sycl::detail::host\_queue, 499
  - cl::sycl::detail::kernel, 284
  - cl::sycl::detail::opencl\_context, 504
  - cl::sycl::detail::opencl\_device, 512
  - cl::sycl::detail::opencl\_kernel, 519
  - cl::sycl::detail::opencl\_platform, 295
  - cl::sycl::detail::opencl\_queue, 526
  - cl::sycl::detail::platform, 299
  - cl::sycl::detail::queue, 532
  - cl::sycl::detail::small\_array, 348
  - cl::sycl::device, 259
  - cl::sycl::group, 399
  - cl::sycl::item, 411, 412
  - cl::sycl::kernel, 287
  - cl::sycl::platform, 304
  - cl::sycl::queue, 315
- get\_access
  - cl::sycl::buffer, 134, 135
  - cl::sycl::pipe, 171
  - cl::sycl::static\_pipe, 198
- get\_address
  - cl::sycl::detail::address\_space\_variable, 225
- get\_boost\_compute
  - cl::sycl::context, 249
  - cl::sycl::detail::context, 236
  - cl::sycl::detail::device, 253
  - cl::sycl::detail::host\_context, 239
  - cl::sycl::detail::host\_device, 494
  - cl::sycl::detail::host\_platform, 290
  - cl::sycl::detail::host\_queue, 499
  - cl::sycl::detail::kernel, 284
  - cl::sycl::detail::opencl\_context, 505
  - cl::sycl::detail::opencl\_device, 512
  - cl::sycl::detail::opencl\_kernel, 519
  - cl::sycl::detail::opencl\_platform, 295
  - cl::sycl::detail::opencl\_queue, 526
  - cl::sycl::detail::platform, 299
  - cl::sycl::detail::queue, 532
  - cl::sycl::device, 260
  - cl::sycl::platform, 304
  - cl::sycl::queue, 315
- get\_boost\_queue
  - cl::sycl::context, 249
  - cl::sycl::detail::context, 236
  - cl::sycl::detail::host\_context, 239
  - cl::sycl::detail::opencl\_context, 505
- get\_buffer
  - cl::sycl::detail::accessor, 80
- get\_cl\_buffer
  - cl::sycl::detail::accessor, 80
  - cl::sycl::detail::buffer\_base, 113
- get\_cl\_code
  - cl::sycl::cl\_exception, 375
- get\_context
  - cl::sycl::detail::host\_queue, 500
  - cl::sycl::detail::opencl\_queue, 526
  - cl::sycl::detail::queue, 533
  - cl::sycl::queue, 315
- get\_count
  - cl::sycl::accessor, 56
  - cl::sycl::buffer, 136
  - cl::sycl::detail::accessor, 81
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 39
  - cl::sycl::detail::buffer, 102
  - cl::sycl::range, 442
- get\_destructor\_future
  - cl::sycl::detail::buffer, 102

- get\_device
  - cl::sycl::detail::host\_queue, 500
  - cl::sycl::detail::opencl\_queue, 526
  - cl::sycl::detail::queue, 533
  - cl::sycl::queue, 315
- get\_devices
  - cl::sycl::context, 250
  - cl::sycl::detail::context, 237
  - cl::sycl::detail::host\_context, 240
  - cl::sycl::detail::opencl\_context, 505
  - cl::sycl::detail::platform, 299
  - cl::sycl::platform, 305
  - Platforms, contexts, devices and queues, 333–335
- get\_global
  - cl::sycl::nd\_item, 420
  - cl::sycl::nd\_range, 435
- get\_global\_linear\_id
  - cl::sycl::nd\_item, 421
- get\_global\_range
  - cl::sycl::group, 399, 400
  - cl::sycl::nd\_item, 422
- get\_group
  - cl::sycl::nd\_item, 423, 424
  - cl::sycl::nd\_range, 436
- get\_group\_linear\_id
  - cl::sycl::nd\_item, 425
- get\_group\_range
  - cl::sycl::group, 400
- get\_info
  - cl::sycl::context, 250
  - cl::sycl::device, 260, 261
  - cl::sycl::platform, 305, 306
  - cl::sycl::queue, 316
- get\_info\_string
  - cl::sycl::detail::host\_platform, 290
  - cl::sycl::detail::opencl\_platform, 295
  - cl::sycl::detail::platform, 299
- get\_item
  - cl::sycl::nd\_item, 425
- get\_kernel
  - cl::sycl::detail::task, 547
- get\_latest\_producer
  - cl::sycl::detail::buffer\_base, 113
- get\_linear
  - cl::sycl::group, 401
- get\_linear\_id
  - cl::sycl::item, 412
- get\_local
  - cl::sycl::nd\_item, 426, 427
  - cl::sycl::nd\_range, 436
- get\_local\_linear\_id
  - cl::sycl::nd\_item, 428
- get\_local\_range
  - cl::sycl::group, 401, 402
  - cl::sycl::nd\_item, 428
- get\_nd\_range
  - cl::sycl::group, 402
  - cl::sycl::nd\_item, 429
- get\_num\_groups
  - cl::sycl::nd\_item, 430, 431
- get\_offset
  - cl::sycl::group, 403
  - cl::sycl::item, 413
  - cl::sycl::nd\_item, 431
  - cl::sycl::nd\_range, 437
- get\_or\_register
  - cl::sycl::detail::cache, 481
- get\_pipe\_detail
  - cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking\_pipe >, 68
  - cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >, 65
  - cl::sycl::detail::pipe\_accessor, 162
  - Data access and storage in SYCL, 201
- get\_platform
  - cl::sycl::context, 251
  - cl::sycl::detail::context, 237
  - cl::sycl::detail::device, 253
  - cl::sycl::detail::host\_context, 240
  - cl::sycl::detail::host\_device, 495
  - cl::sycl::detail::opencl\_context, 506
  - cl::sycl::detail::opencl\_device, 513
  - cl::sycl::device, 261
- get\_platforms
  - cl::sycl::platform, 306
- get\_pointer
  - cl::sycl::accessor, 56
  - cl::sycl::detail::accessor, 81
- get\_queue
  - cl::sycl::detail::task, 547
- get\_range
  - cl::sycl::accessor, 56
  - cl::sycl::buffer, 136
  - cl::sycl::detail::accessor, 82
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 39
  - cl::sycl::detail::buffer, 103
  - cl::sycl::item, 414
- get\_size
  - cl::sycl::accessor, 57
  - cl::sycl::buffer, 136
  - cl::sycl::detail::accessor, 82
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 40
  - cl::sycl::detail::buffer, 104
- gl\_context\_interop
  - Platforms, contexts, devices and queues, 321
- global
  - Dealing with OpenCL address spaces, 227
  - Platforms, contexts, devices and queues, 329
- global\_config.hpp
  - TRISYCL\_WEAK\_ATTRIB\_PREFIX, 656
  - TRISYCL\_WEAK\_ATTRIB\_SUFFIX, 656
- global\_index
  - cl::sycl::item, 415
  - cl::sycl::nd\_item, 433



- global\_mem\_cache\_type
  - Platforms, contexts, devices and queues, 328
- global\_ptr
  - Dealing with OpenCL address spaces, 228
- global\_range
  - cl::sycl::item, 415
  - cl::sycl::nd\_range, 438
- gpu
  - Platforms, contexts, devices and queues, 327
- gpu\_selector
  - Platforms, contexts, devices and queues, 321
- group
  - cl::sycl::group, 398
- group\_id
  - cl::sycl::group, 407
- handler
  - cl::sycl::detail::accessor, 91
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 48
  - cl::sycl::handler, 274
  - cl::sycl::kernel, 288
- handler.hpp
  - TRISYCL\_ParallelForFunctor\_GLOBAL\_OFFSET, 711
  - TRISYCL\_ParallelForKernel\_RANGE, 712
  - TRISYCL\_parallel\_for\_functor\_GLOBAL, 711
- handler\_event, 486
- has\_extension
  - cl::sycl::detail::device, 254
  - cl::sycl::detail::host\_device, 495
  - cl::sycl::detail::host\_platform, 291
  - cl::sycl::detail::opencl\_device, 513
  - cl::sycl::detail::opencl\_platform, 296
  - cl::sycl::detail::platform, 300
  - cl::sycl::device, 262
  - cl::sycl::platform, 306
- hash
  - cl::sycl::detail::shared\_ptr\_implementation, 540
- Helpers to do array and tuple conversion, 337
  - expand, 338, 339
  - fill\_tuple, 339
  - tuple\_to\_array, 340
  - tuple\_to\_array\_iterate, 340
- host
  - Platforms, contexts, devices and queues, 327
- host\_selector
  - Platforms, contexts, devices and queues, 321
- id
  - cl::sycl::id, 408
- image\_allocator
  - Data access and storage in SYCL, 199
- implementation
  - cl::sycl::detail::pipe\_accessor, 167
  - cl::sycl::detail::shared\_ptr\_implementation, 541
  - cl::sycl::pipe\_reservation, 195
- implementation\_t
  - cl::sycl::accessor, 50, 63
  - cl::sycl::buffer, 126, 141
  - cl::sycl::context, 244
  - cl::sycl::detail::buffer\_waiter, 122, 123
  - cl::sycl::detail::pipe, 146
  - cl::sycl::device, 257
  - cl::sycl::kernel, 286, 288
  - cl::sycl::pipe, 170, 171
  - cl::sycl::queue, 309, 319
  - cl::sycl::static\_pipe, 197, 199
- include/CL/sycl.hpp, 557, 558
- include/CL/sycl/access.hpp, 559, 560
- include/CL/sycl/accessor.hpp, 561, 562
- include/CL/sycl/accessor/detail/local\_accessor.hpp, 575, 577
- include/CL/sycl/address\_space.hpp, 588, 590
- include/CL/sycl/address\_space/detail/address\_space.hpp, 581, 583
- include/CL/sycl/allocator.hpp, 592, 593
- include/CL/sycl/buffer.hpp, 601, 602
- include/CL/sycl/buffer/detail/accessor.hpp, 568, 569
- include/CL/sycl/buffer/detail/buffer.hpp, 594, 596
- include/CL/sycl/buffer/detail/buffer\_base.hpp, 609, 610
- include/CL/sycl/buffer/detail/buffer\_waiter.hpp, 614, 616
- include/CL/sycl/buffer\_allocator.hpp, 617, 618
- include/CL/sycl/command\_group/detail/task.hpp, 619, 620
- include/CL/sycl/context.hpp, 626, 627
- include/CL/sycl/context/detail/context.hpp, 623, 625
- include/CL/sycl/context/detail/host\_context.hpp, 632, 634
- include/CL/sycl/context/detail/opencl\_context.hpp, 635, 637
- include/CL/sycl/detail/array\_tuple\_helpers.hpp, 638, 640
- include/CL/sycl/detail/cache.hpp, 642, 643
- include/CL/sycl/detail/container\_element\_aspect.hpp, 645
- include/CL/sycl/detail/debug.hpp, 646, 649
- include/CL/sycl/detail/default\_classes.hpp, 651, 653
- include/CL/sycl/detail/global\_config.hpp, 654, 656
- include/CL/sycl/detail/linear\_id.hpp, 657, 658
- include/CL/sycl/detail/shared\_ptr\_implementation.hpp, 659, 660
- include/CL/sycl/detail/singleton.hpp, 662, 663
- include/CL/sycl/detail/small\_array.hpp, 663, 665
- include/CL/sycl/detail/unimplemented.hpp, 669, 670
- include/CL/sycl/device.hpp, 673, 675
- include/CL/sycl/device/detail/device.hpp, 671, 672
- include/CL/sycl/device/detail/device\_tail.hpp, 684
- include/CL/sycl/device/detail/host\_device.hpp, 685, 686
- include/CL/sycl/device/detail/opencl\_device.hpp, 688, 689
- include/CL/sycl/device\_selector.hpp, 691, 693
- include/CL/sycl/device\_selector/detail/device\_selector\_tail.hpp, 693, 695
- include/CL/sycl/error\_handler.hpp, 697, 698
- include/CL/sycl/event.hpp, 699, 700
- include/CL/sycl/exception.hpp, 700, 703

- include/CL/sycl/group.hpp, 705, 707
- include/CL/sycl/handler.hpp, 709, 712
- include/CL/sycl/handler\_event.hpp, 718
- include/CL/sycl/id.hpp, 719, 721
- include/CL/sycl/image.hpp, 722, 723
- include/CL/sycl/info/context.hpp, 630, 632
- include/CL/sycl/info/device.hpp, 678, 681
- include/CL/sycl/info/param\_traits.hpp, 724, 726
- include/CL/sycl/info/platform.hpp, 726, 729
- include/CL/sycl/item.hpp, 736, 738
- include/CL/sycl/kernel.hpp, 742, 744
- include/CL/sycl/kernel/detail/kernel.hpp, 739, 741
- include/CL/sycl/kernel/detail/opencv\_kernel.hpp, 745, 748
- include/CL/sycl/math.hpp, 749, 753
- include/CL/sycl/nd\_item.hpp, 755, 757
- include/CL/sycl/nd\_range.hpp, 759, 761
- include/CL/sycl/opencv\_types.hpp, 762, 768
- include/CL/sycl/parallelism.hpp, 777, 779
- include/CL/sycl/parallelism/detail/parallelism.hpp, 770, 773
- include/CL/sycl/pipe.hpp, 786, 787
- include/CL/sycl/pipe/detail/pipe.hpp, 779, 781
- include/CL/sycl/pipe/detail/pipe\_accessor.hpp, 789, 790
- include/CL/sycl/pipe\_reservation.hpp, 797, 798
- include/CL/sycl/pipe\_reservation/detail/pipe\_reservation.hpp, 793, 795
- include/CL/sycl/platform.hpp, 733, 734
- include/CL/sycl/platform/detail/host\_platform.hpp, 800, 802
- include/CL/sycl/platform/detail/host\_platform\_tail.hpp, 804
- include/CL/sycl/platform/detail/opencv\_platform.hpp, 805, 807
- include/CL/sycl/platform/detail/opencv\_platform\_tail.hpp, 809
- include/CL/sycl/platform/detail/platform.hpp, 730, 731
- include/CL/sycl/queue.hpp, 818, 820
- include/CL/sycl/queue/detail/host\_queue.hpp, 810, 811
- include/CL/sycl/queue/detail/opencv\_queue.hpp, 812, 814
- include/CL/sycl/queue/detail/queue.hpp, 815, 816
- include/CL/sycl/range.hpp, 824, 826
- include/CL/sycl/static\_pipe.hpp, 827, 828
- include/CL/sycl/vec.hpp, 829, 831
- input\_shared\_pointer
  - cl::sycl::detail::buffer, 109
- instance
  - cl::sycl::detail::opencv\_context, 507
  - cl::sycl::detail::opencv\_device, 514
  - cl::sycl::detail::opencv\_kernel, 520
  - cl::sycl::detail::opencv\_platform, 296
  - cl::sycl::detail::opencv\_queue, 527
  - cl::sycl::detail::singleton, 542
- is\_accelerator
  - cl::sycl::detail::device, 254
  - cl::sycl::detail::host\_device, 496
  - cl::sycl::detail::opencv\_device, 515
- cl::sycl::device, 262
- is\_cached
  - cl::sycl::buffer, 137
  - cl::sycl::detail::buffer\_base, 114
- is\_cpu
  - cl::sycl::detail::device, 254
  - cl::sycl::detail::host\_device, 496
  - cl::sycl::detail::opencv\_device, 515
  - cl::sycl::device, 263
- is\_data\_up\_to\_date
  - cl::sycl::buffer, 137
  - cl::sycl::detail::buffer\_base, 114
- is\_gpu
  - cl::sycl::detail::device, 254
  - cl::sycl::detail::host\_device, 497
  - cl::sycl::detail::opencv\_device, 515
  - cl::sycl::device, 263
- is\_host
  - cl::sycl::context, 251
  - cl::sycl::detail::context, 237
  - cl::sycl::detail::device, 254
  - cl::sycl::detail::host\_context, 241
  - cl::sycl::detail::host\_device, 497
  - cl::sycl::detail::host\_platform, 292
  - cl::sycl::detail::host\_queue, 500
  - cl::sycl::detail::opencv\_context, 508
  - cl::sycl::detail::opencv\_device, 516
  - cl::sycl::detail::opencv\_platform, 297
  - cl::sycl::detail::opencv\_queue, 528
  - cl::sycl::detail::platform, 300
  - cl::sycl::detail::queue, 534
  - cl::sycl::device, 264
  - cl::sycl::platform, 307
  - cl::sycl::queue, 316
- is\_read\_access
  - cl::sycl::detail::accessor, 83
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 40
- is\_read\_only
  - cl::sycl::buffer, 137
- is\_write\_access
  - cl::sycl::detail::accessor, 84
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 41
- item
  - cl::sycl::item, 410
- iterator
  - cl::sycl::detail::accessor, 74
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 34
  - cl::sycl::detail::pipe\_reservation, 174
  - cl::sycl::pipe\_reservation, 186
- k
  - cl::sycl::detail::opencv\_kernel, 522
- kernel
  - cl::sycl::detail::task, 554
  - cl::sycl::kernel, 286, 287
- kernel/detail/kernel.hpp



- TRISYCL\_ParallelForKernel\_RANGE, 741
- kernel\_end
  - cl::sycl::detail::queue, 534
- kernel\_start
  - cl::sycl::detail::queue, 535
- key\_type
  - cl::sycl::detail::cache, 480
- latest\_producer
  - cl::sycl::detail::buffer\_base, 119
- latest\_producer\_mutex
  - cl::sycl::detail::buffer\_base, 120
- linear\_id
  - Some helpers for the implementation, 360
- local
  - Dealing with OpenCL address spaces, 228
  - Platforms, contexts, devices and queues, 329
- local\_index
  - cl::sycl::nd\_item, 433
- local\_mem\_type
  - Platforms, contexts, devices and queues, 329
- local\_ptr
  - Dealing with OpenCL address spaces, 228
- local\_range
  - cl::sycl::nd\_range, 438
- m
  - cl::sycl::detail::cache, 483
- make\_id
  - Expressing parallelism through kernels, 443, 444
- make\_multi
  - Dealing with OpenCL address spaces, 231
- make\_range
  - Expressing parallelism through kernels, 444, 445
- Manage default configuration and types, 366
  - \_\_SYCL\_SINGLE\_SOURCE\_\_, 366
  - CL\_SYCL\_LANGUAGE\_VERSION, 366
  - TRISYCL\_CL\_LANGUAGE\_VERSION, 367
  - TRISYCL\_MAKE\_BOOST\_CIRCULARBUFFER↔\_THREAD\_SAFE, 367
  - TRISYCL\_SKIP\_OPENCL, 367
- map\_allocator
  - Data access and storage in SYCL, 200
- mark\_as\_written
  - cl::sycl::buffer, 138
  - cl::sycl::detail::buffer, 104
- math.hpp
  - TRISYCL\_MATH\_WRAP2, 751
  - TRISYCL\_MATH\_WRAP2s, 752
  - TRISYCL\_MATH\_WRAP3, 752
  - TRISYCL\_MATH\_WRAP3s, 752
  - TRISYCL\_MATH\_WRAP3ss, 753
  - TRISYCL\_MATH\_WRAP, 751
- message
  - cl::sycl::exception, 374
- min
  - cl::sycl, 467
- mode
  - cl::sycl::access, 470
  - cl::sycl::detail::pipe\_accessor, 167
  - cl::sycl::detail::pipe\_reservation, 182
- modified
  - cl::sycl::detail::buffer, 109
- move\_read\_reservation\_forward
  - cl::sycl::detail::pipe, 148
- move\_write\_reservation\_forward
  - cl::sycl::detail::pipe, 149
- multi\_ptr
  - Dealing with OpenCL address spaces, 229
- mutex\_class
  - cl::sycl, 466
- ND\_range
  - cl::sycl::nd\_item, 433
- name
  - Platforms, contexts, devices and queues, 324
- nd\_item
  - cl::sycl::nd\_item, 418
- nd\_range
  - cl::sycl::nd\_range, 435
- ndr
  - cl::sycl::group, 407
- non\_const\_value\_type
  - cl::sycl::detail::buffer, 95
- none
  - Platforms, contexts, devices and queues, 328, 329
- notify\_buffer\_destructor
  - cl::sycl::detail::buffer\_base, 120
- notify\_consumers
  - cl::sycl::detail::task, 548
- numa
  - Platforms, contexts, devices and queues, 325, 327
- number\_of\_users
  - cl::sycl::detail::buffer\_base, 120
- offset
  - cl::sycl::item, 416
  - cl::sycl::nd\_range, 438
- ok
  - cl::sycl::detail::pipe\_accessor, 168
  - cl::sycl::detail::pipe\_reservation, 182
- opengl
  - Platforms, contexts, devices and queues, 327
- opengl\_context
  - cl::sycl::detail::opengl\_context, 504
- opengl\_device
  - cl::sycl::detail::opengl\_device, 511
- opengl\_kernel
  - cl::sycl::detail::opengl\_kernel, 519
- opengl\_kernel.hpp
  - TRISYCL\_ParallelForKernel\_RANGE, 747
- opengl\_platform
  - cl::sycl::detail::opengl\_platform, 294
- opengl\_queue
  - cl::sycl::detail::opengl\_queue, 525
- opengl\_type
  - cl::sycl::detail::address\_space\_base, 222
  - cl::sycl::detail::address\_space\_object, 215

- cl::sycl::detail::address\_space\_variable, 224
- opengl\_types.hpp
  - TRISYCL\_BOOST\_COMPUTE\_NAME, 764
  - TRISYCL\_DECLARE\_CL\_TYPES, 764
  - TRISYCL\_DEFINE\_TYPES, 764
  - TRISYCL\_H\_DEFINE\_TYPE, 764
  - TRISYCL\_IS\_WRAPPER\_TRAIT, 765
  - TRISYCL\_SCALAR\_TYPES, 765
  - TRISYCL\_SIZED\_NAME, 765
  - TRISYCL\_TYPE\_ACTUAL\_NAME, 766
  - TRISYCL\_TYPE\_CL\_NAME, 766
  - TRISYCL\_TYPE\_NAME, 766
  - TRISYCL\_TYPEDEF\_TYPE, 766
  - TRISYCL\_WRAPPER\_CLASS\_2, 766
  - TRISYCL\_WRAPPER\_CLASS\_3, 767
  - TRISYCL\_WRAPPER\_CLASS\_4, 767
- operator BasicType
  - cl::sycl::detail::small\_array\_123< BasicType, FinalType, 1 >, 354
- operator bool
  - cl::sycl::detail::pipe\_accessor, 162
  - cl::sycl::detail::pipe\_reservation, 180
  - cl::sycl::pipe\_reservation, 192
- operator FinalType
  - cl::sycl::detail::small\_array, 348
- operator opengl\_type &
  - cl::sycl::detail::address\_space\_object, 216
  - cl::sycl::detail::address\_space\_variable, 225
- operator<
  - cl::sycl::detail::shared\_ptr\_implementation, 540
- operator<<
  - cl::sycl::detail::pipe\_accessor, 163
- operator>>
  - cl::sycl::detail::pipe\_accessor, 163
- operator\*
  - cl::sycl::accessor, 57, 58
  - cl::sycl::detail::accessor, 84, 85
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 41, 42
- operator()
  - cl::sycl::device\_selector, 272
  - cl::sycl::device\_type\_selector, 267
  - std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >, 487
  - std::hash< cl::sycl::context >, 488
  - std::hash< cl::sycl::device >, 489
  - std::hash< cl::sycl::kernel >, 490
  - std::hash< cl::sycl::platform >, 491
  - std::hash< cl::sycl::queue >, 492
- operator==
  - cl::sycl::detail::shared\_ptr\_implementation, 541
- operator[]
  - cl::sycl::accessor, 58–61
  - cl::sycl::detail::accessor, 85–88
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 42–45
  - cl::sycl::detail::pipe\_reservation, 180
  - cl::sycl::group, 404
  - cl::sycl::item, 414
  - cl::sycl::pipe\_reservation, 192
- owner\_queue
  - cl::sycl::detail::task, 554
- p
  - cl::sycl::detail::opengl\_platform, 297
  - cl::sycl::detail::pipe\_reservation, 182
- parallel\_OpenMP\_for\_iterate
  - cl::sycl::detail::parallel\_OpenMP\_for\_iterate, 439
- parallel\_for
  - cl::sycl::handler, 275
  - Expressing parallelism through kernels, 445–448
- parallel\_for\_global\_offset
  - Expressing parallelism through kernels, 449
- parallel\_for\_iterate
  - cl::sycl::detail::parallel\_for\_iterate, 439
  - cl::sycl::detail::parallel\_for\_iterate< 0, Range, ParallelForFunctor, Id >, 440
- parallel\_for\_work\_group
  - cl::sycl::handler, 276, 277
- parallel\_for\_work\_item
  - cl::sycl::group, 405, 406
  - Expressing parallelism through kernels, 450
- parallel\_for\_workgroup
  - Expressing parallelism through kernels, 451
- parallel\_for\_workitem
  - Expressing parallelism through kernels, 452
- param\_traits.hpp
  - TRISYCL\_INFO\_PARAM\_TRAITS\_ANY\_T, 725
  - TRISYCL\_INFO\_PARAM\_TRAITS, 725
- pipe
  - cl::sycl::detail::pipe, 146
  - cl::sycl::pipe, 170
- pipe\_accessor
  - cl::sycl::detail::pipe\_accessor, 160
- pipe\_reservation
  - cl::sycl::detail::pipe\_reservation, 175, 176
  - cl::sycl::pipe\_reservation, 187, 188
- platform
  - cl::sycl::platform, 302, 303
  - Platforms, contexts, devices and queues, 324, 329
- platform\_extensions
  - cl::sycl::detail::host\_platform, 292
- Platforms, contexts, devices and queues, 232
  - accelerator, 327
  - all, 327
  - context, 322
  - cpu, 327
  - cpu\_selector, 319
  - custom, 327
  - default\_selector, 320
  - defaults, 327
  - denorm, 328
  - device, 322
  - device::get\_info< info::device::device\_type >, 330
  - device::get\_info< info::device::local\_mem\_size >, 330

- device::get\_info< info::device::max\_compute\_units >, [331](#)
- device::get\_info< info::device::max\_mem\_alloc\_size >, [331](#)
- device::get\_info< info::device::max\_work\_group\_size >, [331](#)
- device::get\_info< info::device::name >, [332](#)
- device::get\_info< info::device::profile >, [332](#)
- device::get\_info< info::device::vendor >, [332](#)
- device\_affinity\_domain, [325](#)
- device\_exec\_capabilities, [320](#)
- device\_execution\_capabilities, [325](#)
- device\_fp\_config, [320](#)
- device\_partition\_property, [326](#)
- device\_partition\_type, [326](#)
- device\_queue\_properties, [321](#)
- device\_type, [327](#)
- devices, [322](#)
- extensions, [324](#)
- fma, [328](#)
- fp\_config, [328](#)
- get\_devices, [333–335](#)
- gl\_context\_interop, [321](#)
- global, [329](#)
- global\_mem\_cache\_type, [328](#)
- gpu, [327](#)
- gpu\_selector, [321](#)
- host, [327](#)
- host\_selector, [321](#)
- local, [329](#)
- local\_mem\_type, [329](#)
- name, [324](#)
- none, [328](#), [329](#)
- numa, [325](#), [327](#)
- opencl, [327](#)
- platform, [324](#), [329](#)
- profile, [324](#)
- TRISYCL\_WEAK\_ATTRIB\_SUFFIX, [336](#)
- unsupported, [325](#), [326](#)
- vendor, [324](#)
- pointer
  - cl::sycl::detail::container\_element\_aspect, [343](#)
  - cl::sycl::pipe\_reservation, [186](#)
- pointer\_t
  - cl::sycl::detail::address\_space\_ptr, [218](#)
- postlude
  - cl::sycl::detail::task, [548](#)
- prelude
  - cl::sycl::detail::task, [549](#)
- priv
  - Dealing with OpenCL address spaces, [229](#)
- private\_ptr
  - Dealing with OpenCL address spaces, [230](#)
- producer\_tasks
  - cl::sycl::detail::task, [554](#)
- profile
  - Platforms, contexts, devices and queues, [324](#)
- prologues
  - cl::sycl::detail::task, [554](#)
  - cl::sycl::detail::opencl\_context, [509](#)
  - cl::sycl::detail::opencl\_queue, [529](#)
  - queue
    - cl::sycl::detail::queue, [531](#)
    - cl::sycl::info, [477](#)
    - cl::sycl::queue, [310–314](#)
  - queue\_profiling
    - cl::sycl::info, [477](#)
  - r\_rid\_q
    - cl::sycl::detail::pipe, [156](#)
  - rank
    - cl::sycl::detail::pipe\_accessor, [168](#)
  - rbegin
    - cl::sycl::accessor, [62](#)
    - cl::sycl::detail::accessor, [88](#)
    - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, [46](#)
    - cl::sycl::pipe\_reservation, [193](#)
  - read
    - cl::sycl::detail::pipe, [150](#)
    - cl::sycl::detail::pipe\_accessor, [164](#)
  - read\_done
    - cl::sycl::detail::pipe, [156](#)
  - read\_reserved\_frozen
    - cl::sycl::detail::pipe, [157](#)
  - ready
    - cl::sycl::detail::buffer\_base, [120](#)
    - cl::sycl::detail::reserve\_id, [143](#)
    - cl::sycl::detail::task, [555](#)
  - ready\_mutex
    - cl::sycl::detail::buffer\_base, [120](#)
    - cl::sycl::detail::task, [555](#)
  - reference
    - cl::sycl::buffer, [126](#)
    - cl::sycl::detail::accessor, [74](#)
    - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, [34](#)
    - cl::sycl::detail::container\_element\_aspect, [343](#)
    - cl::sycl::detail::pipe\_accessor, [160](#)
    - cl::sycl::detail::pipe\_reservation, [174](#)
    - cl::sycl::pipe\_reservation, [186](#)
  - reference\_t
    - cl::sycl::detail::address\_space\_ptr, [219](#)
  - register\_accessor
    - cl::sycl::detail::accessor, [89](#)
  - release
    - cl::sycl::detail::buffer\_base, [114](#)
  - release\_buffers
    - cl::sycl::detail::task, [549](#)
  - remove
    - cl::sycl::detail::cache, [482](#)
  - rend
    - cl::sycl::accessor, [62](#)
    - cl::sycl::detail::accessor, [89](#)

- cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, [46](#)
  - cl::sycl::pipe\_reservation, [193](#)
- report\_error
  - cl::sycl::error\_handler, [370](#)
  - cl::sycl::trisycl::default\_error\_handler, [485](#)
- reserve
  - cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking\_pipe >, [69](#)
  - cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >, [66](#)
  - cl::sycl::detail::pipe\_accessor, [165](#)
- reserve\_id
  - cl::sycl::detail::reserve\_id, [142](#)
- reserve\_read
  - cl::sycl::detail::pipe, [151](#)
- reserve\_write
  - cl::sycl::detail::pipe, [152](#)
- reserved\_for\_reading
  - cl::sycl::detail::pipe, [153](#)
- reserved\_for\_writing
  - cl::sycl::detail::pipe, [153](#)
- reverse\_iterator
  - cl::sycl::detail::accessor, [74](#)
  - cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, [34](#)
  - cl::sycl::pipe\_reservation, [186](#)
- rid
  - cl::sycl::detail::pipe\_reservation, [182](#)
- rid\_iterator
  - cl::sycl::detail::pipe, [146](#)
- running\_kernels
  - cl::sycl::detail::queue, [536](#)
- schedule
  - cl::sycl::detail::task, [550](#)
- select\_device
  - cl::sycl::device\_selector, [272](#)
- set
  - cl::sycl::item, [415](#)
- set\_arg
  - cl::sycl::handler, [278](#), [279](#)
- set\_args
  - cl::sycl::handler, [279](#)
- set\_debug
  - cl::sycl::detail::pipe\_accessor, [165](#)
- set\_final\_data
  - cl::sycl::buffer, [138–140](#)
  - cl::sycl::detail::buffer, [105](#), [106](#)
- set\_global
  - cl::sycl::nd\_item, [432](#)
- set\_kernel
  - cl::sycl::detail::task, [552](#)
- set\_latest\_producer
  - cl::sycl::detail::buffer\_base, [115](#)
- set\_local
  - cl::sycl::nd\_item, [432](#)
- shared\_ptr\_class
  - cl::sycl, [466](#)
- shared\_ptr\_implementation
  - cl::sycl::detail::shared\_ptr\_implementation, [539](#)
  - cl::sycl::platform, [307](#)
- single\_task
  - cl::sycl::detail::kernel, [284](#)
  - cl::sycl::detail::opencl\_kernel, [521](#)
  - cl::sycl::handler, [280](#), [281](#)
- size
  - cl::sycl::detail::pipe, [154](#)
  - cl::sycl::detail::pipe\_accessor, [166](#)
  - cl::sycl::detail::pipe\_reservation, [181](#)
  - cl::sycl::detail::reserve\_id, [143](#)
  - cl::sycl::pipe\_reservation, [194](#)
- size\_type
  - cl::sycl::pipe\_reservation, [187](#)
- size\_with\_lock
  - cl::sycl::detail::pipe, [154](#)
- small\_array
  - cl::sycl::detail::small\_array, [346](#), [347](#)
- small\_array\_123
  - cl::sycl::detail::small\_array\_123< BasicType, FinalType, 1 >, [353](#), [354](#)
  - cl::sycl::detail::small\_array\_123< BasicType, FinalType, 2 >, [355](#), [356](#)
  - cl::sycl::detail::small\_array\_123< BasicType, FinalType, 3 >, [358](#), [359](#)
- Some helpers for the implementation, [342](#)
  - linear\_id, [360](#)
  - TRISYCL\_BOOST\_OPERATOR\_VECTOR\_OP, [359](#)
  - TRISYCL\_LOGICAL\_OPERATOR\_VECTOR\_OP, [359](#)
  - unimplemented, [361](#)
- start
  - cl::sycl::detail::reserve\_id, [143](#)
- static\_pipe
  - cl::sycl::static\_pipe, [197](#)
- std, [478](#)
- std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >, [486](#)
  - operator(), [487](#)
- std::hash< cl::sycl::context >, [487](#)
  - operator(), [488](#)
- std::hash< cl::sycl::device >, [488](#)
  - operator(), [489](#)
- std::hash< cl::sycl::kernel >, [489](#)
  - operator(), [490](#)
- std::hash< cl::sycl::platform >, [490](#)
  - operator(), [491](#)
- std::hash< cl::sycl::queue >, [491](#)
  - operator(), [492](#)
- string\_class
  - cl::sycl, [466](#)
- submit
  - cl::sycl::queue, [316](#), [317](#)
- super
  - cl::sycl::detail::address\_space\_array, [209](#)
  - cl::sycl::detail::address\_space\_fundamental, [212](#)

- cl::sycl::detail::address\_space\_ptr, [219](#)
  - cl::sycl::detail::address\_space\_variable, [224](#)
- sync\_with\_host
  - cl::sycl::detail::buffer\_base, [115](#)
- TRISYCL\_BOOST\_COMPUTE\_NAME
  - opcnl\_types.hpp, [764](#)
- TRISYCL\_BOOST\_OPERATOR\_VECTOR\_OP
  - Some helpers for the implementation, [359](#)
- TRISYCL\_CL\_LANGUAGE\_VERSION
  - Manage default configuration and types, [367](#)
- TRISYCL\_DECLARE\_CL\_TYPES
  - opcnl\_types.hpp, [764](#)
- TRISYCL\_DEFINE\_TYPES
  - opcnl\_types.hpp, [764](#)
- TRISYCL\_DEFINE\_VEC\_TYPE\_SIZE
  - Vector types in SYCL, [460](#)
- TRISYCL\_DEFINE\_VEC\_TYPE
  - Vector types in SYCL, [459](#)
- TRISYCL\_DUMP\_T
  - debug.hpp, [648](#)
- TRISYCL\_DUMP
  - debug.hpp, [648](#)
- TRISYCL\_H\_DEFINE\_TYPE
  - opcnl\_types.hpp, [764](#)
- TRISYCL\_INFO\_PARAM\_TRAITS\_ANY\_T
  - param\_traits.hpp, [725](#)
- TRISYCL\_INFO\_PARAM\_TRAITS
  - param\_traits.hpp, [725](#)
- TRISYCL\_INTERNAL\_DUMP
  - debug.hpp, [648](#)
- TRISYCL\_IS\_WRAPPER\_TRAIT
  - opcnl\_types.hpp, [765](#)
- TRISYCL\_LOGICAL\_OPERATOR\_VECTOR\_OP
  - Some helpers for the implementation, [359](#)
- TRISYCL\_MAKE\_BOOST\_CIRCULARBUFFER\_TH↔
  - READ\_SAFE
  - Manage default configuration and types, [367](#)
- TRISYCL\_MATH\_WRAP2
  - math.hpp, [751](#)
- TRISYCL\_MATH\_WRAP2s
  - cl::sycl, [468](#)
  - math.hpp, [752](#)
- TRISYCL\_MATH\_WRAP3
  - math.hpp, [752](#)
- TRISYCL\_MATH\_WRAP3s
  - math.hpp, [752](#)
- TRISYCL\_MATH\_WRAP3ss
  - math.hpp, [753](#)
- TRISYCL\_MATH\_WRAP
  - cl::sycl, [467](#)
  - math.hpp, [751](#)
- TRISYCL\_ParallelForFunctor\_GLOBAL\_OFFSET
  - handler.hpp, [711](#)
- TRISYCL\_ParallelForKernel\_RANGE
  - cl::sycl::detail::kernel, [284](#)
  - cl::sycl::detail::opcnl\_kernel, [521](#)
  - handler.hpp, [712](#)
  - kernel/detail/kernel.hpp, [741](#)
  - opcnl\_kernel.hpp, [747](#)
- TRISYCL\_SCALAR\_TYPES
  - opcnl\_types.hpp, [765](#)
- TRISYCL\_SIZED\_NAME
  - opcnl\_types.hpp, [765](#)
- TRISYCL\_SKIP\_OPENCL
  - Manage default configuration and types, [367](#)
- TRISYCL\_TYPE\_ACTUAL\_NAME
  - opcnl\_types.hpp, [766](#)
- TRISYCL\_TYPE\_CL\_NAME
  - opcnl\_types.hpp, [766](#)
- TRISYCL\_TYPE\_NAME
  - opcnl\_types.hpp, [766](#)
- TRISYCL\_TPEDEF\_TYPE
  - opcnl\_types.hpp, [766](#)
- TRISYCL\_WEAK\_ATTRIB\_PREFIX
  - global\_config.hpp, [656](#)
- TRISYCL\_WEAK\_ATTRIB\_SUFFIX
  - global\_config.hpp, [656](#)
  - Platforms, contexts, devices and queues, [336](#)
- TRISYCL\_WRAPPER\_CLASS\_2
  - opcnl\_types.hpp, [766](#)
- TRISYCL\_WRAPPER\_CLASS\_3
  - opcnl\_types.hpp, [767](#)
- TRISYCL\_WRAPPER\_CLASS\_4
  - opcnl\_types.hpp, [767](#)
- TRISYCL\_parallel\_for\_functor\_GLOBAL
  - cl::sycl::handler, [281](#)
  - handler.hpp, [711](#)
- target
  - cl::sycl::access, [470](#)
  - cl::sycl::detail::pipe\_accessor, [168](#)
  - cl::sycl::detail::pipe\_reservation, [183](#)
- task
  - cl::sycl::detail::accessor, [91](#)
  - cl::sycl::detail::task, [546](#)
  - cl::sycl::handler, [282](#)
- throw\_asynchronous
  - cl::sycl::queue, [317](#)
- trace\_kernel
  - Debugging and tracing support, [364](#)
- track\_access\_mode
  - cl::sycl::detail::buffer, [106](#)
- tuple\_to\_array
  - Helpers to do array and tuple conversion, [340](#)
- tuple\_to\_array\_iterate
  - Helpers to do array and tuple conversion, [340](#)
- type
  - cl::sycl::detail::address\_space\_base, [222](#)
  - cl::sycl::detail::ocl\_type, [205](#)
  - cl::sycl::detail::ocl\_type< T, constant\_address\_↔  
space >, [205](#)
  - cl::sycl::detail::ocl\_type< T, generic\_address\_↔  
space >, [206](#)
  - cl::sycl::detail::ocl\_type< T, global\_address\_space  
>, [206](#)
  - cl::sycl::detail::ocl\_type< T, local\_address\_space  
>, [207](#)

- cl::sycl::detail::ocl\_type< T, private\_address\_↵  
space >, 207
  - cl::sycl::device, 264
- unimplemented
  - Some helpers for the implementation, 361
- unique\_ptr\_class
  - cl::sycl, 467
- unsupported
  - Platforms, contexts, devices and queues, 325, 326
- update\_buffer\_state
  - cl::sycl::detail::buffer\_base, 116
- use
  - cl::sycl::detail::buffer\_base, 118
- use\_count
  - cl::sycl::buffer, 140
- used\_for\_reading
  - cl::sycl::detail::pipe, 157
- used\_for\_writing
  - cl::sycl::detail::pipe, 157
- value\_type
  - cl::sycl::buffer, 127
  - cl::sycl::detail::accessor, 74
  - cl::sycl::detail::accessor< T, Dimensions, Mode,  
access::target::local >, 35
  - cl::sycl::detail::buffer, 95
  - cl::sycl::detail::cache, 480
  - cl::sycl::detail::container\_element\_aspect, 343
  - cl::sycl::detail::pipe, 146
  - cl::sycl::detail::pipe\_accessor, 160
  - cl::sycl::detail::pipe\_reservation, 174
  - cl::sycl::pipe, 170
  - cl::sycl::pipe\_reservation, 187
  - cl::sycl::static\_pipe, 197
- variable
  - cl::sycl::detail::address\_space\_variable, 226
- vec
  - cl::sycl::vec, 457, 458
- Vector types in SYCL, 455
  - TRISYCL\_DEFINE\_VEC\_TYPE\_SIZE, 460
  - TRISYCL\_DEFINE\_VEC\_TYPE, 459
- vector\_class
  - cl::sycl, 467
- vendor
  - Platforms, contexts, devices and queues, 324
- w\_rid\_q
  - cl::sycl::detail::pipe, 157
- wait
  - cl::sycl::detail::buffer\_base, 118
  - cl::sycl::detail::task, 552
  - cl::sycl::queue, 318
- wait\_and\_throw
  - cl::sycl::queue, 318
- wait\_for\_kernel\_execution
  - cl::sycl::detail::queue, 535
- wait\_for\_producers
  - cl::sycl::detail::task, 553
- waiter
  - Data access and storage in SYCL, 202
- weak\_ptr\_class
  - cl::sycl, 467
- what
  - cl::sycl::exception, 373
- writable\_array\_type
  - cl::sycl::detail::accessor< T, Dimensions, Mode,  
access::target::local >, 35
- writable\_array\_view\_type
  - cl::sycl::detail::accessor, 75
- write
  - cl::sycl::detail::pipe, 155
  - cl::sycl::detail::pipe\_accessor, 166
- write\_done
  - cl::sycl::detail::pipe, 157
- x
  - cl::sycl::detail::small\_array, 348
- y
  - cl::sycl, 469
  - cl::sycl::detail::small\_array, 349
- z
  - cl::sycl, 469
  - cl::sycl::detail::small\_array, 350