

Nvwa

1.1

Generated by Doxygen 1.8.14

Contents

1	Namespace Index	1
1.1	Namespace List	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Namespace Documentation	9
5.1	nvwa Namespace Reference	9
5.1.1	Detailed Description	13
5.1.2	Typedef Documentation	13
5.1.2.1	leak_whitelist_callback_t	14
5.1.2.2	stacktrace_print_callback_t	14
5.1.3	Enumeration Type Documentation	14
5.1.3.1	storage_policy	14
5.1.4	Function Documentation	15
5.1.4.1	alloc_mem()	15
5.1.4.2	apply() [1/2]	15
5.1.4.3	apply() [2/2]	16
5.1.4.4	check_leaks()	16

5.1.4.5	check_mem_corruption()	16
5.1.4.6	compose() [1/3]	17
5.1.4.7	compose() [2/3]	17
5.1.4.8	compose() [3/3]	17
5.1.4.9	create_tree() [1/2]	18
5.1.4.10	create_tree() [2/2]	18
5.1.4.11	fix_curry()	18
5.1.4.12	fix_simple() [1/2]	19
5.1.4.13	fix_simple() [2/2]	19
5.1.4.14	fmap() [1/3]	20
5.1.4.15	fmap() [2/3]	20
5.1.4.16	fmap() [3/3]	20
5.1.4.17	free_pointer()	21
5.1.4.18	is_leak_whitelisted()	21
5.1.4.19	lift_optional()	22
5.1.4.20	make_curry() [1/3]	22
5.1.4.21	make_curry() [2/3]	22
5.1.4.22	make_curry() [3/3]	23
5.1.4.23	pipeline()	23
5.1.4.24	print_position()	24
5.1.4.25	print_position_from_addr()	24
5.1.4.26	print_stacktrace()	24
5.1.4.27	reduce() [1/4]	25
5.1.4.28	reduce() [2/4]	25
5.1.4.29	reduce() [3/4]	26
5.1.4.30	reduce() [4/4]	26
5.1.4.31	split()	27
5.1.4.32	swap() [1/2]	27
5.1.4.33	swap() [2/2]	27
5.1.4.34	wrap_args_as_pair()	28
5.1.4.35	wrap_args_as_tuple()	28
5.1.5	Variable Documentation	29
5.1.5.1	__debug_new_count	29
5.1.5.2	BUFFER_SIZE	29
5.1.5.3	leak_whitelist_callback	29
5.1.5.4	new_output_fp	29
5.1.5.5	new_progname	29
5.1.5.6	PLATFORM_MEM_ALIGNMENT	30
5.1.5.7	stacktrace_print_callback	30

6	Class Documentation	31
6.1	nvwa::mem_pool_base::_Block_list Struct Reference	31
6.1.1	Detailed Description	31
6.2	nvwa::bool_array::_Element<_Byte_type> Class Template Reference	32
6.2.1	Detailed Description	32
6.2.2	Constructor & Destructor Documentation	32
6.2.2.1	_Element()	32
6.2.3	Member Function Documentation	32
6.2.3.1	operator bool()	33
6.2.3.2	operator=()	33
6.3	nvwa::fixed_mem_pool<_Tp>::alignment Struct Reference	33
6.3.1	Detailed Description	33
6.4	nvwa::bad_optional_access Class Reference	34
6.4.1	Detailed Description	34
6.5	nvwa::basic_mmap_byte_reader<_Tp> Class Template Reference	34
6.5.1	Detailed Description	34
6.6	nvwa::basic_mmap_line_reader<_Tp> Class Template Reference	34
6.6.1	Detailed Description	35
6.6.2	Member Enumeration Documentation	35
6.6.2.1	strip_type	35
6.6.3	Member Function Documentation	35
6.6.3.1	read()	36
6.7	nvwa::basic_split_view<_StringType, _DelimiterType> Class Template Reference	36
6.7.1	Detailed Description	36
6.7.2	Constructor & Destructor Documentation	37
6.7.2.1	basic_split_view()	37
6.7.3	Member Function Documentation	37
6.7.3.1	to_vector()	37
6.7.3.2	to_vector_sv()	37
6.8	nvwa::fixed_mem_pool<_Tp>::block_size Struct Reference	38

6.8.1	Detailed Description	38
6.9	nvwa::bool_array Class Reference	38
6.9.1	Detailed Description	40
6.9.2	Member Typedef Documentation	40
6.9.2.1	byte	41
6.9.2.2	size_type	41
6.9.3	Constructor & Destructor Documentation	41
6.9.3.1	bool_array() [1/3]	41
6.9.3.2	bool_array() [2/3]	41
6.9.3.3	bool_array() [3/3]	42
6.9.4	Member Function Documentation	42
6.9.4.1	at()	42
6.9.4.2	copy_to_bitmap()	43
6.9.4.3	count() [1/2]	43
6.9.4.4	count() [2/2]	43
6.9.4.5	create()	44
6.9.4.6	find() [1/2]	44
6.9.4.7	find() [2/2]	45
6.9.4.8	find_until()	45
6.9.4.9	get_8bits()	46
6.9.4.10	get_num_bytes_from_bits()	46
6.9.4.11	initialize()	46
6.9.4.12	merge_and()	47
6.9.4.13	merge_or()	47
6.9.4.14	operator=()	48
6.9.4.15	operator[]() [1/2]	48
6.9.4.16	operator[]() [2/2]	48
6.9.4.17	reset()	49
6.9.4.18	set()	49
6.9.4.19	size()	49

6.9.4.20	swap()	50
6.9.5	Member Data Documentation	50
6.9.5.1	_S_bit_ordinal	50
6.9.5.2	npos	50
6.10	nvwa::breadth_first_iteration<_Tree> Class Template Reference	50
6.10.1	Detailed Description	51
6.11	nvwa::class_level_lock<_Host,_RealLock> Class Template Reference	51
6.11.1	Detailed Description	51
6.12	nvwa::class_level_lock<_Host,false> Class Template Reference	52
6.12.1	Detailed Description	52
6.13	nvwa::debug_new_counter Class Reference	52
6.13.1	Detailed Description	52
6.13.2	Constructor & Destructor Documentation	53
6.13.2.1	~debug_new_counter()	53
6.14	nvwa::debug_new_recorder Class Reference	53
6.14.1	Detailed Description	53
6.14.2	Constructor & Destructor Documentation	53
6.14.2.1	debug_new_recorder()	54
6.14.3	Member Function Documentation	54
6.14.3.1	_M_process()	54
6.14.3.2	operator-> *()	54
6.15	nvwa::delete_object Struct Reference	54
6.15.1	Detailed Description	55
6.16	nvwa::depth_first_iteration<_Tree> Class Template Reference	55
6.16.1	Detailed Description	55
6.17	nvwa::dereference Struct Reference	55
6.17.1	Detailed Description	56
6.18	nvwa::dereference_less Struct Reference	56
6.18.1	Detailed Description	56
6.19	nvwa::fast_mutex Class Reference	56

6.19.1 Detailed Description	57
6.20 nvwa::fast_mutex_autolock Class Reference	57
6.20.1 Detailed Description	57
6.21 nvwa::fc_queue< _Tp, _Alloc > Class Template Reference	57
6.21.1 Detailed Description	58
6.21.2 Constructor & Destructor Documentation	59
6.21.2.1 fc_queue() [1/4]	59
6.21.2.2 fc_queue() [2/4]	59
6.21.2.3 fc_queue() [3/4]	60
6.21.2.4 fc_queue() [4/4]	60
6.21.2.5 ~fc_queue()	61
6.21.3 Member Function Documentation	61
6.21.3.1 back() [1/2]	61
6.21.3.2 back() [2/2]	61
6.21.3.3 capacity()	62
6.21.3.4 contains()	62
6.21.3.5 empty()	62
6.21.3.6 front() [1/2]	63
6.21.3.7 front() [2/2]	63
6.21.3.8 full()	63
6.21.3.9 get_allocator()	64
6.21.3.10 operator=() [1/2]	64
6.21.3.11 operator=() [2/2]	64
6.21.3.12 pop()	65
6.21.3.13 push()	65
6.21.3.14 size()	65
6.21.3.15 swap()	66
6.22 nvwa::file_line_reader Class Reference	66
6.22.1 Detailed Description	67
6.22.2 Member Enumeration Documentation	67

6.22.2.1	strip_type	67
6.22.3	Constructor & Destructor Documentation	67
6.22.3.1	file_line_reader()	67
6.22.3.2	~file_line_reader()	68
6.22.4	Member Function Documentation	68
6.22.4.1	read()	68
6.23	nvwa::fixed_mem_pool< _Tp > Class Template Reference	68
6.23.1	Detailed Description	69
6.23.2	Member Function Documentation	70
6.23.2.1	allocate()	70
6.23.2.2	bad_alloc_handler()	70
6.23.2.3	deallocate()	70
6.23.2.4	deinitialize()	70
6.23.2.5	get_alloc_count()	71
6.23.2.6	initialize()	71
6.23.2.7	is_initialized()	71
6.23.3	Member Data Documentation	72
6.23.3.1	_S_alloc_cnt	72
6.23.3.2	_S_first_avail_ptr	72
6.23.3.3	_S_mem_pool_ptr	72
6.24	nvwa::in_order_iteration< _Tree > Class Template Reference	72
6.24.1	Detailed Description	72
6.25	nvwa::istream_line_reader Class Reference	73
6.25.1	Detailed Description	73
6.26	nvwa::basic_mmap_byte_reader< _Tp >::iterator Class Reference	73
6.26.1	Detailed Description	74
6.27	nvwa::basic_mmap_line_reader< _Tp >::iterator Class Reference	74
6.27.1	Detailed Description	74
6.28	nvwa::basic_split_view< _StringType, _DelimiterType >::iterator Class Reference	75
6.28.1	Detailed Description	75

6.29	nvwa::istream_line_reader::iterator Class Reference	75
6.29.1	Detailed Description	75
6.30	nvwa::file_line_reader::iterator Class Reference	75
6.30.1	Detailed Description	76
6.30.2	Constructor & Destructor Documentation	76
6.30.2.1	iterator() [1/2]	76
6.30.2.2	~iterator()	76
6.30.2.3	iterator() [2/2]	77
6.30.3	Member Function Documentation	77
6.30.3.1	operator=()	77
6.30.3.2	swap()	77
6.31	nvwa::object_level_lock<_Host>::lock Class Reference	78
6.31.1	Detailed Description	78
6.32	nvwa::class_level_lock<_Host, _RealLock>::lock Class Reference	78
6.32.1	Detailed Description	79
6.33	nvwa::class_level_lock<_Host, false>::lock Class Reference	79
6.33.1	Detailed Description	79
6.34	nvwa::mem_pool_base Class Reference	79
6.34.1	Detailed Description	80
6.34.2	Member Function Documentation	80
6.34.2.1	alloc_sys()	80
6.34.2.2	dealloc_sys()	81
6.34.2.3	recycle()	81
6.35	nvwa::new_ptr_list_t Struct Reference	81
6.35.1	Detailed Description	82
6.36	nvwa::object_level_lock<_Host> Class Template Reference	82
6.36.1	Detailed Description	83
6.37	nvwa::optional<_Tp> Class Template Reference	83
6.37.1	Detailed Description	83
6.38	nvwa::output_object<_OutputStrm, _StringType> Struct Template Reference	84

6.38.1 Detailed Description	84
6.39 <code>nvwa::smart_ptr<_Tp, _Policy></code> Struct Template Reference	84
6.39.1 Detailed Description	84
6.40 <code>nvwa::smart_ptr<_Tp, storage_policy::shared></code> Struct Template Reference	84
6.40.1 Detailed Description	85
6.41 <code>nvwa::smart_ptr<_Tp, storage_policy::unique></code> Struct Template Reference	85
6.41.1 Detailed Description	85
6.42 <code>nvwa::static_mem_pool<_Sz, _Gid></code> Class Template Reference	85
6.42.1 Detailed Description	86
6.42.2 Member Function Documentation	87
6.42.2.1 <code>allocate()</code>	87
6.42.2.2 <code>deallocate()</code>	87
6.42.2.3 <code>instance()</code>	87
6.42.2.4 <code>instance_known()</code>	88
6.42.2.5 <code>recycle()</code>	88
6.43 <code>nvwa::static_mem_pool_set</code> Class Reference	88
6.43.1 Detailed Description	89
6.43.2 Member Function Documentation	89
6.43.2.1 <code>add()</code>	89
6.43.2.2 <code>instance()</code>	89
6.43.2.3 <code>recycle()</code>	89
6.44 <code>nvwa::tree<_Tp, _Policy></code> Class Template Reference	90
6.44.1 Detailed Description	90

7 File Documentation	91
7.1 <code>_nvwa.h</code> File Reference	91
7.1.1 Detailed Description	91
7.2 <code>bool_array.cpp</code> File Reference	92
7.2.1 Detailed Description	92
7.3 <code>bool_array.h</code> File Reference	92
7.3.1 Detailed Description	93
7.4 <code>c++11.h</code> File Reference	94
7.4.1 Detailed Description	94
7.5 <code>class_level_lock.h</code> File Reference	94
7.5.1 Detailed Description	95
7.6 <code>cont_ptr_utils.h</code> File Reference	96
7.6.1 Detailed Description	97
7.7 <code>debug_new.cpp</code> File Reference	97
7.7.1 Detailed Description	99
7.7.2 Macro Definition Documentation	99
7.7.2.1 <code>_DEBUG_NEW_ALIGNMENT</code>	100
7.7.2.2 <code>_DEBUG_NEW_CALLER_ADDRESS</code>	100
7.7.2.3 <code>_DEBUG_NEW_ERROR_ACTION</code>	100
7.7.2.4 <code>_DEBUG_NEW_FILENAME_LEN</code>	100
7.7.2.5 <code>_DEBUG_NEW_PROGNAME</code>	100
7.7.2.6 <code>_DEBUG_NEW_REDEFINE_NEW</code>	101
7.7.2.7 <code>_DEBUG_NEW_REMEMBER_STACK_TRACE</code>	101
7.7.2.8 <code>_DEBUG_NEW_STD_OPER_NEW</code>	101
7.7.2.9 <code>_DEBUG_NEW_TAILCHECK</code>	101
7.7.2.10 <code>_DEBUG_NEW_USE_ADDR2LINE</code>	102
7.7.3 Function Documentation	102
7.7.3.1 <code>operator delete()</code> [1/3]	102
7.7.3.2 <code>operator delete()</code> [2/3]	102
7.7.3.3 <code>operator delete()</code> [3/3]	103

7.7.3.4	operator delete[]() [1/3]	103
7.7.3.5	operator delete[]() [2/3]	103
7.7.3.6	operator delete[]() [3/3]	104
7.7.3.7	operator new() [1/3]	104
7.7.3.8	operator new() [2/3]	104
7.7.3.9	operator new() [3/3]	105
7.7.3.10	operator new[]() [1/3]	105
7.7.3.11	operator new[]() [2/3]	106
7.7.3.12	operator new[]() [3/3]	106
7.8	debug_new.h File Reference	107
7.8.1	Detailed Description	109
7.8.2	Macro Definition Documentation	109
7.8.2.1	_DEBUG_NEW_TYPE	109
7.8.2.2	DEBUG_NEW	109
7.8.3	Function Documentation	109
7.8.3.1	operator delete()	109
7.8.3.2	operator delete[]()	110
7.8.3.3	operator new()	110
7.8.3.4	operator new[]()	111
7.9	fast_mutex.h File Reference	111
7.9.1	Detailed Description	113
7.9.2	Macro Definition Documentation	113
7.9.2.1	__VOLATILE	113
7.9.2.2	_FAST_MUTEX_ASSERT	113
7.9.2.3	_FAST_MUTEX_CHECK_INITIALIZATION	113
7.10	fc_queue.h File Reference	114
7.10.1	Detailed Description	114
7.11	file_line_reader.cpp File Reference	115
7.11.1	Detailed Description	115
7.12	file_line_reader.h File Reference	116

7.12.1 Detailed Description	117
7.13 fixed_mem_pool.h File Reference	117
7.13.1 Detailed Description	118
7.13.2 Macro Definition Documentation	118
7.13.2.1 DECLARE_FIXED_MEM_POOL	119
7.13.2.2 DECLARE_FIXED_MEM_POOL__NOTHROW	119
7.13.2.3 DECLARE_FIXED_MEM_POOL__THROW_NOCHECK	120
7.14 functional.h File Reference	120
7.14.1 Detailed Description	122
7.15 istream_line_reader.h File Reference	123
7.15.1 Detailed Description	123
7.16 mem_pool_base.cpp File Reference	124
7.16.1 Detailed Description	124
7.17 mem_pool_base.h File Reference	124
7.17.1 Detailed Description	125
7.18 mmap_byte_reader.h File Reference	126
7.18.1 Detailed Description	126
7.19 mmap_line_reader.h File Reference	127
7.19.1 Detailed Description	127
7.20 mmap_reader_base.h File Reference	128
7.20.1 Detailed Description	128
7.21 object_level_lock.h File Reference	129
7.21.1 Detailed Description	129
7.22 pctime.h File Reference	130
7.22.1 Detailed Description	130
7.23 set_assign.h File Reference	130
7.23.1 Detailed Description	131
7.24 split.h File Reference	131
7.24.1 Detailed Description	132
7.25 static_mem_pool.cpp File Reference	132
7.25.1 Detailed Description	133
7.26 static_mem_pool.h File Reference	133
7.26.1 Detailed Description	134
7.26.2 Macro Definition Documentation	134
7.26.2.1 DECLARE_STATIC_MEM_POOL	134
7.26.2.2 DECLARE_STATIC_MEM_POOL__NOTHROW	135
7.26.2.3 DECLARE_STATIC_MEM_POOL_GROUPED	135
7.26.2.4 DECLARE_STATIC_MEM_POOL_GROUPED__NOTHROW	136
7.27 tree.h File Reference	137
7.27.1 Detailed Description	138

Chapter 1

Namespace Index

1.1 Namespace List

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

nvwa	Namespace of the nvwa project	9
----------------------	---	---

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

<code>nvwa::mem_pool_base::_Block_list</code>	31
<code>nvwa::bool_array::_Element<_Byte_type></code>	32
<code>nvwa::fixed_mem_pool<_Tp>::alignment</code>	33
<code>nvwa::bad_optional_access</code>	34
<code>nvwa::basic_mmap_byte_reader<_Tp></code>	34
<code>nvwa::basic_mmap_line_reader<_Tp></code>	34
<code>nvwa::basic_split_view<_StringType, _DelimiterType></code>	36
<code>nvwa::fixed_mem_pool<_Tp>::block_size</code>	38
<code>nvwa::bool_array</code>	38
<code>nvwa::breadth_first_iteration<_Tree></code>	50
<code>nvwa::class_level_lock<_Host, _RealLock></code>	51
<code>nvwa::class_level_lock<_Host, false></code>	52
<code>nvwa::debug_new_counter</code>	52
<code>nvwa::debug_new_recorder</code>	53
<code>nvwa::delete_object</code>	54
<code>nvwa::depth_first_iteration<_Tree></code>	55
<code>nvwa::dereference</code>	55
<code>nvwa::dereference_less</code>	56
<code>nvwa::fast_mutex</code>	56
<code>nvwa::fast_mutex_autolock</code>	57
<code>nvwa::fc_queue<_Tp, _Alloc></code>	57
<code>nvwa::file_line_reader</code>	66
<code>nvwa::fixed_mem_pool<_Tp></code>	68
<code>nvwa::in_order_iteration<_Tree></code>	72
<code>nvwa::istream_line_reader</code>	73
<code>nvwa::basic_mmap_byte_reader<_Tp>::iterator</code>	73
<code>nvwa::basic_mmap_line_reader<_Tp>::iterator</code>	74
<code>nvwa::basic_split_view<_StringType, _DelimiterType>::iterator</code>	75
<code>nvwa::istream_line_reader::iterator</code>	75
<code>nvwa::file_line_reader::iterator</code>	75
<code>nvwa::object_level_lock<_Host>::lock</code>	78
<code>nvwa::class_level_lock<_Host, _RealLock>::lock</code>	78
<code>nvwa::class_level_lock<_Host, false>::lock</code>	79
<code>nvwa::mem_pool_base</code>	79
<code>nvwa::static_mem_pool<_Sz, _Gid></code>	85

<code>nvwa::new_ptr_list_t</code>	81
<code>nvwa::object_level_lock<_Host></code>	82
<code>nvwa::optional<_Tp></code>	83
<code>nvwa::output_object<_OutputStrm, _StringType></code>	84
<code>nvwa::smart_ptr<_Tp, _Policy></code>	84
<code>nvwa::smart_ptr<_Tp, storage_policy::shared></code>	84
<code>nvwa::smart_ptr<_Tp, storage_policy::unique></code>	85
<code>nvwa::static_mem_pool_set</code>	88
<code>nvwa::tree<_Tp, _Policy></code>	90

Chapter 3

Class Index

3.1 Class List

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

nvwa::mem_pool_base::_Block_list	31
Structure to store the next available memory block	
nvwa::bool_array::_Element<_Byte_type>	32
Class to represent a reference to an array element	
nvwa::fixed_mem_pool<_Tp>::alignment	33
Specializable struct to define the alignment of an object in the fixed_mem_pool	
nvwa::bad_optional_access	34
Class for bad optional access exception	
nvwa::basic_mmap_byte_reader<_Tp>	34
Class template to allow iteration over all bytes of a mmappable file	
nvwa::basic_mmap_line_reader<_Tp>	34
Class template to allow iteration over all lines of a mmappable file	
nvwa::basic_split_view<_StringType, _DelimiterType>	36
Class to allow iteration over split items from the input	
nvwa::fixed_mem_pool<_Tp>::block_size	38
Struct to calculate the block size based on the (specializable) alignment value	
nvwa::bool_array	38
Class to represent a packed boolean array	
nvwa::breadth_first_iteration<_Tree>	50
Iteration class for breadth-first traversal	
nvwa::class_level_lock<_Host, _RealLock>	51
Helper class for class-level locking	
nvwa::class_level_lock<_Host, false>	52
Partial specialization that makes null locking	
nvwa::debug_new_counter	52
Counter class for on-exit leakage check	
nvwa::debug_new_recorder	53
Recorder class to remember the call context	
nvwa::delete_object	54
Functor to delete objects pointed by a container of pointers	
nvwa::depth_first_iteration<_Tree>	55
Iteration class for depth-first traversal	
nvwa::dereference	55
Functor to return objects pointed by a container of pointers	
nvwa::dereference_less	56
Functor to compare objects pointed by a container of pointers	

nvwa::fast_mutex	Class for non-reentrant fast mutexes	56
nvwa::fast_mutex_autolock	RAII lock class for fast_mutex	57
nvwa::fc_queue<_Tp, _Alloc>	Class to represent a fixed-capacity queue	57
nvwa::file_line_reader	Class to allow iteration over all lines of a text file	66
nvwa::fixed_mem_pool<_Tp>	Class template to manipulate a fixed-size memory pool	68
nvwa::in_order_iteration<_Tree>	Iteration class for in-order traversal	72
nvwa::istream_line_reader	Class to allow iteration over all lines from an input stream	73
nvwa::basic_mmap_byte_reader<_Tp>::iterator	Iterator over the bytes	73
nvwa::basic_mmap_line_reader<_Tp>::iterator	Iterator that contains the line content	74
nvwa::basic_split_view<_StringType, _DelimiterType>::iterator	Iterator over the split items	75
nvwa::istream_line_reader::iterator	Iterator that contains the line content	75
nvwa::file_line_reader::iterator	Iterator that contains the line content	75
nvwa::object_level_lock<_Host>::lock	Type that provides locking/unlocking semantics	78
nvwa::class_level_lock<_Host, _RealLock>::lock	Type that provides locking/unlocking semantics	78
nvwa::class_level_lock<_Host, false>::lock	Type that provides locking/unlocking semantics	79
nvwa::mem_pool_base	Base class for memory pools	79
nvwa::new_ptr_list_t	Structure to store the position information where <code>new</code> occurs	81
nvwa::object_level_lock<_Host>	Helper class for object-level locking	82
nvwa::optional<_Tp>	Class for optional values	83
nvwa::output_object<_OutputStrm, _StringType>	Functor to output objects pointed by a container of pointers	84
nvwa::smart_ptr<_Tp, _Policy>	Declaration of policy class to generate the smart pointer type	84
nvwa::smart_ptr<_Tp, storage_policy::shared>	Partial specialization to get <code>std::shared_ptr</code>	84
nvwa::smart_ptr<_Tp, storage_policy::unique>	Partial specialization to get <code>std::unique_ptr</code>	85
nvwa::static_mem_pool<_Sz, _Gid>	Singleton class template to manage the allocation/deallocation of memory blocks of one specific size	85
nvwa::static_mem_pool_set	Singleton class to maintain a set of existing instantiations of static_mem_pool	88
nvwa::tree<_Tp, _Policy>	Basic tree (node) class template that owns all its children	90

Chapter 4

File Index

4.1 File List

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

_nvw.h	Common definitions for preprocessing	91
bool_array.cpp	Code for class bool_array (packed boolean array)	92
bool_array.h	Header file for class bool_array (packed boolean array)	92
c++11.h	C++11 feature detection macros and workarounds	94
class_level_lock.h	In essence Loki ClassLevelLockable re-engineered to use a fast_mutex class	94
cont_ptr_utils.h	Utility functors for containers of pointers (adapted from Scott Meyers' <i>Effective STL</i>)	96
debug_new.cpp	Implementation of debug versions of new and delete to check leakage	97
debug_new.h	Header file for checking leaks caused by unmatched new/delete	107
fast_mutex.h	A fast mutex implementation for POSIX, Win32, and modern C++	111
fc_queue.h	Definition of a fixed-capacity queue	114
file_line_reader.cpp	Code for file_line_reader, an easy-to-use line-based file reader	115
file_line_reader.h	Header file for file_line_reader, an easy-to-use line-based file reader	116
fixed_mem_pool.h	Definition of a fixed-size memory pool template for structs/classes	117
functional.h	Utility templates for functional programming style	120
istream_line_reader.h	Header file for istream_line_reader, an easy-to-use line-based istream reader	123
mem_pool_base.cpp	Implementation for the memory pool base	124
mem_pool_base.h	Header file for the memory pool base	124
mmap_byte_reader.h	Header file for mmap_byte_reader, an easy-to-use byte-based file reader	126

mmap_line_reader.h	Header file for mmap_line_reader and mmap_line_reader_sv, easy-to-use line-based file readers	127
mmap_reader_base.h	Header file for mmap_reader_base, common base for mmap-based file readers	128
object_level_lock.h	In essence Loki ObjectLevelLockable re-engineered to use a fast_mutex class	129
pctimer.h	Function to get a high-resolution timer for Win32/Cygwin/Unix	130
set_assign.h	Definition of template functions set_assign_union and set_assign_difference	130
split.h	Header file for an efficient, lazy split function, when using ranges seems an overkill	131
static_mem_pool.cpp	Non-template and non-inline code for the 'static' memory pool	132
static_mem_pool.h	Header file for the 'static' memory pool	133
tree.h	A generic tree class template and the traversal utilities	137

Chapter 5

Namespace Documentation

5.1 nvwa Namespace Reference

Namespace of the nvwa project.

Classes

- class [bad_optional_access](#)
Class for bad optional access exception.
- class [basic_mmap_byte_reader](#)
Class template to allow iteration over all bytes of a mmappable file.
- class [basic_mmap_line_reader](#)
Class template to allow iteration over all lines of a mmappable file.
- class [basic_split_view](#)
Class to allow iteration over split items from the input.
- class [bool_array](#)
Class to represent a packed boolean array.
- class [breadth_first_iteration](#)
Iteration class for breadth-first traversal.
- class [class_level_lock](#)
Helper class for class-level locking.
- class [class_level_lock< _Host, false >](#)
Partial specialization that makes null locking.
- class [debug_new_counter](#)
Counter class for on-exit leakage check.
- class [debug_new_recorder](#)
Recorder class to remember the call context.
- struct [delete_object](#)
Functor to delete objects pointed by a container of pointers.
- class [depth_first_iteration](#)
Iteration class for depth-first traversal.
- struct [dereference](#)
Functor to return objects pointed by a container of pointers.
- struct [dereference_less](#)
Functor to compare objects pointed by a container of pointers.

- class [fast_mutex](#)
Class for non-reentrant fast mutexes.
- class [fast_mutex_autolock](#)
RAII lock class for [fast_mutex](#).
- class [fc_queue](#)
Class to represent a fixed-capacity queue.
- class [file_line_reader](#)
Class to allow iteration over all lines of a text file.
- class [fixed_mem_pool](#)
Class template to manipulate a fixed-size memory pool.
- class [in_order_iteration](#)
Iteration class for in-order traversal.
- class [istream_line_reader](#)
Class to allow iteration over all lines from an input stream.
- class [mem_pool_base](#)
Base class for memory pools.
- struct [new_ptr_list_t](#)
Structure to store the position information where `new` occurs.
- class [object_level_lock](#)
Helper class for object-level locking.
- class [optional](#)
Class for optional values.
- struct [output_object](#)
Functor to output objects pointed by a container of pointers.
- struct [smart_ptr](#)
Declaration of policy class to generate the smart pointer type.
- struct [smart_ptr< _Tp, storage_policy::shared >](#)
Partial specialization to get `std::shared_ptr`.
- struct [smart_ptr< _Tp, storage_policy::unique >](#)
Partial specialization to get `std::weak_ptr`.
- class [static_mem_pool](#)
Singleton class template to manage the allocation/deallocation of memory blocks of one specific size.
- class [static_mem_pool_set](#)
Singleton class to maintain a set of existing instantiations of [static_mem_pool](#).
- class [tree](#)
Basic tree (node) class template that owns all its children.

Typedefs

- typedef void(* [stacktrace_print_callback_t](#)) (FILE *fp, void **stacktrace)
Callback type for stack trace printing.
- typedef bool(* [leak_whitelist_callback_t](#)) (char const *file, int line, void *addr, void **stacktrace)
Callback type for the leak whitelist function.

Enumerations

- enum [storage_policy](#) { [storage_policy::unique](#), [storage_policy::shared](#) }
Policy class for how to store members.

Functions

- int [check_leaks](#) ()
Checks for memory leaks.
- int [check_mem_corruption](#) ()
Checks for heap corruption.
- static bool [print_position_from_addr](#) (const void *addr)
Tries printing the position information from an instruction address.
- static void [print_position](#) (const void *ptr, int line)
Prints the position information of a memory operation point.
- static void [print_stacktrace](#) (void **stacktrace)
Prints the stack backtrace.
- static bool [is_leak_whitelisted](#) (new_ptr_list_t *ptr)
Checks whether a leak should be ignored.
- static void * [alloc_mem](#) (size_t size, const char *file, int line, bool is_array)
Allocates memory and initializes control data.
- static void [free_pointer](#) (void *usr_ptr, void *addr, bool is_array)
Frees memory and adjusts pointers.
- void [swap](#) (bool_array &lhs, bool_array &rhs) noexcept
Exchanges the content of two bool_arrays.
- template<class _Tp, class _Alloc >
void [swap](#) (fc_queue< _Tp, _Alloc > &lhs, fc_queue< _Tp, _Alloc > &rhs) noexcept(noexcept(lhs.↵
swap(rhs)))
Exchanges the elements of two queues.
- template<typename _StringType, typename _DelimiterType >
constexpr [basic_split_view](#)< _StringType, _DelimiterType > [split](#) (const _StringType &src, _DelimiterType
delimiter) noexcept
Splits a string (or string_view) into lazy views.
- template<storage_policy _Policy = NVWA_TREE_DEFAULT_STORAGE_POLICY, typename _Tp >
[tree](#)< typename std::decay< _Tp >::type, _Policy >::tree_ptr [create_tree](#) (_Tp &&value)
Creates a tree without any children.
- template<storage_policy _Policy = NVWA_TREE_DEFAULT_STORAGE_POLICY, typename _Tp, typename... Args>
[tree](#)< typename std::decay< _Tp >::type, _Policy >::tree_ptr [create_tree](#) (_Tp &&value, Args &&... args)
Creates a tree with children.
- template<typename _Fn >
auto [lift_optional](#) (_Fn &&f)
Lifts a function so that it takes optionals and returns an optional.
- template<typename _Fn, typename... _Opt>
constexpr auto [apply](#) (_Fn &&f, _Opt &&... args) -> decltype(has_value(args...), [optional](#)< std::decay_t<
decltype(f(std::forward< _Opt >(args).value(...))>>())
Applies a function to the values of optionals if they are all valid.
- template<typename _Fn, class _Tuple >
constexpr auto [apply](#) (_Fn &&f, _Tuple &&t) -> decltype(detail::tuple_apply_impl(std::forward< _Fn >(f),
std::forward< _Tuple >(t), std::make_index_sequence< std::tuple_size< std::decay_t< _Tuple >>::value
>()))
Applies the function with all elements of the tuple as arguments.
- template<typename _Fn, typename _T1, typename _T2 >
constexpr auto [fmap](#) (_Fn &&f, const std::pair< _T1, _T2 > &args)
Applies a function to both elements of a pair, and makes the results a pair.
- template<typename _Fn, typename... _Targs>
constexpr auto [fmap](#) (_Fn &&f, const std::tuple< _Targs... > &args)
Applies a function to all elements of a tuple, and makes the results a tuple.

- `template<template< typename, typename > class _OutCont = std::vector, template< typename > class _Alloc = std::allocator, typename _Fn, class _Rng >`
`constexpr auto fmap (_Fn &&f, _Rng &&inputs) -> decltype(detail::adl_begin(inputs), detail::adl_end(inputs), _OutCont< std::decay_t< decltype(f(*detail::adl_begin(inputs)))>, _Alloc< std::decay_t< decltype(f(*detail::adl_begin(inputs)))>>>())`
Applies a function to each element in the input range.
- `template<typename _Rs, typename _Fn, typename... _Targs>`
`constexpr auto reduce (_Fn &&f, const std::tuple< _Targs... > &args, _Rs &&value)`
Applies a function cumulatively to all elements of a tuple.
- `template<typename _Fn, class _Rng >`
`constexpr auto reduce (_Fn &&f, _Rng &&inputs)`
Applies a function cumulatively to elements in the input range.
- `template<typename _Rs, typename _Fn, typename _Iter >`
`constexpr _Rs && reduce (_Fn &&f, _Rs &&value, _Iter begin, _Iter end)`
Applies a function cumulatively to a range.
- `template<typename _Rs, typename _Fn, class _Rng >`
`constexpr auto reduce (_Fn &&f, _Rng &&inputs, _Rs &&initval) -> decltype(f(initval, *detail::adl_begin(inputs)))`
Applies a function cumulatively to elements in the input range.
- `template<typename _T1, typename _T2, typename _Fn >`
`constexpr auto wrap_args_as_pair (_Fn &&f)`
Makes a two-argument function accept a pair instead.
- `template<typename _Tuple, typename _Fn >`
`constexpr auto wrap_args_as_tuple (_Fn &&f)`
Makes a function accept a tuple as its arguments.
- `template<typename _Tp >`
`constexpr _Tp pipeline (_Tp &&data)`
Returns the data intact to terminate the recursion.
- `template<typename _Tp, typename _Fn, typename... _Fargs>`
`decltype(auto) constexpr pipeline (_Tp &&data, _Fn &&f, _Fargs &&... args)`
Applies the functions in the arguments to the data consecutively.
- `auto compose ()`
Constructs a function (object) that composes the passed functions.
- `template<typename _Fn >`
`auto compose (_Fn f)`
Constructs a function (object) that composes the passed functions.
- `template<typename _Fn, typename... _Fargs>`
`auto compose (_Fn f, _Fargs... args)`
Constructs a function (object) that composes the passed functions.
- `template<typename _Rs, typename _Tp >`
`std::function< _Rs(_Tp)> fix_simple (std::function< _Rs(std::function< _Rs(_Tp)>, _Tp)> f)`
Generates the fixed point using the simple fixed-point combinator.
- `template<typename _Rs, typename _Tp >`
`std::function< _Rs(_Tp)> fix_simple (std::function< std::function< _Rs(_Tp)>(std::function< _Rs(_Tp)>> f)`
Generates the fixed point using the simple fixed-point combinator.
- `template<typename _Rs, typename _Tp >`
`std::function< _Rs(_Tp)> fix_curry (std::function< std::function< _Rs(_Tp)>(std::function< _Rs(_Tp)>> f)`
Generates the fixed point using the Curry-style fixed-point combinator.
- `template<typename _Rs, typename... _Targs>`
`auto make_curry (std::function< _Rs(_Targs...)> f)`
Makes a curried function.

- `template<typename _Rs, typename... _Targs>`
`auto make_curry (_Rs>(*f)(_Targs...))`
Makes a curried function.
- `template<typename _FnType, typename _Fn >`
`auto make_curry (_Fn &&f)`
Makes a curried function.

Variables

- `bool new_autocheck_flag = true`
Flag to control whether `nvwa::check_leaks` will be automatically called on program exit.
- `bool new_verbose_flag = false`
Flag to control whether verbose messages are output.
- `FILE * new_output_fp = stderr`
Pointer to the output stream.
- `const char * new_programname = _DEBUG_NEW_PROGNAME`
Pointer to the program name.
- `stacktrace_print_callback_t stacktrace_print_callback = nullptr`
Pointer to the callback used to print the stack backtrace in case of a memory problem.
- `leak_whitelist_callback_t leak_whitelist_callback = nullptr`
Pointer to the callback used to filter out false positives from leak reports.
- `static debug_new_counter __debug_new_count`
Counting object for each file including `debug_new.h`.
- `const size_t PLATFORM_MEM_ALIGNMENT = sizeof(size_t) * 2`
The platform memory alignment.
- `static const unsigned DEBUG_NEW_MAGIC = 0x4442474E`
Definition of the constant magic number used for error detection.
- `static const int ALIGNED_LIST_ITEM_SIZE = ALIGN(sizeof(new_ptr_list_t))`
The extra memory allocated by `operator new`.
- `static new_ptr_list_t new_ptr_list`
List of all new'd pointers.
- `static fast_mutex new_ptr_lock`
The mutex guard to protect simultaneous access to the pointer list.
- `static fast_mutex new_output_lock`
The mutex guard to protect simultaneous output to `new_output_fp`.
- `static size_t total_mem_alloc = 0`
Total memory allocated in bytes.
- `const size_t BUFFER_SIZE = 256`
Size of buffer.

5.1.1 Detailed Description

Namespace of the nvwa project.

Most functions and global variables are defined in this namespace.

5.1.2 Typedef Documentation

5.1.2.1 leak_whitelist_callback_t

```
typedef bool(* nvwa::leak_whitelist_callback_t) (char const *file, int line, void *addr, void
**stacktrace)
```

Callback type for the leak whitelist function.

file, *address*, and *backtrace* might be null depending on library configuration, platform, and amount of runtime information available. *line* can be 0 when line number info is not available at runtime.

Parameters

<i>file</i>	null-terminated string of the file name
<i>line</i>	line number
<i>addr</i>	address of code where leakage happens
<i>stacktrace</i>	pointer to the stack trace array (null-terminated)

Returns

`true` if the leak should be whitelisted; `false` otherwise

5.1.2.2 stacktrace_print_callback_t

```
typedef void(* nvwa::stacktrace_print_callback_t) (FILE *fp, void **stacktrace)
```

Callback type for stack trace printing.

Parameters

<i>fp</i>	pointer to the output stream
<i>stacktrace</i>	pointer to the stack trace array (null-terminated)

5.1.3 Enumeration Type Documentation

5.1.3.1 storage_policy

```
enum nvwa::storage_policy [strong]
```

Policy class for how to store members.

Enumerator

unique	Members are directly owned.
shared	Members may be shared and passed around.

5.1.4 Function Documentation

5.1.4.1 alloc_mem()

```
static void* nvwa::alloc_mem (
    size_t size,
    const char * file,
    int line,
    bool is_array ) [static]
```

Allocates memory and initializes control data.

Parameters

<i>size</i>	size of the required memory block
<i>file</i>	null-terminated string of the file name
<i>line</i>	line number
<i>is_array</i>	boolean value whether this is an array operation

Returns

pointer to the user-requested memory area; null if memory allocation is not successful

5.1.4.2 apply() [1/2]

```
template<typename _Fn , typename... _Opt>
constexpr auto nvwa::apply (
    _Fn && f,
    _Opt &&... args ) -> decltype( has_value(args...), optional< std::decay_t<decltype(f(std←
::forward<_Opt>(args).value()...))>>() )
```

Applies a function to the values of optionals if they are all valid.

If any of the optionals are invalid, the result is invalid too.

Parameters

<i>f</i>	the function to apply
<i>args</i>	optionals whose values will be passed to <i>f</i>

Returns

an optional that either is invalid (if any of the input is invalid) or contains the output of *f*

5.1.4.3 `apply()` [2/2]

```
template<typename _Fn , class _Tuple >
constexpr auto nvwa::apply (
    _Fn && f,
    _Tuple && t ) -> decltype(detail::tuple_apply_impl( std::forward<_Fn>(f), std::
::forward<_Tuple>(t), std::make_index_sequence< std::tuple_size<std::decay_t<_Tuple>>>←
::value>()))
```

Applies the function with all elements of the tuple as arguments.

It is exactly like the C++17 `std::apply`.

Parameters

<i>f</i>	the function to apply
<i>t</i>	the tuple that can expand to the function arguments

Returns

the result of the function applied

5.1.4.4 `check_leaks()`

```
int nvwa::check_leaks ( )
```

Checks for memory leaks.

Returns

zero if no leakage is found; the number of leaks otherwise

5.1.4.5 `check_mem_corruption()`

```
int nvwa::check_mem_corruption ( )
```

Checks for heap corruption.

Returns

zero if no problem is found; the number of found memory corruptions otherwise

5.1.4.6 compose() [1/3]

```
auto nvwa::compose ( )
```

Constructs a function (object) that composes the passed functions.

Returns

the forwarding function

5.1.4.7 compose() [2/3]

```
template<typename _Fn >  
auto nvwa::compose (   
    _Fn f )
```

Constructs a function (object) that composes the passed functions.

Parameters

<i>f</i>	the function to compose
----------	-------------------------

Returns

the function object that composes the passed function

5.1.4.8 compose() [3/3]

```
template<typename _Fn , typename... _Fargs>  
auto nvwa::compose (   
    _Fn f,  
    _Fargs... args )
```

Constructs a function (object) that composes the passed functions.

Parameters

<i>f</i>	the last function to compose
<i>args</i>	the other functions to compose

Returns

the function object that composes the passed functions

5.1.4.9 `create_tree()` [1/2]

```
template<storage_policy _Policy = NVWA_TREE_DEFAULT_STORAGE_POLICY, typename _Tp >
tree<typename std::decay<_Tp>::type, _Policy>::tree_ptr nvwa::create_tree (
    _Tp && value )
```

Creates a tree without any children.

Parameters

<i>value</i>	the value to assign to the tree node
--------------	--------------------------------------

Returns

the `unique_ptr` to the newly created tree

5.1.4.10 `create_tree()` [2/2]

```
template<storage_policy _Policy = NVWA_TREE_DEFAULT_STORAGE_POLICY, typename _Tp , typename...
Args>
tree<typename std::decay<_Tp>::type, _Policy>::tree_ptr nvwa::create_tree (
    _Tp && value,
    Args &&... args )
```

Creates a tree with children.

Parameters

<i>value</i>	the value to assign to the tree node
<i>args</i>	the <code>unique_ptr</code> s to children of the tree

Returns

the `unique_ptr` to the newly created tree

5.1.4.11 `fix_curry()`

```
template<typename _Rs , typename _Tp >
std::function<_Rs(_Tp)> nvwa::fix_curry (
    std::function< std::function< _Rs(_Tp)> (std::function< _Rs(_Tp)>)> f )
```

Generates the fixed point using the Curry-style fixed-point combinator.

This function takes a curried function of one parameter.

Parameters

<i>f</i>	the second-order function to combine with
----------	---

Returns

the fixed point

5.1.4.12 fix_simple() [1/2]

```
template<typename _Rs , typename _Tp >
std::function<_Rs(_Tp)> nvwa::fix_simple (
    std::function< _Rs(std::function< _Rs(_Tp)>, _Tp)> f )
```

Generates the fixed point using the simple fixed-point combinator.

This function takes a non-curried function of two parameters.

Parameters

<i>f</i>	the second-order function to combine with
----------	---

Returns

the fixed point

5.1.4.13 fix_simple() [2/2]

```
template<typename _Rs , typename _Tp >
std::function<_Rs(_Tp)> nvwa::fix_simple (
    std::function< std::function< _Rs(_Tp)>(std::function< _Rs(_Tp)>)> f )
```

Generates the fixed point using the simple fixed-point combinator.

This function takes a curried function of one parameter.

Parameters

<i>f</i>	the second-order function to combine with
----------	---

Returns

the fixed point

5.1.4.14 fmap() [1/3]

```
template<typename _Fn , typename _T1 , typename _T2 >
constexpr auto nvwa::fmap (
    _Fn && f,
    const std::pair< _T1, _T2 > & args )
```

Applies a function to both elements of a pair, and makes the results a pair.

Parameters

<i>f</i>	the function to apply
<i>args</i>	pair of arguments to pass

Returns

pair of results of function invocation

5.1.4.15 fmap() [2/3]

```
template<typename _Fn , typename... _Targs>
constexpr auto nvwa::fmap (
    _Fn && f,
    const std::tuple< _Targs... > & args )
```

Applies a function to all elements of a tuple, and makes the results a tuple.

Parameters

<i>f</i>	the function to apply
<i>args</i>	tuple of arguments to pass

Returns

tuple of results of function invocation

5.1.4.16 fmap() [3/3]

```
template<template< typename, typename > class _OutCont = std::vector, template< typename >
class _Alloc = std::allocator, typename _Fn , class _Rng >
constexpr auto nvwa::fmap (
    _Fn && f,
    _Rng && inputs ) -> decltype( detail::adl_begin(inputs), detail::adl_end(inputs),
    _OutCont< std::decay_t<decltype(f(*detail::adl_begin(inputs)))>, _Alloc<std::decay_t<decltype(f(*detail::adl_begin(inputs)))>>>())
```

Applies a function to each element in the input range.

This is similar to `std::transform`, but the style is more functional and more suitable for chaining operations.

Parameters

<i>f</i>	the function to apply
<i>inputs</i>	the input range

Precondition

f shall take one argument of the type of the elements in *inputs*, the output container shall support `push_back`, and the input range shall support iteration.

Returns

the container of results

5.1.4.17 `free_pointer()`

```
static void nvwa::free_pointer (
    void * usr_ptr,
    void * addr,
    bool is_array ) [static]
```

Frees memory and adjusts pointers.

Parameters

<i>usr_ptr</i>	pointer to the previously allocated memory
<i>addr</i>	pointer to the caller
<i>is_array</i>	flag indicating whether it is invoked by a <code>delete[]</code> call

5.1.4.18 `is_leak_whitelisted()`

```
static bool nvwa::is_leak_whitelisted (
    new_ptr_list_t * ptr ) [static]
```

Checks whether a leak should be ignored.

Its runtime performance depends on the callback [nvwa::leak_whitelist_callback](#).

Parameters

<i>ptr</i>	pointer to a new_ptr_list_t struct
------------	--

Returns

`true` if the leak should be whitelisted; `false` otherwise

5.1.4.19 lift_optional()

```
template<typename _Fn >
auto nvwa::lift_optional (
    _Fn && f )
```

Lifts a function so that it takes optionals and returns an optional.

Parameters

<code>f</code>	function to lift
----------------	------------------

Returns

the lifted function

5.1.4.20 make_curry() [1/3]

```
template<typename _Rs , typename... _Targs>
auto nvwa::make_curry (
    std::function< _Rs(_Targs...)> f )
```

Makes a curried function.

The returned function takes one argument at a time, and return a function that takes the next argument until all arguments are exhausted, in which case it returns the final result. This overload takes an `std::function` and can deduce its argument types and return type.

Parameters

<code>f</code>	the function to be curried as a <code>std::function</code>
----------------	--

Returns

the curried function

5.1.4.21 make_curry() [2/3]

```
template<typename _Rs , typename... _Targs>
auto nvwa::make_curry (
    _Rs(*) (_Targs...) f )
```

Makes a curried function.

The returned function takes one argument at a time, and return a function that takes the next argument until all arguments are exhausted, in which case it returns the final result. This overload takes a pointer to function and can deduce its argument types and return type.

Parameters

<i>f</i>	the function to be curried as a function pointer
----------	--

Returns

the curried function

5.1.4.22 make_curry() [3/3]

```
template<typename _FnType , typename _Fn >
auto nvwa::make_curry (
    _Fn && f )
```

Makes a curried function.

The returned function takes one argument at a time, and return a function that takes the next argument until all arguments are exhausted, in which case it returns the final result. This overload takes a generic function object, and the function type must be specified when this function template is instantiated.

Parameters

<i>f</i>	the function to be curried as a function pointer
----------	--

Returns

the curried function

5.1.4.23 pipeline()

```
template<typename _Tp , typename _Fn , typename... _Fargs>
decltype(auto) constexpr nvwa::pipeline (
    _Tp && data,
    _Fn && f,
    _Fargs &&... args )
```

Applies the functions in the arguments to the data consecutively.

Parameters

<i>data</i>	the data to operate on
<i>f</i>	the first function to apply
<i>args</i>	the rest functions to apply

5.1.4.24 `print_position()`

```
static void nvwa::print_position (
    const void * ptr,
    int line ) [static]
```

Prints the position information of a memory operation point.

When `_DEBUG_NEW_USE_ADDR2LINE` is defined to a non-zero value, this function will try to convert a given caller address to file/line information with *addr2line*.

Parameters

<i>ptr</i>	source file name if <i>line</i> is non-zero; caller address otherwise
<i>line</i>	source line number if non-zero; indication that <i>ptr</i> is the caller address otherwise

5.1.4.25 `print_position_from_addr()`

```
static bool nvwa::print_position_from_addr (
    const void * addr ) [static]
```

Tries printing the position information from an instruction address.

This is the version that uses *addr2line*.

Parameters

<i>addr</i>	the instruction address to convert and print
-------------	--

Returns

`true` if the address is converted successfully (and the result is printed); `false` if no useful information is got (and nothing is printed)

5.1.4.26 `print_stacktrace()`

```
static void nvwa::print_stacktrace (
    void ** stacktrace ) [static]
```

Prints the stack backtrace.

When `nvwa::stacktrace_print_callback` is not null, it is used for printing the stacktrace items. Default implementation of call stack printing is very spartan—only stack frame pointers are printed—but even that output is still useful. Just do address lookup in LLDB etc.

Parameters

<i>stacktrace</i>	pointer to the stack trace array
-------------------	----------------------------------

5.1.4.27 `reduce()` [1/4]

```
template<typename _Rs , typename _Fn , typename... _Targs>
constexpr auto nvwa::reduce (
    _Fn && f,
    const std::tuple< _Targs... > & args,
    _Rs && value )
```

Applies a function cumulatively to all elements of a tuple.

Parameters

<i>f</i>	the function to apply
<i>value</i>	the first argument to be passed to the function
<i>args</i>	the input tuple

Precondition

f shall take one argument of the result type, and one argument of the type of the elements in *args*.

5.1.4.28 `reduce()` [2/4]

```
template<typename _Fn , class _Rng >
constexpr auto nvwa::reduce (
    _Fn && f,
    _Rng && inputs )
```

Applies a function cumulatively to elements in the input range.

This is similar to `std::accumulate`, but the style is more functional and more suitable for chaining operations.

Parameters

<i>f</i>	the function to apply
<i>inputs</i>	the input range

Precondition

f shall take two arguments of the type of the elements in *inputs*, and the input range shall support iteration.

5.1.4.29 reduce() [3/4]

```
template<typename _Rs , typename _Fn , typename _Iter >
constexpr _Rs&& nvwa::reduce (
    _Fn && f,
    _Rs && value,
    _Iter begin,
    _Iter end )
```

Applies a function cumulatively to a range.

This is similar to `std::accumulate`, but the interface is different, and *f* is allowed to have arguments of different types (the first argument shall have the same type as the function return type). Perfect forwarding allows the result to be a reference type.

Parameters

<i>f</i>	the function to apply
<i>value</i>	the first argument to be passed to <i>f</i>
<i>begin</i>	beginning of the range
<i>end</i>	end of the range

Precondition

f shall take one argument of the result type, and one argument of the type of the elements in *inputs*, and the input range shall support iteration.

5.1.4.30 reduce() [4/4]

```
template<typename _Rs , typename _Fn , class _Rng >
constexpr auto nvwa::reduce (
    _Fn && f,
    _Rng && inputs,
    _Rs && initval ) -> decltype(f(initval, *detail::adl_begin(inputs)))
```

Applies a function cumulatively to elements in the input range.

This is similar to `std::accumulate`, but the style is more functional and more suitable for chaining operations. This implementation allows the two arguments of *f* to have different types (the first argument shall have the same type as the function return type). Perfect forwarding allows the result to be a reference type, and it is possible to write code like:

```
// Declaration of the output function
std::ostream& print(std::ostream& os, const my_type&);
...
// Dump all contents of the container_of_my_type to cout
nvwa::reduce(print, container_of_my_type, std::cout);
```


Parameters

<i>f</i>	the function to apply
<i>inputs</i>	the input range
<i>initval</i>	initial value for the cumulative calculation

Precondition

f shall take one argument of the result type, and one argument of the type of the elements in *inputs*, and the input range shall support iteration.

5.1.4.31 split()

```
template<typename _StringType , typename _DelimiterType >
constexpr basic_split_view<_StringType, _DelimiterType> nvwa::split (
    const _StringType & src,
    _DelimiterType delimiter ) [noexcept]
```

Splits a string (or string_view) into lazy views.

The source input shall remain unchanged when the generated `basic_split_view` is used in anyway.

Parameters

<i>src</i>	the source input to be split
<i>delimiter</i>	delimiter used to split <i>src</i> ; its type should be the same as that of <i>src</i> , or its character type

5.1.4.32 swap() [1/2]

```
void nvwa::swap (
    bool_array & lhs,
    bool_array & rhs ) [inline], [noexcept]
```

Exchanges the content of two bool_arrays.

Parameters

<i>lhs</i>	the first <code>bool_array</code> to exchange
<i>rhs</i>	the second <code>bool_array</code> to exchange

5.1.4.33 swap() [2/2]

```
template<class _Tp , class _Alloc >
```

```
void nvwa::swap (
    fc_queue< _Tp, _Alloc > & lhs,
    fc_queue< _Tp, _Alloc > & rhs ) [noexcept]
```

Exchanges the elements of two queues.

Parameters

<i>lhs</i>	the first queue to exchange
<i>rhs</i>	the second queue to exchange

Postcondition

If swapping the allocators does not throw, *lhs* will be swapped with *rhs*. If swapping the allocators throws with strong exception safety guarantee, this function will also provide such guarantee.

5.1.4.34 wrap_args_as_pair()

```
template<typename _T1 , typename _T2 , typename _Fn >
constexpr auto nvwa::wrap_args_as_pair (
    _Fn && f )
```

Makes a two-argument function accept a pair instead.

Parameters

<i>f</i>	a function that accepts two arguments
----------	---------------------------------------

Returns

a function that accepts a pair

5.1.4.35 wrap_args_as_tuple()

```
template<typename _Tuple , typename _Fn >
constexpr auto nvwa::wrap_args_as_tuple (
    _Fn && f )
```

Makes a function accept a tuple as its arguments.

The tuple shall contain exactly the same type/number of argument as the function needs.

Parameters

<i>f</i>	a function that accepts two arguments
----------	---------------------------------------

Returns

a function that accepts a pair

5.1.5 Variable Documentation

5.1.5.1 `__debug_new_count`

```
debug_new_counter nvwa::__debug_new_count [static]
```

Counting object for each file including [debug_new.h](#).

5.1.5.2 `BUFFER_SIZE`

```
const size_t nvwa::BUFFER_SIZE = 256
```

Size of buffer.

5.1.5.3 `leak_whitelist_callback`

```
leak_whitelist_callback_t nvwa::leak_whitelist_callback = nullptr
```

Pointer to the callback used to filter out false positives from leak reports.

A null value means the lack of filtering.

5.1.5.4 `new_output_fp`

```
FILE * nvwa::new_output_fp = stderr
```

Pointer to the output stream.

The default output is *stderr*, and one may change it to a user stream if needed (say, [new_verbose_flag](#) is `true` and there are a lot of (de)allocations).

5.1.5.5 `new_progname`

```
const char * nvwa::new_progname = _DEBUG_NEW_PROGNAME
```

Pointer to the program name.

Its initial value is the macro [_DEBUG_NEW_PROGNAME](#). You should try to assign the program path to it early in your application. Assigning `argv[0]` to it in *main* is one way. If you use *bash* or *ksh* (or similar), the following statement is probably what you want: `'new_progname = getenv("_");'`.

5.1.5.6 PLATFORM_MEM_ALIGNMENT

```
const size_t nvwa::PLATFORM_MEM_ALIGNMENT = sizeof(size_t) * 2
```

The platform memory alignment.

The current value works well in platforms I have tested: Windows XP, Windows 7 x64, and Mac OS X Leopard. It may be smaller than the real alignment, but must be bigger than `sizeof(size_t)` for it work. [nvwa::debug_new_recorder](#) uses it to detect misaligned pointer returned by `'new NonPODType[size]'`.

5.1.5.7 stacktrace_print_callback

```
stacktrace_print_callback_t nvwa::stacktrace_print_callback = nullptr
```

Pointer to the callback used to print the stack backtrace in case of a memory problem.

A null value causes the default stack trace printing routine to be used.

Chapter 6

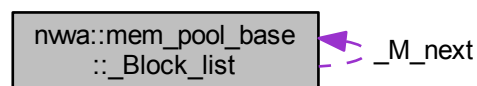
Class Documentation

6.1 `nwa::mem_pool_base::_Block_list` Struct Reference

Structure to store the next available memory block.

```
#include <mem_pool_base.h>
```

Collaboration diagram for `nwa::mem_pool_base::_Block_list`:



Public Attributes

- `_Block_list * _M_next`
Pointer to the next memory block.

6.1.1 Detailed Description

Structure to store the next available memory block.

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

- `mem_pool_base.h`

6.2 nvwa::bool_array::_Element<_Byte_type> Class Template Reference

Class to represent a reference to an array element.

Public Member Functions

- [_Element](#) ([_Byte_type](#) *ptr, [size_type](#) pos)
Constructs a reference to an array element.
- [bool operator=](#) (bool value)
Assigns a new boolean value to an array element.
- [operator bool](#) () const
Reads the boolean value from an array element.

6.2.1 Detailed Description

```
template<typename _Byte_type>
class nvwa::bool_array::_Element<_Byte_type>
```

Class to represent a reference to an array element.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 _Element()

```
template<typename _Byte_type >
nvwa::bool_array::_Element<_Byte_type>::_Element (
    _Byte_type * ptr,
    size_type pos ) [inline]
```

Constructs a reference to an array element.

Parameters

<i>ptr</i>	ptr to the internal boolean data
<i>pos</i>	position of the array element to access

6.2.3 Member Function Documentation

6.2.3.1 operator bool()

```
template<typename _Byte_type >
nvwa::bool_array::_Element< _Byte_type >::operator bool ( ) const [inline]
```

Reads the boolean value from an array element.

Returns

the boolean value of the accessed array element

6.2.3.2 operator=()

```
template<typename _Byte_type >
bool nvwa::bool_array::_Element< _Byte_type >::operator= (
    bool value ) [inline]
```

Assigns a new boolean value to an array element.

Parameters

<i>value</i>	the new boolean value
--------------	-----------------------

Returns

the assigned boolean value

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

- [bool_array.h](#)

6.3 nvwa::fixed_mem_pool<_Tp>::alignment Struct Reference

Specializable struct to define the alignment of an object in the [fixed_mem_pool](#).

```
#include <fixed_mem_pool.h>
```

6.3.1 Detailed Description

```
template<class _Tp>
struct nvwa::fixed_mem_pool<_Tp>::alignment
```

Specializable struct to define the alignment of an object in the [fixed_mem_pool](#).

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

- [fixed_mem_pool.h](#)

6.4 nvwa::bad_optional_access Class Reference

Class for bad optional access exception.

```
#include <functional.h>
```

Inherits `logic_error`.

6.4.1 Detailed Description

Class for bad optional access exception.

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

- [functional.h](#)

6.5 nvwa::basic_mmap_byte_reader< _Tp > Class Template Reference

Class template to allow iteration over all bytes of a mmappable file.

```
#include <mmap_byte_reader.h>
```

Inherits `nvwa::mmap_reader_base`.

Classes

- class [iterator](#)
Iterator over the bytes.

6.5.1 Detailed Description

```
template<typename _Tp>  
class nvwa::basic_mmap_byte_reader< _Tp >
```

Class template to allow iteration over all bytes of a mmappable file.

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

- [mmap_byte_reader.h](#)

6.6 nvwa::basic_mmap_line_reader< _Tp > Class Template Reference

Class template to allow iteration over all lines of a mmappable file.

```
#include <mmap_line_reader.h>
```

Inherits `nvwa::mmap_reader_base`.

Classes

- class [iterator](#)
Iterator that contains the line content.

Public Types

- enum [strip_type](#) { [strip_delimiter](#), [no_strip_delimiter](#) }
Enumeration of whether the delimiter should be stripped.

Public Member Functions

- bool [read](#) (_Tp &output, size_t &offset)
Reads content from the mmaped file.

6.6.1 Detailed Description

```
template<typename _Tp>
class nvwa::basic_mmap_line_reader<_Tp>
```

Class template to allow iteration over all lines of a mmappable file.

6.6.2 Member Enumeration Documentation

6.6.2.1 strip_type

```
template<typename _Tp>
enum nvwa::basic_mmap_line_reader::strip_type
```

Enumeration of whether the delimiter should be stripped.

Enumerator

strip_delimiter	The delimiter should be stripped.
no_strip_delimiter	The delimiter should be retained.

6.6.3 Member Function Documentation

6.6.3.1 read()

```
template<typename _Tp >
bool nvwa::basic_mmap_line_reader< _Tp >::read (
    _Tp & output,
    size_t & offset )
```

Reads content from the mmaped file.

Parameters

out	<i>output</i>	object to receive the line
in, out	<i>offset</i>	offset of reading pos on entry; end offset on exit

Returns

`true` if line content is returned; `false` otherwise

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

- [mmap_line_reader.h](#)

6.7 nvwa::basic_split_view< _StringType, _DelimiterType > Class Template Reference

Class to allow iteration over split items from the input.

```
#include <split.h>
```

Classes

- class [iterator](#)
Iterator over the split items.

Public Member Functions

- constexpr [basic_split_view](#) (const string_type &src, delimiter_type delimiter) noexcept
Constructor.
- std::vector< std::basic_string< char_type > > [to_vector](#) () const
Converts the view to a string vector.
- std::vector< std::basic_string_view< char_type > > [to_vector_sv](#) () const
Converts the view to a string_view vector.

6.7.1 Detailed Description

```
template<typename _StringType, typename _DelimiterType>
class nvwa::basic_split_view< _StringType, _DelimiterType >
```

Class to allow iteration over split items from the input.

Parameters

<i>_StringType</i>	either string or string_view (or something similar)
<i>_DelimiterType</i>	the same type as <i>_StringType</i> or its character type (other types may cause unpredictable results)

6.7.2 Constructor & Destructor Documentation

6.7.2.1 basic_split_view()

```
template<typename _StringType, typename _DelimiterType>
constexpr nvwa::basic_split_view<_StringType, _DelimiterType>::basic_split_view (
    const string_type & src,
    delimiter_type delimiter ) [inline], [explicit], [noexcept]
```

Constructor.

Parameters

<i>src</i>	the source input to be split
<i>delimiter</i>	delimiter used to split <i>src</i> ; its type should be the same as that of <i>src</i> , or its character type

6.7.3 Member Function Documentation

6.7.3.1 to_vector()

```
template<typename _StringType, typename _DelimiterType>
std::vector<std::basic_string<char_type>> > nvwa::basic_split_view<_StringType, _DelimiterType>::to_vector ( ) const [inline]
```

Converts the view to a string vector.

6.7.3.2 to_vector_sv()

```
template<typename _StringType, typename _DelimiterType>
std::vector<std::basic_string_view<char_type>> > nvwa::basic_split_view<_StringType, _DelimiterType>::to_vector_sv ( ) const [inline]
```

Converts the view to a string_view vector.

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

- [split.h](#)

6.8 nvwa::fixed_mem_pool<_Tp>::block_size Struct Reference

Struct to calculate the block size based on the (specializable) alignment value.

```
#include <fixed_mem_pool.h>
```

6.8.1 Detailed Description

```
template<class _Tp>
struct nvwa::fixed_mem_pool<_Tp>::block_size
```

Struct to calculate the block size based on the (specializable) alignment value.

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

- [fixed_mem_pool.h](#)

6.9 nvwa::bool_array Class Reference

Class to represent a packed boolean array.

```
#include <bool_array.h>
```

Classes

- [class _Element](#)
Class to represent a reference to an array element.

Public Types

- typedef unsigned long [size_type](#)
Type of array indices.
- typedef [_Element](#)< [byte](#) > [reference](#)
Type of reference.
- typedef [_Element](#)< const [byte](#) > [const_reference](#)
Type of const reference.

Public Member Functions

- [bool_array](#) () noexcept
Constructs an empty [bool_array](#).
- [bool_array](#) (size_type size)
Constructs a [bool_array](#) with a specific size.
- [bool_array](#) (const void *ptr, size_type size)
Constructs a [bool_array](#) from a given bitmap.
- [~bool_array](#) ()
Destroys the [bool_array](#) and releases memory.
- [bool_array](#) (const [bool_array](#) &rhs)
Copy-constructor.
- [bool_array](#) & operator= (const [bool_array](#) &rhs)
Assignment operator.
- bool create (size_type size) noexcept
Creates the packed boolean array with a specific size.
- void initialize (bool value) noexcept
Initializes all array elements to a specific value optimally.
- reference operator[] (size_type pos)
Creates a reference to an array element.
- const_reference operator[] (size_type pos) const
Creates a const reference to an array element.
- bool at (size_type pos) const
Reads the boolean value of an array element at a specified position.
- void reset (size_type pos)
*Resets an array element to *false* at a specified position.*
- void set (size_type pos)
*Sets an array element to *true* at a specified position.*
- size_type size () const noexcept
Gets the size of the [bool_array](#).
- size_type count () const noexcept
*Counts elements with a *true* value.*
- size_type count (size_type begin, size_type end=npos) const
*Counts elements with a *true* value in a specified range.*
- size_type find (bool value, size_type offset=0) const
Searches for the specified boolean value.
- size_type find (bool value, size_type offset, size_type count) const
Searches for the specified boolean value.
- size_type find_until (bool value, size_type begin, size_type end) const
Searches for the specified boolean value.
- void flip () noexcept
*Changes all *true* elements to *false*, and *false* ones to *true*.*
- void swap ([bool_array](#) &rhs) noexcept
Exchanges the content of this [bool_array](#) with another.
- void merge_and (const [bool_array](#) &rhs, size_type begin=0, size_type end=npos, size_type offset=0)
Merges elements of another [bool_array](#) with a logical AND.
- void merge_or (const [bool_array](#) &rhs, size_type begin=0, size_type end=npos, size_type offset=0)
Merges elements of another [bool_array](#) with a logical OR.
- void copy_to_bitmap (void *dest, size_type begin=0, size_type end=npos)
Copies the [bool_array](#) content as bitmap to a specified buffer.

Static Public Member Functions

- static `size_t` [get_num_bytes_from_bits](#) (`size_type` num_bits)
Converts the number of bits to number of bytes.

Static Public Attributes

- static const `size_type` [npos](#) = (`size_type`)-1
Constant representing 'not found'.

Private Types

- typedef unsigned char [byte](#)
Private definition of byte to avoid polluting the global namespace.

Private Member Functions

- `byte` [get_8bits](#) (`size_type` offset, `size_type` end) const
Retrieve contiguous 8 bits from the [bool_array](#).

Static Private Attributes

- static `byte` [_S_bit_count](#) [256]
Array that contains pre-calculated values how many 1-bits there are in a given byte.
- static `byte` [_S_bit_ordinal](#) [256]
Array that contains pre-calculated values which the first 1-bit is for a given byte.

6.9.1 Detailed Description

Class to represent a packed boolean array.

This was first written in April 1995, before I knew of any existing implementation of this kind of classes. Of course, the C++ Standard Template Library now demands an implementation of packed boolean array as `vector<bool>`, but the code here should still be useful for the following reasons:

1. Some compilers (like MSVC 6) did not implement this specialization (and they may not have a `bit_vector` either);
2. I included some additional member functions, like [initialize](#), [count](#), and [find](#), which should be useful;
3. My tests show that the code here is significantly FASTER than `vector<bool>` (and the normal boolean array) under MSVC versions 6/8/9 and GCC versions before 4.3 (while the `vector<bool>` implementations of MSVC 7.1 and GCC 4.3 have performance similar to that of [bool_array](#)).

6.9.2 Member Typedef Documentation

6.9.2.1 byte

```
typedef unsigned char nvwa::bool_array::byte [private]
```

Private definition of byte to avoid polluting the global namespace.

6.9.2.2 size_type

```
typedef unsigned long nvwa::bool_array::size_type
```

Type of array indices.

6.9.3 Constructor & Destructor Documentation

6.9.3.1 bool_array() [1/3]

```
nvwa::bool_array::bool_array (
    size_type size ) [explicit]
```

Constructs a `bool_array` with a specific size.

Parameters

<i>size</i>	size of the array
-------------	-------------------

Exceptions

<i>out_of_range</i>	<i>size</i> equals 0
<i>bad_alloc</i>	memory is insufficient

6.9.3.2 bool_array() [2/3]

```
nvwa::bool_array::bool_array (
    const void * ptr,
    size_type size )
```

Constructs a `bool_array` from a given bitmap.

Parameters

<i>ptr</i>	pointer to a bitmap
<i>size</i>	size of the array

Exceptions

<i>out_of_range</i>	<i>size</i> equals 0
<i>bad_alloc</i>	memory is insufficient

6.9.3.3 `bool_array()` [3/3]

```
nvwa::bool_array::bool_array (
    const bool_array & rhs )
```

Copy-constructor.

Parameters

<i>rhs</i>	the <code>bool_array</code> to copy from
------------	--

Exceptions

<i>bad_alloc</i>	memory is insufficient
------------------	------------------------

6.9.4 Member Function Documentation

6.9.4.1 `at()`

```
bool nvwa::bool_array::at (
    size_type pos ) const [inline]
```

Reads the boolean value of an array element at a specified position.

Parameters

<i>pos</i>	position of the array element to access
------------	---

Returns

the boolean value of the accessed array element

Exceptions

<i>out_of_range</i>	<i>pos</i> is greater than the size of the array
---------------------	--

6.9.4.2 copy_to_bitmap()

```
void nvwa::bool_array::copy_to_bitmap (
    void * dest,
    size_type begin = 0,
    size_type end = npos )
```

Copies the `bool_array` content as bitmap to a specified buffer.

The caller needs to ensure the destination buffer is big enough.

Parameters

<i>dest</i>	address of the destination buffer
<i>begin</i>	beginning of the range
<i>end</i>	end of the range (exclusive)

Exceptions

<i>out_of_range</i>	bad range for the source or the destination
---------------------	---

6.9.4.3 count() [1/2]

```
bool_array::size_type nvwa::bool_array::count ( ) const [noexcept]
```

Counts elements with a `true` value.

Returns

the count of `true` elements

6.9.4.4 count() [2/2]

```
bool_array::size_type nvwa::bool_array::count (
    size_type begin,
    size_type end = npos ) const
```

Counts elements with a `true` value in a specified range.

Parameters

<i>begin</i>	beginning of the range
<i>end</i>	end of the range (exclusive)

Returns

the count of `true` elements

Exceptions

<code>out_of_range</code>	the range [begin, end) is invalid
---------------------------	-----------------------------------

6.9.4.5 create()

```
bool nvwa::bool_array::create (
    size_type size ) [noexcept]
```

Creates the packed boolean array with a specific size.

Parameters

<code>size</code>	size of the array
-------------------	-------------------

Returns

`false` if `size` equals 0 or is too big, or if memory is insufficient; `true` if `size` has a suitable value and memory allocation is successful.

6.9.4.6 find() [1/2]

```
bool_array::size_type nvwa::bool_array::find (
    bool value,
    size_type offset = 0 ) const [inline]
```

Searches for the specified boolean value.

This function seaches from the specified position (default to beginning) to the end.

Parameters

<code>offset</code>	the position at which the search is to begin
<code>value</code>	the boolean value to find

Returns

position of the first value found if successful; `npos` otherwise

6.9.4.7 find() [2/2]

```
bool_array::size_type nvwa::bool_array::find (
    bool value,
    size_type offset,
    size_type count ) const [inline]
```

Searches for the specified boolean value.

This function accepts a range expressed in {position, count}.

Parameters

<i>offset</i>	the position at which the search is to begin
<i>count</i>	the number of bits to search
<i>value</i>	the boolean value to find

Returns

position of the first value found if successful; `npos` otherwise

Exceptions

<i>out_of_range</i>	<i>offset</i> and/or <i>count</i> is too big
---------------------	--

6.9.4.8 find_until()

```
bool_array::size_type nvwa::bool_array::find_until (
    bool value,
    size_type begin,
    size_type end ) const
```

Searches for the specified boolean value.

This function accepts a range expressed in [begin, end).

Parameters

<i>begin</i>	the position at which the search is to begin
<i>end</i>	the end position (exclusive) to stop searching
<i>value</i>	the boolean value to find

Returns

position of the first value found if successful; `npos` otherwise

Exceptions

<i>out_of_range</i>	the range [begin, end) is invalid
---------------------	-----------------------------------

6.9.4.9 `get_8bits()`

```
bool_array::byte nvwa::bool_array::get_8bits (
    size_type offset,
    size_type end ) const [private]
```

Retrieve contiguous 8 bits from the `bool_array`.

If fewer than 8 bits are available, the extra bits are undefined.

Parameters

<i>offset</i>	beginning position to retrieve the bits
<i>end</i>	end position beyond whose byte no bits will be taken

6.9.4.10 `get_num_bytes_from_bits()`

```
size_t nvwa::bool_array::get_num_bytes_from_bits (
    size_type num_bits ) [inline], [static]
```

Converts the number of bits to number of bytes.

Parameters

<i>num_bits</i>	number of bits
-----------------	----------------

Returns

number of bytes needed to store *num_bits* bits

6.9.4.11 `initialize()`

```
void nvwa::bool_array::initialize (
    bool value ) [noexcept]
```

Initializes all array elements to a specific value optionally.

Parameters

<i>value</i>	the boolean value to assign to all elements
--------------	---

6.9.4.12 merge_and()

```
void nvwa::bool_array::merge_and (
    const bool_array & rhs,
    size_type begin = 0,
    size_type end = npos,
    size_type offset = 0 )
```

Merges elements of another [bool_array](#) with a logical AND.

Parameters

<i>rhs</i>	another bool_array to merge
<i>begin</i>	beginning of the range in <i>rhs</i>
<i>end</i>	end of the range (exclusive) in <i>rhs</i>
<i>offset</i>	position to merge in this bool_array

Exceptions

<i>out_of_range</i>	bad range for the source or the destination
---------------------	---

6.9.4.13 merge_or()

```
void nvwa::bool_array::merge_or (
    const bool_array & rhs,
    size_type begin = 0,
    size_type end = npos,
    size_type offset = 0 )
```

Merges elements of another [bool_array](#) with a logical OR.

Parameters

<i>rhs</i>	another bool_array to merge
<i>begin</i>	beginning of the range in <i>rhs</i>
<i>end</i>	end of the range (exclusive) in <i>rhs</i>
<i>offset</i>	position to merge in this bool_array

Exceptions

<i>out_of_range</i>	bad range for the source or the destination
---------------------	---

6.9.4.14 operator=()

```
bool_array & nvwa::bool_array::operator= (
    const bool_array & rhs )
```

Assignment operator.

Parameters

<i>rhs</i>	the <code>bool_array</code> to copy from
------------	--

Exceptions

<i>bad_alloc</i>	memory is insufficient
------------------	------------------------

6.9.4.15 operator[]() [1/2]

```
bool_array::reference nvwa::bool_array::operator[] (
    size_type pos ) [inline]
```

Creates a reference to an array element.

Parameters

<i>pos</i>	position of the array element to access
------------	---

Returns

reference to the specified element

6.9.4.16 operator[]() [2/2]

```
bool_array::const_reference nvwa::bool_array::operator[] (
    size_type pos ) const [inline]
```

Creates a const reference to an array element.

Parameters

<i>pos</i>	position of the array element to access
------------	---

Returns

const reference to the specified element

6.9.4.17 reset()

```
void nvwa::bool_array::reset (
    size_type pos ) [inline]
```

Resets an array element to `false` at a specified position.

Parameters

<i>pos</i>	position of the array element to access
------------	---

Exceptions

<i>out_of_range</i>	<i>pos</i> is greater than the size of the array
---------------------	--

6.9.4.18 set()

```
void nvwa::bool_array::set (
    size_type pos ) [inline]
```

Sets an array element to `true` at a specified position.

Parameters

<i>pos</i>	position of the array element to access
------------	---

Exceptions

<i>out_of_range</i>	<i>pos</i> is greater than the size of the array
---------------------	--

6.9.4.19 size()

```
bool_array::size_type nvwa::bool_array::size ( ) const [inline], [noexcept]
```

Gets the size of the `bool_array`.

Returns

the number of bits of the `bool_array`

6.9.4.20 swap()

```
void nvwa::bool_array::swap (
    bool_array & rhs ) [noexcept]
```

Exchanges the content of this `bool_array` with another.

Parameters

<i>rhs</i>	another <code>bool_array</code> to exchange content with
------------	--

6.9.5 Member Data Documentation

6.9.5.1 _S_bit_ordinal

```
bool_array::byte nvwa::bool_array::_S_bit_ordinal [static], [private]
```

Array that contains pre-calculated values which the first 1-bit is for a given byte.

The first element indicates an invalid value (there are only 0-bits).

6.9.5.2 npos

```
const size_type nvwa::bool_array::npos = (size_type)-1 [static]
```

Constant representing 'not found'.

The documentation for this class was generated from the following files:

- `bool_array.h`
- `bool_array.cpp`

6.10 nvwa::breadth_first_iteration<_Tree> Class Template Reference

Iteration class for breadth-first traversal.

```
#include <tree.h>
```


6.10.1 Detailed Description

```
template<typename _Tree>
class nvwa::breadth_first_iteration< _Tree >
```

Iteration class for breadth-first traversal.

Mutating (adding or removing children) or removing the part of the tree nodes at the current traversal level has undefined behaviour.

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

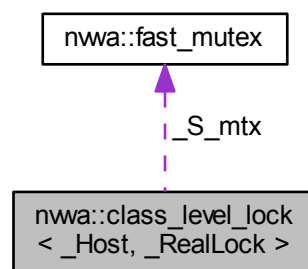
- [tree.h](#)

6.11 `nvwa::class_level_lock< _Host, _RealLock >` Class Template Reference

Helper class for class-level locking.

```
#include <class_level_lock.h>
```

Collaboration diagram for `nvwa::class_level_lock< _Host, _RealLock >`:



Classes

- class [lock](#)
Type that provides locking/unlocking semantics.

6.11.1 Detailed Description

```
template<class _Host, bool _RealLock = true>
class nvwa::class_level_lock< _Host, _RealLock >
```

Helper class for class-level locking.

This is the multi-threaded implementation. The main departure from Loki `ClassLevelLockable` is that there is an additional template parameter which can make the lock not lock at all even in multi-threaded environments. See [static_mem_pool.h](#) for real usage.

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

- [class_level_lock.h](#)

6.12 nvwa::class_level_lock< _Host, false > Class Template Reference

Partial specialization that makes null locking.

```
#include <class_level_lock.h>
```

Classes

- class [lock](#)
Type that provides locking/unlocking semantics.

6.12.1 Detailed Description

```
template<class _Host>  
class nvwa::class_level_lock< _Host, false >
```

Partial specialization that makes null locking.

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

- [class_level_lock.h](#)

6.13 nvwa::debug_new_counter Class Reference

Counter class for on-exit leakage check.

```
#include <debug_new.h>
```

Public Member Functions

- [debug_new_counter](#) ()
Constructor to increment the count.
- [~debug_new_counter](#) ()
Destructor to decrement the count.

Static Private Attributes

- static int [_S_count](#) = 0
Count of source files that use debug_new.

6.13.1 Detailed Description

Counter class for on-exit leakage check.

This technique is learnt from *The C++ Programming Language* by Bjarne Stroustrup.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 ~debug_new_counter()

```
nvwa::debug_new_counter::~~debug_new_counter ( )
```

Destructor to decrement the count.

When the count is zero, [nvwa::check_leaks](#) will be called.

The documentation for this class was generated from the following files:

- [debug_new.h](#)
- [debug_new.cpp](#)

6.14 nvwa::debug_new_recorder Class Reference

Recorder class to remember the call context.

```
#include <debug_new.h>
```

Public Member Functions

- [debug_new_recorder](#) (const char *file, int line)
Constructor to remember the call context.
- `template<class _Tp >`
`_Tp * operator-> * (_Tp *ptr)`
Operator to write the context information to memory.

Private Member Functions

- `void _M_process (void *ptr)`
Processes the allocated memory and inserts file/line informatin.

6.14.1 Detailed Description

Recorder class to remember the call context.

The idea comes from [Greg Herlihy's post](#) in comp.lang.c++.moderated.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 debug_new_recorder()

```
nvwa::debug_new_recorder::debug_new_recorder (
    const char * file,
    int line ) [inline]
```

Constructor to remember the call context.

The information will be used in `debug_new_recorder::operator->*`.

6.14.3 Member Function Documentation

6.14.3.1 _M_process()

```
void nvwa::debug_new_recorder::_M_process (
    void * usr_ptr ) [private]
```

Processes the allocated memory and inserts file/line informatin.

It will only be done when it can ensure the memory is allocated by one of our operator new variants.

Parameters

<code><i>usr_ptr</i></code>	pointer returned by a new-expression
-----------------------------	--------------------------------------

6.14.3.2 operator-> *()

```
template<class _Tp >
_Tp* nvwa::debug_new_recorder::operator->* (
    _Tp * ptr ) [inline]
```

Operator to write the context information to memory.

`operator->*` is chosen because it has the right precedence, it is rarely used, and it looks good: so people can tell the special usage more quickly.

The documentation for this class was generated from the following files:

- [debug_new.h](#)
- [debug_new.cpp](#)

6.15 nvwa::delete_object Struct Reference

Functor to delete objects pointed by a container of pointers.

```
#include <cont_ptr_utils.h>
```

6.15.1 Detailed Description

Functor to delete objects pointed by a container of pointers.

A typical usage might be like:

```
list<Object*> l;  
...  
for_each(l.begin(), l.end(), delete_object());
```

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

- [cont_ptr_utils.h](#)

6.16 nvwa::depth_first_iteration<_Tree> Class Template Reference

Iteration class for depth-first traversal.

```
#include <tree.h>
```

6.16.1 Detailed Description

```
template<typename _Tree>  
class nvwa::depth_first_iteration<_Tree>
```

Iteration class for depth-first traversal.

Mutating (adding or removing children) or removing the part of the tree nodes that have already been traversed has undefined behaviour.

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

- [tree.h](#)

6.17 nvwa::dereference Struct Reference

Functor to return objects pointed by a container of pointers.

```
#include <cont_ptr_utils.h>
```

6.17.1 Detailed Description

Functor to return objects pointed by a container of pointers.

A typical usage might be like:

```
vector<Object*> v;  
...  
transform(v.begin(), v.end(),  
          ostream_iterator<Object>(cout, " "),  
          dereference());
```

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

- [cont_ptr_utils.h](#)

6.18 nvwa::dereference_less Struct Reference

Functor to compare objects pointed by a container of pointers.

```
#include <cont_ptr_utils.h>
```

6.18.1 Detailed Description

Functor to compare objects pointed by a container of pointers.

```
vector<Object*> v;  
...  
sort(v.begin(), v.end(), dereference_less());
```

or

```
set<Object*, dereference_less> s;
```

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

- [cont_ptr_utils.h](#)

6.19 nvwa::fast_mutex Class Reference

Class for non-reentrant fast mutexes.

```
#include <fast_mutex.h>
```

6.19.1 Detailed Description

Class for non-reentrant fast mutexes.

This is the implementation for POSIX threads.

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

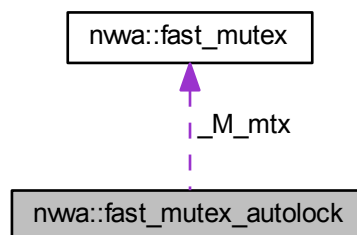
- [fast_mutex.h](#)

6.20 `nwa::fast_mutex_autolock` Class Reference

RAII lock class for [fast_mutex](#).

```
#include <fast_mutex.h>
```

Collaboration diagram for `nwa::fast_mutex_autolock`:



6.20.1 Detailed Description

RAII lock class for [fast_mutex](#).

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

- [fast_mutex.h](#)

6.21 `nwa::fc_queue<_Tp, _Alloc>` Class Template Reference

Class to represent a fixed-capacity queue.

```
#include <fc_queue.h>
```

Public Member Functions

- `fc_queue ()` `noexcept(noexcept(allocator_type()))`
Default-constructor that creates an empty queue.
- `fc_queue (size_type max_size, const allocator_type &alloc=allocator_type())`
Constructor that creates the queue with a maximum size (capacity).
- `fc_queue (const fc_queue &rhs)`
Copy-constructor that copies all elements from another queue.
- `fc_queue (fc_queue &&rhs) noexcept(noexcept(allocator_type(std::declval< allocator_type &>())))`
Move-constructor that moves all elements from another queue.
- `~fc_queue () noexcept(noexcept(std::declval< _Tp *>() ->~_Tp()))`
Destructor.
- `fc_queue & operator= (const fc_queue &rhs)`
Assignment operator that copies all elements from another queue.
- `fc_queue & operator= (fc_queue &&rhs) noexcept(noexcept(allocator_type(std::declval< allocator_type &>())))`
Assignment operator that moves all elements from another queue.
- `bool empty () const noexcept`
Checks whether the queue is empty (containing no elements).
- `bool full () const noexcept`
Checks whether the queue is full (containing the maximum allowed elements).
- `size_type capacity () const noexcept`
Gets the maximum number of allowed elements in the queue.
- `size_type size () const noexcept`
Gets the number of existing elements in the queue.
- `reference front ()`
Gets the first element in the queue.
- `const_reference front () const`
Gets the first element in the queue.
- `reference back ()`
Gets the last element in the queue.
- `const_reference back () const`
Gets the last element in the queue.
- `template<typename... _Targs>`
`void push (_Targs &&... args)`
Inserts a new element at the end of the queue.
- `void pop ()`
Discards the first element in the queue.
- `bool contains (const value_type &value) const`
Checks whether the queue contains a specific element.
- `void swap (fc_queue &rhs) noexcept(noexcept(std::swap(std::declval< allocator_type &>(), std::declval< allocator_type &>())))`
Exchanges the elements of two queues.
- `allocator_type get_allocator () const`
Gets the allocator of the queue.

6.21.1 Detailed Description

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
class nvwa::fc_queue<_Tp, _Alloc>
```

Class to represent a fixed-capacity queue.

This class has an interface close to `std::queue`, but it allows very efficient and lockless one-producer, one-consumer access, as long as the producer does not try to queue an element when the queue is already full.

Parameters

<code>_Tp</code>	the type of elements in the queue
<code>_Alloc</code>	allocator to use for memory management

Precondition

`_Tp` shall be `CopyConstructible` and `Destructible`, and `_Alloc` shall meet the allocator requirements (Table 28 in the C++11 spec).

6.21.2 Constructor & Destructor Documentation

6.21.2.1 `fc_queue()` [1/4]

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
nvwa::fc_queue< _Tp, _Alloc >::fc_queue ( ) [inline], [noexcept]
```

Default-constructor that creates an empty queue.

It is not very useful, except as the target of an assignment.

Postcondition

The following conditions will hold:

- `empty()`
- `full()`
- `capacity() == 0`
- `size() == 0`

6.21.2.2 `fc_queue()` [2/4]

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
nvwa::fc_queue< _Tp, _Alloc >::fc_queue (
    size_type max_size,
    const allocator_type & alloc = allocator_type() ) [inline], [explicit]
```

Constructor that creates the queue with a maximum size (capacity).

Parameters

<code>max_size</code>	the maximum size allowed
<code>alloc</code>	the allocator to use

Precondition

max_size shall be not be zero.

Postcondition

Unless memory allocation throws an exception, this queue will be constructed with the specified maximum size, and the following conditions will hold:

- `empty()`
- `! full()`
- `capacity() == max_size`
- `size() == 0`
- `get_allocator() == alloc`

6.21.2.3 `fc_queue()` [3/4]

```
template<class _Tp , class _Alloc >
nvwa::fc_queue< _Tp, _Alloc >::fc_queue (
    const fc_queue< _Tp, _Alloc > & rhs )
```

Copy-constructor that copies all elements from another queue.

Parameters

<i>rhs</i>	the queue to copy
------------	-------------------

Postcondition

If copy-construction is successful (no exception is thrown during memory allocation and element copy), this queue will have the same elements as *rhs*.

6.21.2.4 `fc_queue()` [4/4]

```
template<class _Tp , class _Alloc >
nvwa::fc_queue< _Tp, _Alloc >::fc_queue (
    fc_queue< _Tp, _Alloc > && rhs ) [noexcept]
```

Move-constructor that moves all elements from another queue.

Parameters

<i>rhs</i>	the queue to move from
------------	------------------------

Postcondition

If the allocator does not throw on move, this queue will have the same elements as the original *rhs*.

6.21.2.5 ~fc_queue()

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
nvwa::fc_queue< _Tp, _Alloc >::~~fc_queue ( ) [inline], [noexcept]
```

Destructor.

It erases all elements and frees memory.

6.21.3 Member Function Documentation**6.21.3.1 back() [1/2]**

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
reference nvwa::fc_queue< _Tp, _Alloc >::back ( ) [inline]
```

Gets the last element in the queue.

Precondition

the queue is not empty

Returns

reference to the last element

6.21.3.2 back() [2/2]

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
const_reference nvwa::fc_queue< _Tp, _Alloc >::back ( ) const [inline]
```

Gets the last element in the queue.

Precondition

the queue is not empty

Returns

const reference to the last element

6.21.3.3 capacity()

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
size_type nvwa::fc_queue< _Tp, _Alloc >::capacity ( ) const [inline], [noexcept]
```

Gets the maximum number of allowed elements in the queue.

Returns

the maximum number of allowed elements in the queue

6.21.3.4 contains()

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
bool nvwa::fc_queue< _Tp, _Alloc >::contains (
    const value_type & value ) const [inline]
```

Checks whether the queue contains a specific element.

Parameters

<i>value</i>	the value to be compared
--------------	--------------------------

Precondition

`value_type` shall be EqualityComparable.

Returns

true if found; false otherwise

6.21.3.5 empty()

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
bool nvwa::fc_queue< _Tp, _Alloc >::empty ( ) const [inline], [noexcept]
```

Checks whether the queue is empty (containing no elements).

Returns

true if it is empty; false otherwise

6.21.3.6 front() [1/2]

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
reference nvwa::fc_queue<_Tp, _Alloc >::front ( ) [inline]
```

Gets the first element in the queue.

Precondition

the queue is not empty

Returns

reference to the first element

6.21.3.7 front() [2/2]

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
const_reference nvwa::fc_queue<_Tp, _Alloc >::front ( ) const [inline]
```

Gets the first element in the queue.

Precondition

the queue is not empty

Returns

const reference to the first element

6.21.3.8 full()

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
bool nvwa::fc_queue<_Tp, _Alloc >::full ( ) const [inline], [noexcept]
```

Checks whether the queue is full (containing the maximum allowed elements).

Returns

true if it is full; false otherwise

6.21.3.9 get_allocator()

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
allocator_type nvwa::fc_queue< _Tp, _Alloc >::get_allocator ( ) const [inline]
```

Gets the allocator of the queue.

Returns

the allocator of the queue

6.21.3.10 operator=() [1/2]

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
fc_queue& nvwa::fc_queue< _Tp, _Alloc >::operator= (
    const fc_queue< _Tp, _Alloc > & rhs ) [inline]
```

Assignment operator that copies all elements from another queue.

Parameters

<i>rhs</i>	the queue to copy
------------	-------------------

Postcondition

If assignment is successful (no exception is thrown during memory allocation and element copy), this queue will have the same elements as *rhs*. Otherwise this queue is unchanged (strong exception safety is guaranteed).

6.21.3.11 operator=() [2/2]

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
fc_queue& nvwa::fc_queue< _Tp, _Alloc >::operator= (
    fc_queue< _Tp, _Alloc > && rhs ) [inline], [noexcept]
```

Assignment operator that moves all elements from another queue.

Parameters

<i>rhs</i>	the queue to move from
------------	------------------------

Postcondition

If assignment is successful (no exception is thrown during memory allocation and element copy), this queue will have the same elements as *rhs*. Otherwise this queue is unchanged (strong exception safety is guaranteed).

6.21.3.12 pop()

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
void nvwa::fc_queue<_Tp, _Alloc >::pop ( ) [inline]
```

Discards the first element in the queue.

Precondition

This queue is not empty.

Postcondition

One element is discarded at the front, `size()` is decremented by one, and `full()` is false.

6.21.3.13 push()

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
template<typename... _Targs>
void nvwa::fc_queue<_Tp, _Alloc >::push (
    _Targs &&... args ) [inline]
```

Inserts a new element at the end of the queue.

The first element will be discarded if the queue is full.

Parameters

<code>args</code>	arguments to construct a new element
-------------------	--------------------------------------

Precondition

`capacity() > 0`

Postcondition

`size() <= capacity() && back() == value`, unless an exception is thrown, in which case this queue is unchanged (strong exception safety is guaranteed).

6.21.3.14 size()

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
size_type nvwa::fc_queue<_Tp, _Alloc >::size ( ) const [inline], [noexcept]
```

Gets the number of existing elements in the queue.

Returns

the number of existing elements in the queue

6.21.3.15 swap()

```
template<class _Tp, class _Alloc = std::allocator<_Tp>>
void nvwa::fc_queue< _Tp, _Alloc >::swap (
    fc_queue< _Tp, _Alloc > & rhs ) [inline], [noexcept]
```

Exchanges the elements of two queues.

Parameters

<i>rhs</i>	the queue to exchange with
------------	----------------------------

Postcondition

If swapping the allocators does not throw, **this* will be swapped with *rhs*. If swapping the allocators throws with strong exception safety guarantee, this function will also provide such guarantee.

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

- [fc_queue.h](#)

6.22 nvwa::file_line_reader Class Reference

Class to allow iteration over all lines of a text file.

```
#include <file_line_reader.h>
```

Classes

- class [iterator](#)

Iterator that contains the line content.

Public Types

- enum [strip_type](#) { [strip_delimiter](#), [no_strip_delimiter](#) }

Enumeration of whether the delimiter should be stripped.

Public Member Functions

- [file_line_reader](#) (FILE *stream, char delimiter='\n', [strip_type](#) strip=[strip_delimiter](#))
Constructor.
- [~file_line_reader](#) ()
Destructor.
- bool [read](#) (char *&output, size_t &size, size_t &capacity)
Reads content from the file stream.

6.22.1 Detailed Description

Class to allow iteration over all lines of a text file.

6.22.2 Member Enumeration Documentation

6.22.2.1 strip_type

```
enum nvwa::file_line_reader::strip_type
```

Enumeration of whether the delimiter should be stripped.

Enumerator

strip_delimiter	The delimiter should be stripped.
no_strip_delimiter	The delimiter should be retained.

6.22.3 Constructor & Destructor Documentation

6.22.3.1 file_line_reader()

```
nvwa::file_line_reader::file_line_reader (
    FILE * stream,
    char delimiter = '\n',
    strip\_type strip = strip\_delimiter ) [explicit]
```

Constructor.

Parameters

<i>stream</i>	the file stream to read from
<i>delimiter</i>	the delimiter between text 'lines' (default to LF)
<i>strip</i>	enumerator about whether to strip the delimiter

6.22.3.2 ~file_line_reader()

```
nvwa::file_line_reader::~~file_line_reader ( )
```

Destructor.

6.22.4 Member Function Documentation

6.22.4.1 read()

```
bool nvwa::file_line_reader::read (
    char *& output,
    size_t & size,
    size_t & capacity )
```

Reads content from the file stream.

If necessary, the receiving buffer will be expanded so that it is big enough to contain all the line content.

Parameters

in, out	<i>output</i>	initial receiving buffer
out	<i>size</i>	size of the line
in, out	<i>capacity</i>	capacity of the initial receiving buffer on entering the function; it can be increased when necessary

Returns

`true` if line content is returned; `false` otherwise

The documentation for this class was generated from the following files:

- [file_line_reader.h](#)
- [file_line_reader.cpp](#)

6.23 nvwa::fixed_mem_pool< _Tp > Class Template Reference

Class template to manipulate a fixed-size memory pool.

```
#include <fixed_mem_pool.h>
```

Classes

- struct [alignment](#)
Specializable struct to define the alignment of an object in the [fixed_mem_pool](#).
- struct [block_size](#)
Struct to calculate the block size based on the (specializable) alignment value.

Static Public Member Functions

- static void * [allocate](#) ()
Allocates a memory block from the memory pool.
- static void [deallocate](#) (void *)
Deallocates a memory block and returns it to the memory pool.
- static bool [initialize](#) (size_t size)
Initializes the memory pool.
- static int [deinitialize](#) ()
Deinitializes the memory pool.
- static int [get_alloc_count](#) ()
Gets the allocation count.
- static bool [is_initialized](#) ()
Is the memory pool initialized?

Static Protected Member Functions

- static bool [bad_alloc_handler](#) ()
Bad allocation handler.

Static Private Attributes

- static void * [_S_mem_pool_ptr](#) = nullptr
Pointer to the allocated chunk of memory.
- static void * [_S_first_avail_ptr](#) = nullptr
Pointer to the first available memory block.
- static int [_S_alloc_cnt](#) = 0
Count of allocations.

6.23.1 Detailed Description

```
template<class _Tp>
class nvwa::fixed_mem_pool<_Tp>
```

Class template to manipulate a fixed-size memory pool.

Please notice that only allocate and deallocate are protected by a lock.

Parameters

<code>_Tp</code>	class to use the fixed_mem_pool
------------------	---

6.23.2 Member Function Documentation

6.23.2.1 `allocate()`

```
template<class _Tp >
void * nvwa::fixed_mem_pool< _Tp >::allocate ( ) [inline], [static]
```

Allocates a memory block from the memory pool.

Returns

pointer to the allocated memory block

6.23.2.2 `bad_alloc_handler()`

```
template<class _Tp >
bool nvwa::fixed_mem_pool< _Tp >::bad_alloc_handler ( ) [static], [protected]
```

Bad allocation handler.

Called when there are no memory blocks available in the memory pool. If this function returns `false` (default behaviour if not explicitly specialized), it indicates that it can do nothing and `allocate()` should return null; if this function returns `true`, it indicates that it has freed some memory blocks and `allocate()` should try allocating again.

6.23.2.3 `deallocate()`

```
template<class _Tp >
void nvwa::fixed_mem_pool< _Tp >::deallocate (
    void * block_ptr ) [inline], [static]
```

Deallocates a memory block and returns it to the memory pool.

Parameters

<code>block_ptr</code>	pointer to the memory block to return
------------------------	---------------------------------------

6.23.2.4 `deinitialize()`

```
template<class _Tp >
int nvwa::fixed_mem_pool< _Tp >::deinitialize ( ) [static]
```

Deinitializes the memory pool.

Returns

0 if all memory blocks are returned and the memory pool successfully freed; or a non-zero value indicating number of memory blocks still in allocation

6.23.2.5 get_alloc_count()

```
template<class _Tp >
int nvwa::fixed_mem_pool< _Tp >::get_alloc_count ( ) [inline], [static]
```

Gets the allocation count.

Returns

the number of memory blocks still in allocation

6.23.2.6 initialize()

```
template<class _Tp >
bool nvwa::fixed_mem_pool< _Tp >::initialize (
    size_t size ) [static]
```

Initializes the memory pool.

Parameters

<i>size</i>	number of memory blocks to put in the memory pool
-------------	---

Returns

true if successful; false if memory insufficient

6.23.2.7 is_initialized()

```
template<class _Tp >
bool nvwa::fixed_mem_pool< _Tp >::is_initialized ( ) [inline], [static]
```

Is the memory pool initialized?

Returns

true if it is successfully initialized; false otherwise

6.23.3 Member Data Documentation

6.23.3.1 `_S_alloc_cnt`

```
template<class _Tp >
int nvwa::fixed_mem_pool< _Tp >::_S_alloc_cnt = 0 [static], [private]
```

Count of allocations.

6.23.3.2 `_S_first_avail_ptr`

```
template<class _Tp >
void * nvwa::fixed_mem_pool< _Tp >::_S_first_avail_ptr = nullptr [static], [private]
```

Pointer to the first available memory block.

6.23.3.3 `_S_mem_pool_ptr`

```
template<class _Tp >
void * nvwa::fixed_mem_pool< _Tp >::_S_mem_pool_ptr = nullptr [static], [private]
```

Pointer to the allocated chunk of memory.

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

- [fixed_mem_pool.h](#)

6.24 `nvwa::in_order_iteration< _Tree >` Class Template Reference

Iteration class for in-order traversal.

```
#include <tree.h>
```

6.24.1 Detailed Description

```
template<typename _Tree>
class nvwa::in_order_iteration< _Tree >
```

Iteration class for in-order traversal.

Mutating (adding or removing children) or removing the part of the tree nodes that have already been traversed has undefined behaviour.

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

- [tree.h](#)

6.25 nvwa::istream_line_reader Class Reference

Class to allow iteration over all lines from an input stream.

```
#include <istream_line_reader.h>
```

Classes

- class [iterator](#)

Iterator that contains the line content.

6.25.1 Detailed Description

Class to allow iteration over all lines from an input stream.

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

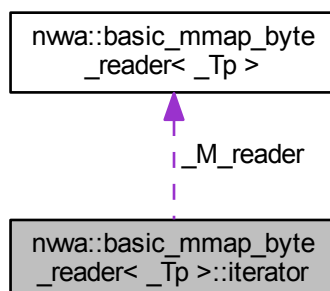
- [istream_line_reader.h](#)

6.26 nvwa::basic_mmap_byte_reader< _Tp >::iterator Class Reference

Iterator over the bytes.

```
#include <mmap_byte_reader.h>
```

Collaboration diagram for nvwa::basic_mmap_byte_reader< _Tp >::iterator:



6.26.1 Detailed Description

```
template<typename _Tp>
class nvwa::basic_mmap_byte_reader< _Tp >::iterator
```

Iterator over the bytes.

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

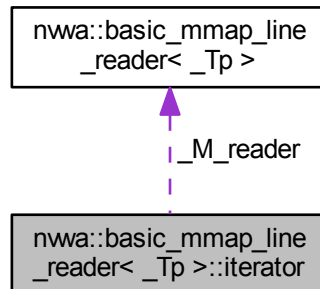
- [mmap_byte_reader.h](#)

6.27 nvwa::basic_mmap_line_reader< _Tp >::iterator Class Reference

Iterator that contains the line content.

```
#include <mmap_line_reader.h>
```

Collaboration diagram for nvwa::basic_mmap_line_reader< _Tp >::iterator:



6.27.1 Detailed Description

```
template<typename _Tp>
class nvwa::basic_mmap_line_reader< _Tp >::iterator
```

Iterator that contains the line content.

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

- [mmap_line_reader.h](#)

6.28 nvwa::basic_split_view<_StringType, _DelimiterType >::iterator Class Reference

Iterator over the split items.

```
#include <split.h>
```

6.28.1 Detailed Description

```
template<typename _StringType, typename _DelimiterType>
class nvwa::basic_split_view<_StringType, _DelimiterType >::iterator
```

Iterator over the split items.

The iterator *owns* the content.

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

- [split.h](#)

6.29 nvwa::istream_line_reader::iterator Class Reference

Iterator that contains the line content.

```
#include <istream_line_reader.h>
```

6.29.1 Detailed Description

Iterator that contains the line content.

The iterator *owns* the content.

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

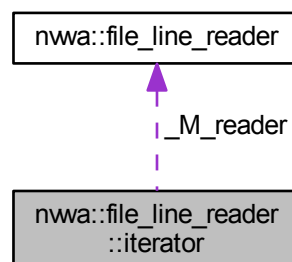
- [istream_line_reader.h](#)

6.30 nvwa::file_line_reader::iterator Class Reference

Iterator that contains the line content.

```
#include <file_line_reader.h>
```

Collaboration diagram for nvwa::file_line_reader::iterator:



Public Member Functions

- [iterator](#) ([file_line_reader](#) *reader)
Constructs the beginning iterator.
- [~iterator](#) ()
Destuctor.
- [iterator](#) (const [iterator](#) &rhs)
Copy-constructor.
- [iterator](#) & [operator=](#) (const [iterator](#) &rhs)
Assignment.
- void [swap](#) ([iterator](#) &rhs) noexcept
Swaps the iterator with another.

6.30.1 Detailed Description

Iterator that contains the line content.

The iterator *owns* the content.

6.30.2 Constructor & Destructor Documentation

6.30.2.1 [iterator](#)() [1/2]

```
nvwa::file_line_reader::iterator::iterator (
    file\_line\_reader * reader ) [explicit]
```

Constructs the beginning iterator.

Parameters

<i>reader</i>	pointer to the file_line_reader object
---------------	--

6.30.2.2 [~iterator](#)()

```
nvwa::file_line_reader::iterator::~~iterator ( )
```

Destuctor.

6.30.2.3 iterator() [2/2]

```
nvwa::file_line_reader::iterator::iterator (
    const iterator & rhs )
```

Copy-constructor.

The line content will be copied to the newly constructed iterator.

Parameters

<i>rhs</i>	the iterator to copy from
------------	---------------------------

6.30.3 Member Function Documentation

6.30.3.1 operator=()

```
file_line_reader::iterator & nvwa::file_line_reader::iterator::operator= (
    const iterator & rhs )
```

Assignment.

The line content will be copied to the newly constructed iterator.

Parameters

<i>rhs</i>	the iterator to copy from
------------	---------------------------

6.30.3.2 swap()

```
void nvwa::file_line_reader::iterator::swap (
    file_line_reader::iterator & rhs ) [noexcept]
```

Swaps the iterator with another.

Parameters

<i>rhs</i>	the iterator to swap with
------------	---------------------------

The documentation for this class was generated from the following files:

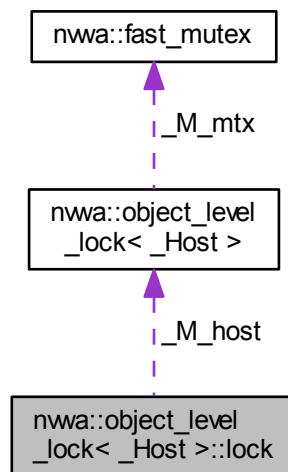
- [file_line_reader.h](#)
- [file_line_reader.cpp](#)

6.31 `nvwa::object_level_lock<_Host>::lock` Class Reference

Type that provides locking/unlocking semantics.

```
#include <object_level_lock.h>
```

Collaboration diagram for `nvwa::object_level_lock<_Host>::lock`:



6.31.1 Detailed Description

```
template<class _Host>
class nvwa::object_level_lock<_Host>::lock
```

Type that provides locking/unlocking semantics.

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

- [object_level_lock.h](#)

6.32 `nvwa::class_level_lock<_Host, _RealLock>::lock` Class Reference

Type that provides locking/unlocking semantics.

```
#include <class_level_lock.h>
```

6.32.1 Detailed Description

```
template<class _Host, bool _RealLock = true>
class nwa::class_level_lock< _Host, _RealLock >::lock
```

Type that provides locking/unlocking semantics.

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

- [class_level_lock.h](#)

6.33 nwa::class_level_lock< _Host, false >::lock Class Reference

Type that provides locking/unlocking semantics.

```
#include <class_level_lock.h>
```

6.33.1 Detailed Description

```
template<class _Host>
class nwa::class_level_lock< _Host, false >::lock
```

Type that provides locking/unlocking semantics.

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

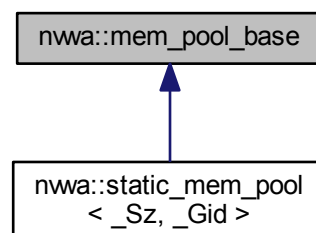
- [class_level_lock.h](#)

6.34 nwa::mem_pool_base Class Reference

Base class for memory pools.

```
#include <mem_pool_base.h>
```

Inheritance diagram for nwa::mem_pool_base:



Classes

- struct [_Block_list](#)

Structure to store the next available memory block.

Public Member Functions

- virtual [~mem_pool_base](#) ()

Empty base destructor.

- virtual void [recycle](#) ()=0

Recycles unused memory from memory pools.

Static Public Member Functions

- static void * [alloc_sys](#) (size_t size)

Allocates memory from the run-time system.

- static void [dealloc_sys](#) (void *ptr)

Frees memory and returns it to the run-time system.

6.34.1 Detailed Description

Base class for memory pools.

6.34.2 Member Function Documentation

6.34.2.1 [alloc_sys\(\)](#)

```
void * nvwa::mem_pool_base::alloc_sys (
    size_t size ) [static]
```

Allocates memory from the run-time system.

Parameters

<i>size</i>	size of the memory to allocate in bytes
-------------	---

Returns

pointer to allocated memory block if successful; or null if memory allocation fails

6.34.2.2 dealloc_sys()

```
void nvwa::mem_pool_base::dealloc_sys (
    void * ptr ) [static]
```

Frees memory and returns it to the run-time system.

Parameters

<i>ptr</i>	pointer to the memory block previously allocated
------------	--

6.34.2.3 recycle()

```
void nvwa::mem_pool_base::recycle ( ) [pure virtual]
```

Recycles unused memory from memory pools.

It is an interface and needs to be implemented in subclasses.

Implemented in [nvwa::static_mem_pool< _Sz, _Gid >](#).

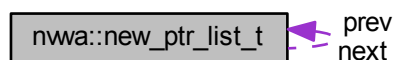
The documentation for this class was generated from the following files:

- [mem_pool_base.h](#)
- [mem_pool_base.cpp](#)

6.35 nvwa::new_ptr_list_t Struct Reference

Structure to store the position information where `new` occurs.

Collaboration diagram for `nvwa::new_ptr_list_t`:



Public Attributes

- `new_ptr_list_t * next`
Pointer to the next memory block.
- `new_ptr_list_t * prev`
Pointer to the previous memory block.
- `size_t size`
Size of the memory block.
- `unsigned line:31`
Line number of the caller; or 0.
- `unsigned is_array:1`
Non-zero iff `new[]` is used.
- `void ** stacktrace`
Pointer to stack trace information.
- `unsigned magic`
Magic number for error detection.
- `char file [_DEBUG_NEW_FILENAME_LEN]`
File name of the caller.
- `void * addr`
Address of the caller to `new`.

6.35.1 Detailed Description

Structure to store the position information where `new` occurs.

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

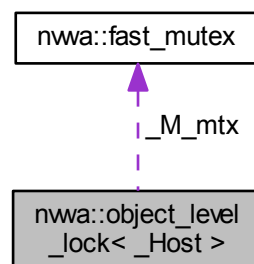
- `debug_new.cpp`

6.36 `nwa::object_level_lock<_Host>` Class Template Reference

Helper class for object-level locking.

```
#include <object_level_lock.h>
```

Collaboration diagram for `nwa::object_level_lock<_Host>`:



Classes

- class [lock](#)

Type that provides locking/unlocking semantics.

6.36.1 Detailed Description

```
template<class _Host>
class nvwa::object_level_lock< _Host >
```

Helper class for object-level locking.

This is the multi-threaded implementation.

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

- [object_level_lock.h](#)

6.37 nvwa::optional<_Tp> Class Template Reference

Class for optional values.

```
#include <functional.h>
```

Inherits nvwa::optional_base<_Tp, bool>.

6.37.1 Detailed Description

```
template<typename _Tp>
class nvwa::optional<_Tp>
```

Class for optional values.

It was initially modelled after the Maybe type in Haskell, but the interface was later changed to be more like (but not fully conformant to) the C++17 std::optional.

Parameters

<code>_Tp</code>	the optional type to store
------------------	----------------------------

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

- [functional.h](#)

6.38 nvwa::output_object<_OutputStrm, _StringType > Struct Template Reference

Functor to output objects pointed by a container of pointers.

```
#include <cont_ptr_utils.h>
```

6.38.1 Detailed Description

```
template<typename _OutputStrm, typename _StringType = const char*>
struct nvwa::output_object<_OutputStrm, _StringType >
```

Functor to output objects pointed by a container of pointers.

A typical usage might be like:

```
list<Object*> l;
...
for_each(l.begin(), l.end(), output_object<ostream>(cout, " "));
```

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

- [cont_ptr_utils.h](#)

6.39 nvwa::smart_ptr<_Tp, _Policy > Struct Template Reference

Declaration of policy class to generate the smart pointer type.

```
#include <tree.h>
```

6.39.1 Detailed Description

```
template<typename _Tp, storage_policy _Policy>
struct nvwa::smart_ptr<_Tp, _Policy >
```

Declaration of policy class to generate the smart pointer type.

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

- [tree.h](#)

6.40 nvwa::smart_ptr<_Tp, storage_policy::shared > Struct Template Reference

Partial specialization to get std::shared_ptr.

```
#include <tree.h>
```

6.40.1 Detailed Description

```
template<typename _Tp>
struct nwa::smart_ptr<_Tp, storage_policy::shared>
```

Partial specialization to get `std::shared_ptr`.

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

- [tree.h](#)

6.41 `nwa::smart_ptr<_Tp, storage_policy::unique>` Struct Template Reference

Partial specialization to get `std::unique_ptr`.

```
#include <tree.h>
```

6.41.1 Detailed Description

```
template<typename _Tp>
struct nwa::smart_ptr<_Tp, storage_policy::unique>
```

Partial specialization to get `std::unique_ptr`.

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

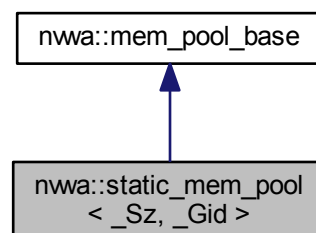
- [tree.h](#)

6.42 `nwa::static_mem_pool<_Sz, _Gid>` Class Template Reference

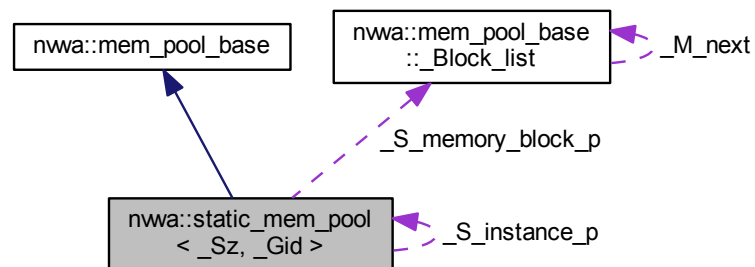
Singleton class template to manage the allocation/deallocation of memory blocks of one specific size.

```
#include <static_mem_pool.h>
```

Inheritance diagram for `nwa::static_mem_pool<_Sz, _Gid>`:



Collaboration diagram for `nwa::static_mem_pool<_Sz, _Gid>`:



Public Member Functions

- `void * allocate ()`
Allocates memory and returns its pointer.
- `void deallocate (void *ptr)`
Deallocates memory by putting the memory block into the pool.
- virtual `void recycle ()` override
Recycles half of the free memory blocks in the memory pool to the system.

Static Public Member Functions

- static `static_mem_pool & instance ()`
Gets the instance of the static memory pool.
- static `static_mem_pool & instance_known ()`
Gets the known instance of the static memory pool.

6.42.1 Detailed Description

```
template<size_t _Sz, int _Gid = -1>
class nwa::static_mem_pool<_Sz, _Gid>
```

Singleton class template to manage the allocation/deallocation of memory blocks of one specific size.

Parameters

<code>_Sz</code>	size of elements in the static_mem_pool
<code>_Gid</code>	group ID of a static_mem_pool : if it is negative, simultaneous accesses to this static_mem_pool will be protected from each other; otherwise no protection is given

6.42.2 Member Function Documentation

6.42.2.1 allocate()

```
template<size_t _Sz, int _Gid = -1>
void* nvwa::static_mem_pool< _Sz, _Gid >::allocate ( ) [inline]
```

Allocates memory and returns its pointer.

The template will try to get it from the memory pool first, and request memory from the system if there is no free memory in the pool.

Returns

pointer to allocated memory if successful; null otherwise

6.42.2.2 deallocate()

```
template<size_t _Sz, int _Gid = -1>
void nvwa::static_mem_pool< _Sz, _Gid >::deallocate (
    void * ptr ) [inline]
```

Deallocates memory by putting the memory block into the pool.

Parameters

<i>ptr</i>	pointer to memory to be deallocated
------------	-------------------------------------

6.42.2.3 instance()

```
template<size_t _Sz, int _Gid = -1>
static static_mem_pool& nvwa::static_mem_pool< _Sz, _Gid >::instance ( ) [inline], [static]
```

Gets the instance of the static memory pool.

It will create the instance if it does not already exist. Generally this function is now not needed.

Returns

reference to the instance of the static memory pool

See also

[instance_known](#)

6.42.2.4 `instance_known()`

```
template<size_t _Sz, int _Gid = -1>
static static\_mem\_pool& nvwa::static\_mem\_pool< _Sz, _Gid >::instance_known ( ) [inline],
[static]
```

Gets the known instance of the static memory pool.

The instance must already exist. Generally the static initializer of the template guarantees it.

Returns

reference to the instance of the static memory pool

6.42.2.5 `recycle()`

```
template<size_t _Sz, int _Gid>
void nvwa::static\_mem\_pool< _Sz, _Gid >::recycle ( ) [override], [virtual]
```

Recycles half of the free memory blocks in the memory pool to the system.

It is called when a memory request to the system (in other instances of the static memory pool) fails.

Implements [nvwa::mem_pool_base](#).

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

- [static_mem_pool.h](#)

6.43 `nvwa::static_mem_pool_set` Class Reference

Singleton class to maintain a set of existing instantiations of [static_mem_pool](#).

```
#include <static_mem_pool.h>
```

Public Member Functions

- void [recycle](#) ()
Asks all static memory pools to recycle unused memory blocks back to the system.
- void [add](#) ([mem_pool_base](#) *memory_pool_p)
Adds a new memory pool to [nvwa::static_mem_pool_set](#).

Static Public Member Functions

- static [static_mem_pool_set](#) & [instance](#) ()
Gets the singleton instance of [nvwa::static_mem_pool_set](#).

6.43.1 Detailed Description

Singleton class to maintain a set of existing instantiations of [static_mem_pool](#).

6.43.2 Member Function Documentation

6.43.2.1 add()

```
void nvwa::static_mem_pool_set::add (
    mem_pool_base * memory_pool_p )
```

Adds a new memory pool to [nvwa::static_mem_pool_set](#).

Parameters

<i>memory_pool_p</i>	pointer to the memory pool to add
----------------------	-----------------------------------

6.43.2.2 instance()

```
static_mem_pool_set & nvwa::static_mem_pool_set::instance ( ) [static]
```

Gets the singleton instance of [nvwa::static_mem_pool_set](#).

The instance will be created on the first invocation.

Returns

reference to the instance of [nvwa::static_mem_pool_set](#)

6.43.2.3 recycle()

```
void nvwa::static_mem_pool_set::recycle ( )
```

Asks all static memory pools to recycle unused memory blocks back to the system.

The caller should get the lock to prevent other operations to [nvwa::static_mem_pool_set](#) during its execution.

The documentation for this class was generated from the following files:

- [static_mem_pool.h](#)
- [static_mem_pool.cpp](#)

6.44 nvwa::tree< _Tp, _Policy > Class Template Reference

Basic tree (node) class template that owns all its children.

```
#include <tree.h>
```

6.44.1 Detailed Description

```
template<typename _Tp, storage_policy _Policy = NVWA_TREE_DEFAULT_STORAGE_POLICY>  
class nvwa::tree< _Tp, _Policy >
```

Basic tree (node) class template that owns all its children.

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

- [tree.h](#)

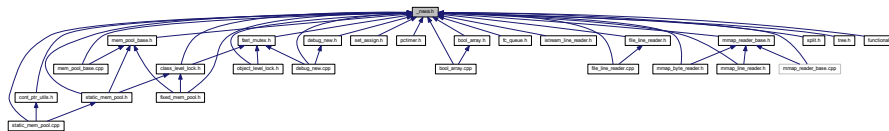
Chapter 7

File Documentation

7.1 _nvwa.h File Reference

Common definitions for preprocessing.

This graph shows which files directly or indirectly include this file:



Namespaces

- [nvwa](#)

Namespace of the nvwa project.

7.1.1 Detailed Description

Common definitions for preprocessing.

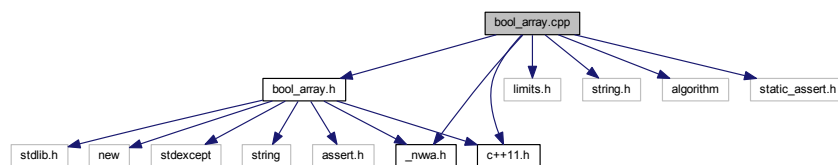
Date _____

2015-10-28

7.2 bool_array.cpp File Reference

Code for class bool_array (packed boolean array).

```
#include "bool_array.h"
#include <limits.h>
#include <string.h>
#include <algorithm>
#include "_nwa.h"
#include "c++11.h"
#include "static_assert.h"
Include dependency graph for bool_array.cpp:
```



Namespaces

- [nvwa](#)

Namespace of the nvwa project.

7.2.1 Detailed Description

Code for class bool_array (packed boolean array).

Date

2017-09-09

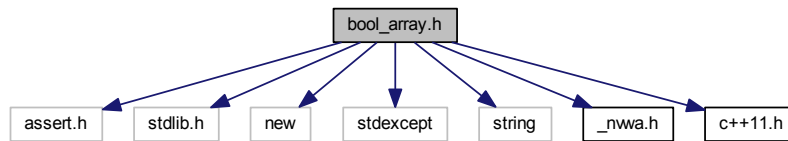
7.3 bool_array.h File Reference

Header file for class bool_array (packed boolean array).

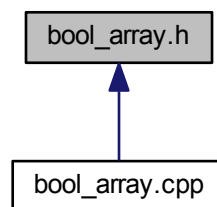
```
#include <assert.h>
#include <stdlib.h>
#include <new>
#include <stdexcept>
#include <string>
#include "_nwa.h"
```

```
#include "c++11.h"
```

Include dependency graph for bool_array.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [nvwa::bool_array](#)
Class to represent a packed boolean array.
- class [nvwa::bool_array::_Element< _Byte_type >](#)
Class to represent a reference to an array element.

Namespaces

- [nvwa](#)
Namespace of the nvwa project.

Functions

- void [nvwa::swap](#) (bool_array &lhs, bool_array &rhs) noexcept
Exchanges the content of two bool_arrays.

7.3.1 Detailed Description

Header file for class bool_array (packed boolean array).

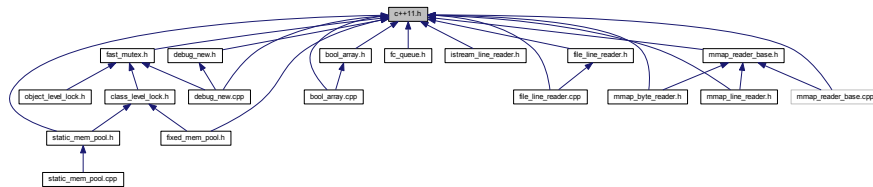
Date

2014-11-29

7.4 c++11.h File Reference

C++11 feature detection macros and workarounds.

This graph shows which files directly or indirectly include this file:



7.4.1 Detailed Description

C++11 feature detection macros and workarounds.

Date

2017-04-03

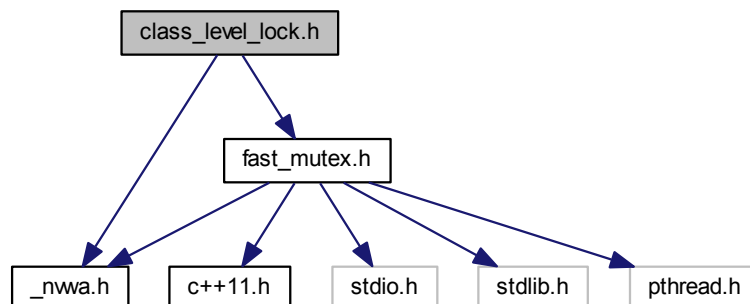
7.5 class_level_lock.h File Reference

In essence Loki ClassLevelLockable re-engineered to use a fast_mutex class.

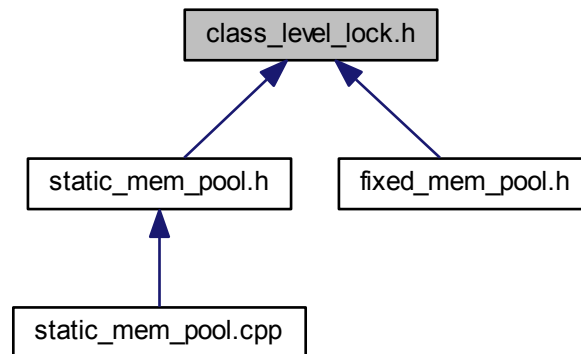
```
#include "fast_mutex.h"
```

```
#include "_nwa.h"
```

Include dependency graph for class_level_lock.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [nvwa::class_level_lock< _Host, _RealLock >](#)
Helper class for class-level locking.
- class [nvwa::class_level_lock< _Host, _RealLock >::lock](#)
Type that provides locking/unlocking semantics.
- class [nvwa::class_level_lock< _Host, false >](#)
Partial specialization that makes null locking.
- class [nvwa::class_level_lock< _Host, false >::lock](#)
Type that provides locking/unlocking semantics.

Namespaces

- [nvwa](#)
Namespace of the nvwa project.

7.5.1 Detailed Description

In essence Loki ClassLevelLockable re-engineered to use a fast_mutex class.

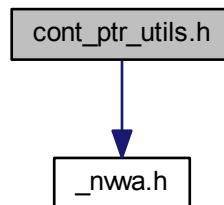
Date

2017-11-25

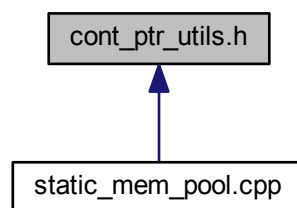
7.6 cont_ptr_utils.h File Reference

Utility functors for containers of pointers (adapted from Scott Meyers' *Effective STL*).

```
#include "_nwa.h"
Include dependency graph for cont_ptr_utils.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct [nvwa::dereference](#)
Functor to return objects pointed by a container of pointers.
- struct [nvwa::dereference_less](#)
Functor to compare objects pointed by a container of pointers.
- struct [nvwa::delete_object](#)
Functor to delete objects pointed by a container of pointers.
- struct [nvwa::output_object](#)< [_OutputStrm](#), [_StringType](#) >
Functor to output objects pointed by a container of pointers.

Namespaces

- [nvwa](#)
Namespace of the nvwa project.

7.6.1 Detailed Description

Utility functors for containers of pointers (adapted from Scott Meyers' *Effective STL*).

Date

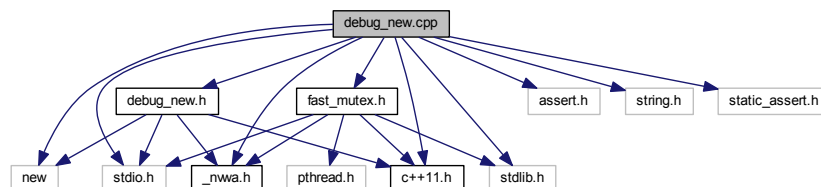
2013-10-06

7.7 debug_new.cpp File Reference

Implementation of debug versions of new and delete to check leakage.

```
#include <new>
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "_nwa.h"
#include "c++11.h"
#include "fast_mutex.h"
#include "static_assert.h"
#include "debug_new.h"
```

Include dependency graph for debug_new.cpp:



Classes

- struct [nwaa::new_ptr_list_t](#)

Structure to store the position information where *new* occurs.

Namespaces

- [nwaa](#)

Namespace of the *nwaa* project.

Macros

- `#define _DEBUG_NEW_REDEFINE_NEW 0`
Macro to indicate whether redefinition of *new* is wanted.
- `#define _DEBUG_NEW_ALIGNMENT 16`
The alignment requirement of allocated memory blocks.
- `#define _DEBUG_NEW_CALLER_ADDRESS __builtin_return_address(0)`
The expression to return the caller address.
- `#define _DEBUG_NEW_ERROR_ACTION abort()`
The action to take when an error occurs.
- `#define _DEBUG_NEW_FILENAME_LEN 44`
The length of file name stored if greater than zero.
- `#define _DEBUG_NEW_PROGNAME _NULLPTR`
The program (executable) name to be set at compile time.
- `#define _DEBUG_NEW_STD_OPER_NEW 1`
Macro to indicate whether the standard-conformant behaviour of operator *new* is wanted.
- `#define _DEBUG_NEW_TAILCHECK 0`
Macro to indicate whether a writing-past-end check will be performed.
- `#define _DEBUG_NEW_TAILCHECK_CHAR 0xCC`
Value of the padding bytes at the end of a memory block.
- `#define _DEBUG_NEW_USE_ADDR2LINE 1`
Whether to use `addr2line` to convert a caller address to file/line information.
- `#define ALIGN(s) (((s) + _DEBUG_NEW_ALIGNMENT - 1) & ~(_DEBUG_NEW_ALIGNMENT - 1))`
Gets the aligned value of memory block size.
- `#define _DEBUG_NEW_REMEMBER_STACK_TRACE 0`
Macro to indicate whether stack traces of allocations should be included in NVWA allocation information and be printed while leaks are reported.

Functions

- static bool `nvwa::print_position_from_addr` (const void *addr)
Tries printing the position information from an instruction address.
- static void `nvwa::print_position` (const void *ptr, int line)
Prints the position information of a memory operation point.
- static void `nvwa::print_stacktrace` (void **stacktrace)
Prints the stack backtrace.
- static bool `nvwa::is_leak_whitelisted` (new_ptr_list_t *ptr)
Checks whether a leak should be ignored.
- static void * `nvwa::alloc_mem` (size_t size, const char *file, int line, bool is_array)
Allocates memory and initializes control data.
- static void `nvwa::free_pointer` (void *usr_ptr, void *addr, bool is_array)
Frees memory and adjusts pointers.
- int `nvwa::check_leaks` ()
Checks for memory leaks.
- int `nvwa::check_mem_corruption` ()
Checks for heap corruption.
- void * `operator new` (size_t size, const char *file, int line)
Allocates memory with file/line information.
- void * `operator new[]` (size_t size, const char *file, int line)
Allocates array memory with file/line information.

- void * [operator new](#) (size_t size) throw (std::bad_alloc)
Allocates memory without file/line information.
- void * [operator new\[\]](#) (size_t size) throw (std::bad_alloc)
Allocates array memory without file/line information.
- void * [operator new](#) (size_t size, const std::nothrow_t &) noexcept
Allocates memory with no-throw guarantee.
- void * [operator new\[\]](#) (size_t size, const std::nothrow_t &) noexcept
Allocates array memory with no-throw guarantee.
- void [operator delete](#) (void *ptr) noexcept
Deallocates memory.
- void [operator delete\[\]](#) (void *ptr) noexcept
Deallocates array memory.
- void [operator delete](#) (void *ptr, const char *file, int line) noexcept
Placement deallocation function.
- void [operator delete\[\]](#) (void *ptr, const char *file, int line) noexcept
Placement deallocation function.
- void [operator delete](#) (void *ptr, const std::nothrow_t &) noexcept
Placement deallocation function.
- void [operator delete\[\]](#) (void *ptr, const std::nothrow_t &) noexcept
Placement deallocation function.

Variables

- const size_t [nvwa::PLATFORM_MEM_ALIGNMENT](#) = sizeof(size_t) * 2
The platform memory alignment.
- static const unsigned [nvwa::DEBUG_NEW_MAGIC](#) = 0x4442474E
Definition of the constant magic number used for error detection.
- static const int [nvwa::ALIGNED_LIST_ITEM_SIZE](#) = ALIGN(sizeof(new_ptr_list_t))
The extra memory allocated by operator new.
- static new_ptr_list_t [nvwa::new_ptr_list](#)
List of all new'd pointers.
- static fast_mutex [nvwa::new_ptr_lock](#)
The mutex guard to protect simultaneous access to the pointer list.
- static fast_mutex [nvwa::new_output_lock](#)
The mutex guard to protect simultaneous output to [new_output_fp](#).
- static size_t [nvwa::total_mem_alloc](#) = 0
Total memory allocated in bytes.

7.7.1 Detailed Description

Implementation of debug versions of new and delete to check leakage.

Date

2017-09-09

7.7.2 Macro Definition Documentation

7.7.2.1 `_DEBUG_NEW_ALIGNMENT`

```
#define _DEBUG_NEW_ALIGNMENT 16
```

The alignment requirement of allocated memory blocks.

It must be a power of two.

7.7.2.2 `_DEBUG_NEW_CALLER_ADDRESS`

```
#define _DEBUG_NEW_CALLER_ADDRESS __builtin_return_address(0)
```

The expression to return the caller address.

[nwva::print_position](#) will later on use this address to print the position information of memory operation points.

7.7.2.3 `_DEBUG_NEW_ERROR_ACTION`

```
#define _DEBUG_NEW_ERROR_ACTION abort()
```

The action to take when an error occurs.

The default behaviour is to call *abort*, unless `_DEBUG_NEW_ERROR_CRASH` is defined, in which case a segmentation fault will be triggered instead (which can be useful on platforms like Windows that do not generate a core dump when *abort* is called).

7.7.2.4 `_DEBUG_NEW_FILENAME_LEN`

```
#define _DEBUG_NEW_FILENAME_LEN 44
```

The length of file name stored if greater than zero.

If it is zero, only a const char pointer will be stored. Currently the default value is non-zero (thus to copy the file name) on non-Windows platforms, because I once found that the exit leakage check could not access the address of the file name on Linux (in my case, a core dump occurred when `check_leaks` tried to access the file name in a shared library after a `SIGINT`). This value makes the size of `new_ptr_list_t` 64 on non-Windows 32-bit platforms (w/o stack backtrace).

7.7.2.5 `_DEBUG_NEW_PROGNAME`

```
#define _DEBUG_NEW_PROGNAME _NULLPTR
```

The program (executable) name to be set at compile time.

It is better to assign the full program path to [nwva::new_progname](#) in *main* (at run time) than to use this (compile-time) macro, but this macro serves well as a quick hack. Note also that double quotation marks need to be used around the program name, i.e., one should specify a command-line option like `-D_DEBUG_NEW_PROGNAME=\"a.out\"` in *bash*, or `-D_DEBUG_NEW_PROGNAME=\"a.exe\"` in the Windows command prompt.

7.7.2.6 _DEBUG_NEW_REDEFINE_NEW

```
#define _DEBUG_NEW_REDEFINE_NEW 0
```

Macro to indicate whether redefinition of `new` is wanted.

If one wants to define one's own `operator new`, or to call `operator new` directly, it should be defined to 0 to alter the default behaviour. Unless, of course, one is willing to take the trouble to write something like:

```
# ifdef new
#   define _NEW_REDEFINED
#   undef new
# endif

// Code that uses new is here

# ifdef _NEW_REDEFINED
#   ifdef DEBUG_NEW
#     define new DEBUG_NEW
#   endif
#   undef _NEW_REDEFINED
# endif
```

Here it is defined to 0 to disable the redefinition of `new`.

7.7.2.7 _DEBUG_NEW_REMEMBER_STACK_TRACE

```
#define _DEBUG_NEW_REMEMBER_STACK_TRACE 0
```

Macro to indicate whether stack traces of allocations should be included in NVWA allocation information and be printed while leaks are reported.

Useful to pinpoint leaks caused by `strdup` and other custom allocation functions. It is also very helpful in filtering out false positives caused by internal STL/C runtime operations. It is off by default because it is quite memory heavy. Set it to 1 to make all allocations have stack trace attached, or set to 2 to make only allocations that lack calling source code information (file and line) have the stack trace attached.

7.7.2.8 _DEBUG_NEW_STD_OPER_NEW

```
#define _DEBUG_NEW_STD_OPER_NEW 1
```

Macro to indicate whether the standard-conformant behaviour of `operator new` is wanted.

It is on by default now, but the user may set it to 0 to revert to the old behaviour.

7.7.2.9 _DEBUG_NEW_TAILCHECK

```
#define _DEBUG_NEW_TAILCHECK 0
```

Macro to indicate whether a writing-past-end check will be performed.

Define it to a positive integer as the number of padding bytes at the end of a memory block for checking.

7.7.2.10 `_DEBUG_NEW_USE_ADDR2LINE`

```
#define _DEBUG_NEW_USE_ADDR2LINE 1
```

Whether to use *addr2line* to convert a caller address to file/line information.

Defining it to a non-zero value will enable the conversion (automatically done if GCC is detected). Defining it to zero will disable the conversion.

7.7.3 Function Documentation

7.7.3.1 `operator delete()` [1/3]

```
void operator delete (
    void * ptr ) [noexcept]
```

Deallocates memory.

Parameters

<i>ptr</i>	pointer to the previously allocated memory
------------	--

7.7.3.2 `operator delete()` [2/3]

```
void operator delete (
    void * ptr,
    const char * file,
    int line ) [noexcept]
```

Placement deallocation function.

For details, please check Section 5.3.4 of the C++ 1998 or 2011 Standard.

Parameters

<i>ptr</i>	pointer to the previously allocated memory
<i>file</i>	null-terminated string of the file name
<i>line</i>	line number

See also

<http://www.csci.csusb.edu/dick/c++std/cd2/expr.html#expr.new>
<http://wyw.dcweb.cn/leakage.htm>

7.7.3.3 operator delete() [3/3]

```
void operator delete (
    void * ptr,
    const std::nothrow_t & ) [nothrow]
```

Placement deallocation function.

For details, please check Section 5.3.4 of the C++ 1998 or 2011 Standard.

Parameters

<i>ptr</i>	pointer to the previously allocated memory
------------	--

7.7.3.4 operator delete[]() [1/3]

```
void operator delete[] (
    void * ptr ) [nothrow]
```

Deallocates array memory.

Parameters

<i>ptr</i>	pointer to the previously allocated memory
------------	--

7.7.3.5 operator delete[]() [2/3]

```
void operator delete[] (
    void * ptr,
    const char * file,
    int line ) [nothrow]
```

Placement deallocation function.

For details, please check Section 5.3.4 of the C++ 1998 or 2011 Standard.

Parameters

<i>ptr</i>	pointer to the previously allocated memory
<i>file</i>	null-terminated string of the file name
<i>line</i>	line number

7.7.3.6 operator delete[]() [3/3]

```
void operator delete[] (
    void * ptr,
    const std::nothrow_t & ) [nothrow]
```

Placement deallocation function.

For details, please check Section 5.3.4 of the C++ 1998 or 2011 Standard.

Parameters

<i>ptr</i>	pointer to the previously allocated memory
------------	--

7.7.3.7 operator new() [1/3]

```
void* operator new (
    size_t size,
    const char * file,
    int line )
```

Allocates memory with file/line information.

Parameters

<i>size</i>	size of the required memory block
<i>file</i>	null-terminated string of the file name
<i>line</i>	line number

Returns

pointer to the memory allocated; or null if memory is insufficient ([_DEBUG_NEW_STD_OPER_NEW](#) is 0)

Exceptions

<i>bad_alloc</i>	memory is insufficient (_DEBUG_NEW_STD_OPER_NEW is 1)
------------------	--

7.7.3.8 operator new() [2/3]

```
void* operator new (
    size_t size ) throw std::bad_alloc)
```

Allocates memory without file/line information.

Parameters

<i>size</i>	size of the required memory block
-------------	-----------------------------------

Returns

pointer to the memory allocated; or null if memory is insufficient (`_DEBUG_NEW_STD_OPER_NEW` is 0)

Exceptions

<i>bad_alloc</i>	memory is insufficient (<code>_DEBUG_NEW_STD_OPER_NEW</code> is 1)
------------------	---

7.7.3.9 operator new() [3/3]

```
void* operator new (  
    size_t size,  
    const std::nothrow_t & ) [nothrow]
```

Allocates memory with no-throw guarantee.

Parameters

<i>size</i>	size of the required memory block
-------------	-----------------------------------

Returns

pointer to the memory allocated; or null if memory is insufficient

7.7.3.10 operator new[]() [1/3]

```
void* operator new[] (  
    size_t size,  
    const char * file,  
    int line )
```

Allocates array memory with file/line information.

Parameters

<i>size</i>	size of the required memory block
<i>file</i>	null-terminated string of the file name
<i>line</i>	line number

Returns

pointer to the memory allocated; or null if memory is insufficient ([_DEBUG_NEW_STD_OPER_NEW](#) is 0)

Exceptions

<i>bad_alloc</i>	memory is insufficient (_DEBUG_NEW_STD_OPER_NEW is 1)
------------------	--

7.7.3.11 operator new[]() [2/3]

```
void* operator new[] (
    size_t size ) throw std::bad_alloc)
```

Allocates array memory without file/line information.

Parameters

<i>size</i>	size of the required memory block
-------------	-----------------------------------

Returns

pointer to the memory allocated; or null if memory is insufficient ([_DEBUG_NEW_STD_OPER_NEW](#) is 0)

Exceptions

<i>bad_alloc</i>	memory is insufficient (_DEBUG_NEW_STD_OPER_NEW is 1)
------------------	--

7.7.3.12 operator new[]() [3/3]

```
void* operator new[] (
    size_t size,
    const std::nothrow_t & ) [nothrow]
```

Allocates array memory with no-throw guarantee.

Parameters

<i>size</i>	size of the required memory block
-------------	-----------------------------------

Returns

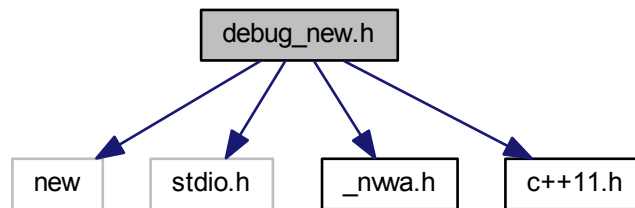
pointer to the memory allocated; or null if memory is insufficient

7.8 debug_new.h File Reference

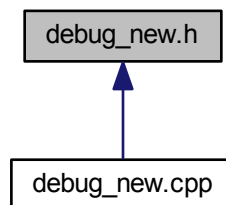
Header file for checking leaks caused by unmatched new/delete.

```
#include <new>
#include <stdio.h>
#include "_nwa.h"
#include "c++11.h"
```

Include dependency graph for debug_new.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [nvwa::debug_new_recorder](#)
Recorder class to remember the call context.
- class [nvwa::debug_new_counter](#)
Counter class for on-exit leakage check.

Namespaces

- [nvwa](#)
Namespace of the nvwa project.

Macros

- `#define _DEBUG_NEW_TYPE 1`
Macro to indicate which variant of `DEBUG_NEW` is wanted.
- `#define DEBUG_NEW NVWA::debug_new_recorder(__FILE__, __LINE__) ->* new`
Macro to catch file/line information on allocation.

Typedefs

- `typedef void(* nvwa::stacktrace_print_callback_t) (FILE *fp, void **stacktrace)`
Callback type for stack trace printing.
- `typedef bool(* nvwa::leak_whitelist_callback_t) (char const *file, int line, void *addr, void **stacktrace)`
Callback type for the leak whitelist function.

Functions

- `void * operator new (size_t size, const char *file, int line)`
Allocates memory with file/line information.
- `void * operator new[] (size_t size, const char *file, int line)`
Allocates array memory with file/line information.
- `void operator delete (void *ptr, const char *file, int line) noexcept`
Placement deallocation function.
- `void operator delete[] (void *ptr, const char *file, int line) noexcept`
Placement deallocation function.
- `int nvwa::check_leaks ()`
Checks for memory leaks.
- `int nvwa::check_mem_corruption ()`
Checks for heap corruption.

Variables

- `bool nvwa::new_autocheck_flag = true`
Flag to control whether `nvwa::check_leaks` will be automatically called on program exit.
- `bool nvwa::new_verbose_flag = false`
Flag to control whether verbose messages are output.
- `FILE * nvwa::new_output_fp = stderr`
Pointer to the output stream.
- `const char * nvwa::new_progname = _DEBUG_NEW_PROGNAME`
Pointer to the program name.
- `stacktrace_print_callback_t nvwa::stacktrace_print_callback = nullptr`
Pointer to the callback used to print the stack backtrace in case of a memory problem.
- `leak_whitelist_callback_t nvwa::leak_whitelist_callback = nullptr`
Pointer to the callback used to filter out false positives from leak reports.
- `static debug_new_counter nvwa::__debug_new_count`
Counting object for each file including `debug_new.h`.

7.8.1 Detailed Description

Header file for checking leaks caused by unmatched new/delete.

Date

2017-09-01

7.8.2 Macro Definition Documentation

7.8.2.1 _DEBUG_NEW_TYPE

```
#define _DEBUG_NEW_TYPE 1
```

Macro to indicate which variant of `DEBUG_NEW` is wanted.

The default value 1 allows the use of placement new (like `new(std::nothrow)`), but the verbose output (when `nvwa::new_verbose_flag` is `true`) looks worse than some older versions (no file/line information for allocations). Define it to 2 to revert to the old behaviour that records file and line information directly on the call to `operator new`.

7.8.2.2 DEBUG_NEW

```
#define DEBUG_NEW NVWA::debug_new_recorder(__FILE__, __LINE__) ->* new
```

Macro to catch file/line information on allocation.

If `_DEBUG_NEW_REDEFINE_NEW` is 0, one can use this macro directly; otherwise `new` will be defined to it, and one must use `new` instead.

7.8.3 Function Documentation

7.8.3.1 operator delete()

```
void operator delete (
    void * ptr,
    const char * file,
    int line ) [noexcept]
```

Placement deallocation function.

For details, please check Section 5.3.4 of the C++ 1998 or 2011 Standard.

Parameters

<i>ptr</i>	pointer to the previously allocated memory
<i>file</i>	null-terminated string of the file name
<i>line</i>	line number

See also

<http://www.csci.csusb.edu/dick/c++std/cd2/expr.html#expr.new>
<http://wyw.dcweb.cn/leakage.htm>

7.8.3.2 operator delete[]()

```
void operator delete[] (
    void * ptr,
    const char * file,
    int line ) [noexcept]
```

Placement deallocation function.

For details, please check Section 5.3.4 of the C++ 1998 or 2011 Standard.

Parameters

<i>ptr</i>	pointer to the previously allocated memory
<i>file</i>	null-terminated string of the file name
<i>line</i>	line number

7.8.3.3 operator new()

```
void* operator new (
    size_t size,
    const char * file,
    int line )
```

Allocates memory with file/line information.

Parameters

<i>size</i>	size of the required memory block
<i>file</i>	null-terminated string of the file name
<i>line</i>	line number

Returns

pointer to the memory allocated; or null if memory is insufficient (`_DEBUG_NEW_STD_OPER_NEW` is 0)

Exceptions

<i>bad_alloc</i>	memory is insufficient (<code>_DEBUG_NEW_STD_OPER_NEW</code> is 1)
------------------	---

7.8.3.4 operator new[]()

```
void* operator new[] (
    size_t size,
    const char * file,
    int line )
```

Allocates array memory with file/line information.

Parameters

<i>size</i>	size of the required memory block
<i>file</i>	null-terminated string of the file name
<i>line</i>	line number

Returns

pointer to the memory allocated; or null if memory is insufficient (`_DEBUG_NEW_STD_OPER_NEW` is 0)

Exceptions

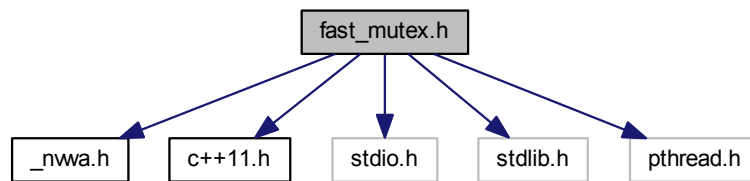
<i>bad_alloc</i>	memory is insufficient (<code>_DEBUG_NEW_STD_OPER_NEW</code> is 1)
------------------	---

7.9 fast_mutex.h File Reference

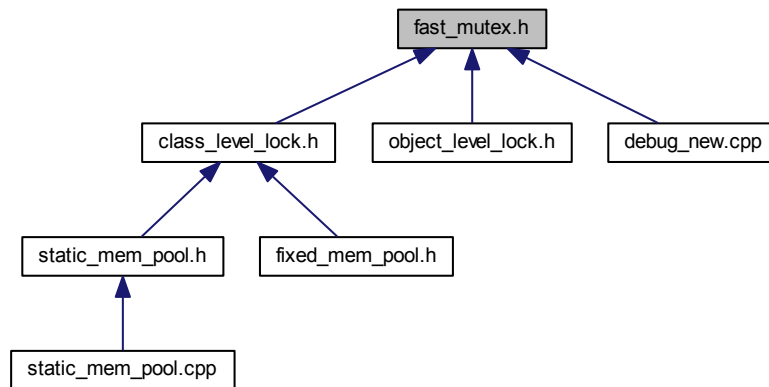
A fast mutex implementation for POSIX, Win32, and modern C++.

```
#include "_nvwa.h"
#include "c++11.h"
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
```

Include dependency graph for `fast_mutex.h`:



This graph shows which files directly or indirectly include this file:



Classes

- class [nvwa::fast_mutex](#)
Class for non-reentrant fast mutexes.
- class [nvwa::fast_mutex_autolock](#)
RAII lock class for [fast_mutex](#).

Namespaces

- [nvwa](#)
Namespace of the nvwa project.

Macros

- [#define _FAST_MUTEX_CHECK_INITIALIZATION 1](#)
Macro to control whether to check for initialization status for each lock/unlock operation.
- [#define _FAST_MUTEX_ASSERT\(_Expr, _Msg\)](#)
Macro for [fast_mutex](#) assertions.
- [#define __VOLATILE volatile](#)
Macro alias to 'volatile' semantics.

7.9.1 Detailed Description

A fast mutex implementation for POSIX, Win32, and modern C++.

Date

2017-09-01

7.9.2 Macro Definition Documentation

7.9.2.1 __VOLATILE

```
#define __VOLATILE volatile
```

Macro alias to 'volatile' semantics.

Here it is truly volatile since it is in a multi-threaded (POSIX threads) environment.

7.9.2.2 _FAST_MUTEX_ASSERT

```
#define _FAST_MUTEX_ASSERT(  
    _Expr,  
    _Msg )
```

Value:

```
if (!(_Expr)) { \n    fprintf(stderr, "fast_mutex::%s\\n", _Msg); \n    abort(); \n}
```

Macro for fast_mutex assertions.

Real version (for debug mode).

7.9.2.3 _FAST_MUTEX_CHECK_INITIALIZATION

```
#define _FAST_MUTEX_CHECK_INITIALIZATION 1
```

Macro to control whether to check for initialization status for each lock/unlock operation.

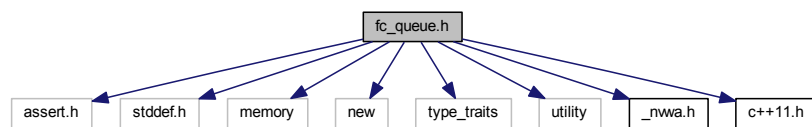
Defining it to a non-zero value will enable the check, so that the construction/destruction of a static object using a static fast_mutex not yet constructed or already destroyed will work (with lock/unlock operations ignored). Defining it to zero will disable the check.

7.10 fc_queue.h File Reference

Definition of a fixed-capacity queue.

```
#include <assert.h>
#include <stddef.h>
#include <memory>
#include <new>
#include <type_traits>
#include <utility>
#include "_nwa.h"
#include "c++11.h"
```

Include dependency graph for fc_queue.h:



Classes

- class [nvwa::fc_queue< _Tp, _Alloc >](#)
Class to represent a fixed-capacity queue.

Namespaces

- [nvwa](#)
Namespace of the nvwa project.

Functions

- `template<class _Tp, class _Alloc >`
void [nvwa::swap](#) (fc_queue< _Tp, _Alloc > &lhs, fc_queue< _Tp, _Alloc > &rhs) noexcept(noexcept(lhs.↔ swap(rhs)))
Exchanges the elements of two queues.

7.10.1 Detailed Description

Definition of a fixed-capacity queue.

Date

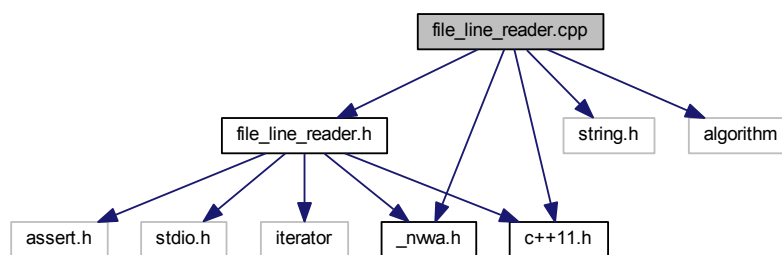
2017-06-10

7.11 file_line_reader.cpp File Reference

Code for file_line_reader, an easy-to-use line-based file reader.

```
#include "file_line_reader.h"
#include <string.h>
#include "_nwa.h"
#include "c++11.h"
#include <algorithm>
```

Include dependency graph for file_line_reader.cpp:



Namespaces

- [nwa](#)

Namespace of the nwa project.

Variables

- `const size_t nwa::BUFFER_SIZE = 256`

Size of buffer.

7.11.1 Detailed Description

Code for file_line_reader, an easy-to-use line-based file reader.

Date

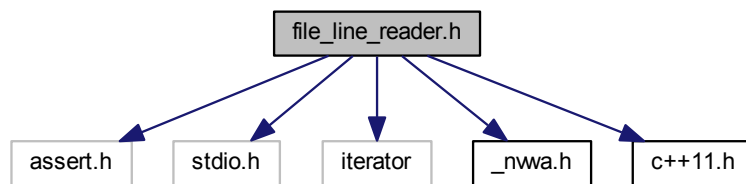
2017-09-09

7.12 file_line_reader.h File Reference

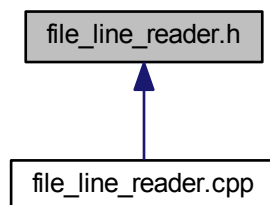
Header file for file_line_reader, an easy-to-use line-based file reader.

```
#include <assert.h>
#include <stdio.h>
#include <iterator>
#include "_nwa.h"
#include "c++11.h"
```

Include dependency graph for file_line_reader.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [nvwa::file_line_reader](#)
Class to allow iteration over all lines of a text file.
- class [nvwa::file_line_reader::iterator](#)
Iterator that contains the line content.

Namespaces

- [nvwa](#)
Namespace of the nvwa project.

7.12.1 Detailed Description

Header file for file_line_reader, an easy-to-use line-based file reader.

Date

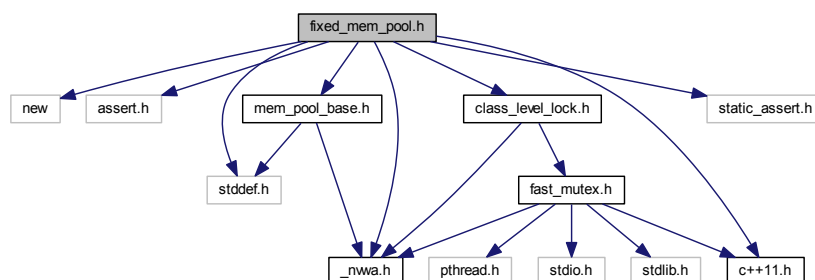
2017-03-23

7.13 fixed_mem_pool.h File Reference

Definition of a fixed-size memory pool template for structs/classes.

```
#include <new>
#include <assert.h>
#include <stddef.h>
#include "_nwa.h"
#include "c++11.h"
#include "class_level_lock.h"
#include "mem_pool_base.h"
#include "static_assert.h"
```

Include dependency graph for fixed_mem_pool.h:



Classes

- class [nvwa::fixed_mem_pool< _Tp >](#)
Class template to manipulate a fixed-size memory pool.
- struct [nvwa::fixed_mem_pool< _Tp >::alignment](#)
Specializable struct to define the alignment of an object in the [fixed_mem_pool](#).
- struct [nvwa::fixed_mem_pool< _Tp >::block_size](#)
Struct to calculate the block size based on the (specializable) alignment value.

Namespaces

- [nvwa](#)
Namespace of the nvwa project.

Macros

- `#define MEM_POOL_ALIGNMENT sizeof(void*)`
Defines the alignment of memory blocks.
- `#define DECLARE_FIXED_MEM_POOL(_Cls)`
Declares the normal (throwing) allocation and deallocation functions.
- `#define DECLARE_FIXED_MEM_POOL__NOSTHROW(_Cls)`
Declares the nothrow allocation and deallocation functions.
- `#define DECLARE_FIXED_MEM_POOL__THROW_NOCHECK(_Cls)`
Declares the throwing, non-checking allocation and deallocation functions.

7.13.1 Detailed Description

Definition of a fixed-size memory pool template for structs/classes.

This is a easy-to-use class template for pre-allocated memory pools. The client side needs to do the following things:

- Use one of the macros
 - `DECLARE_FIXED_MEM_POOL`,
 - `DECLARE_FIXED_MEM_POOL__NOSTHROW`, or
 - `DECLARE_FIXED_MEM_POOL__THROW_NOCHECK`

at the end of the class (say, `class _Cls`) definitions.

- Optionally, specialize `fixed_mem_pool::alignment` to change the alignment value for this specific type.
- Optionally, specialize `fixed_mem_pool::bad_alloc_handler` to change the behaviour when all memory blocks are allocated.
- Call `fixed_mem_pool<_Cls>::initialize` at the beginning of the program.
- Optionally, call `fixed_mem_pool<_Cls>::deinitialize` at exit of the program to check for memory leaks.
- Optionally, call `fixed_mem_pool<_Cls>::get_alloc_count` to check memory usage when the program is running.

Date

2014-11-29

7.13.2 Macro Definition Documentation

7.13.2.1 DECLARE_FIXED_MEM_POOL

```
#define DECLARE_FIXED_MEM_POOL(  
    _Cls )
```

Value:

```
public: \
    static void* operator new(size_t size) \
    { \
        assert(size == sizeof(_Cls)); \
        if (void* ptr = NVWA::fixed_mem_pool<_Cls>::allocate()) \
            return ptr; \
        else \
            throw std::bad_alloc(); \
    } \
    static void operator delete(void* ptr) \
    { \
        if (ptr != _NULLPTR) \
            NVWA::fixed_mem_pool<_Cls>::deallocate(ptr); \
    }
```

Declares the normal (throwing) allocation and deallocation functions.

Parameters

<code>_Cls</code>	class to use the fixed_mem_pool
-------------------	---------------------------------

See also

[DECLARE_FIXED_MEM_POOL__THROW_NOCHECK](#), which, too, defines an **operator new** that will never return null, but requires more discipline on the programmer's side.

7.13.2.2 DECLARE_FIXED_MEM_POOL__NOTHROW

```
#define DECLARE_FIXED_MEM_POOL__NOTHROW(  
    _Cls )
```

Value:

```
public: \
    static void* operator new(size_t size) _NOEXCEPT \
    { \
        assert(size == sizeof(_Cls)); \
        return NVWA::fixed_mem_pool<_Cls>::allocate(); \
    } \
    static void operator delete(void* ptr) \
    { \
        if (ptr != _NULLPTR) \
            NVWA::fixed_mem_pool<_Cls>::deallocate(ptr); \
    }
```

Declares the nothrow allocation and deallocation functions.

Parameters

<code>_Cls</code>	class to use the fixed_mem_pool
-------------------	---------------------------------

7.13.2.3 DECLARE_FIXED_MEM_POOL__THROW_NOCHECK

```
#define DECLARE_FIXED_MEM_POOL__THROW_NOCHECK(  
    _Cls )
```

Value:

```
public: \  
    static void* operator new(size_t size) \  
    { \  
        assert(size == sizeof(_Cls)); \  
        return NVWA::fixed_mem_pool<_Cls>::allocate(); \  
    } \  
    static void operator delete(void* ptr) \  
    { \  
        if (ptr != _NULLPTR) \  
            NVWA::fixed_mem_pool<_Cls>::deallocate(ptr); \  
    }
```

Declares the throwing, non-checking allocation and deallocation functions.

N.B. Using this macro *requires* users to explicitly specialize `fixed_mem_pool::bad_alloc_handler` so that it shall never return `false` (it may throw exceptions, say, `std::bad_alloc`, or simply abort). Otherwise a segmentation fault might occur (instead of returning a null pointer).

Parameters

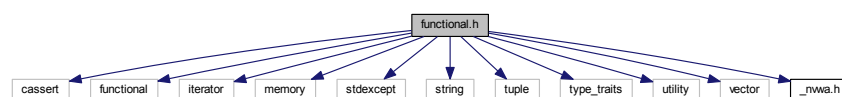
<code>_Cls</code>	class to use the <code>fixed_mem_pool</code>
-------------------	--

7.14 functional.h File Reference

Utility templates for functional programming style.

```
#include <cassert>  
#include <functional>  
#include <iterator>  
#include <memory>  
#include <stdexcept>  
#include <string>  
#include <tuple>  
#include <type_traits>  
#include <utility>  
#include <vector>  
#include "_nvwa.h"
```

Include dependency graph for `functional.h`:



Classes

- class [nvwa::bad_optional_access](#)
Class for bad optional access exception.
- class [nvwa::optional< _Tp >](#)
Class for optional values.

Namespaces

- [nvwa](#)
Namespace of the nvwa project.

Functions

- `template<typename _Fn >`
`auto nvwa::lift_optional (_Fn &&f)`
Lifts a function so that it takes optionals and returns an optional.
- `template<typename _Fn, typename... _Opt>`
`constexpr auto nvwa::apply (_Fn &&f, _Opt &&... args) -> decltype(has_value(args...), optional< std::decay_t< decltype(f(std::forward< _Opt >(args).value()...))>>())`
Applies a function to the values of optionals if they are all valid.
- `template<typename _Fn, class _Tuple >`
`constexpr auto nvwa::apply (_Fn &&f, _Tuple &&t) -> decltype(detail::tuple_apply_impl(std::forward< _Fn >(f), std::forward< _Tuple >(t), std::make_index_sequence< std::tuple_size< std::decay_t< _Tuple >>::value >()))`
Applies the function with all elements of the tuple as arguments.
- `template<typename _Fn, typename _T1, typename _T2 >`
`constexpr auto nvwa::fmap (_Fn &&f, const std::pair< _T1, _T2 > &args)`
Applies a function to both elements of a pair, and makes the results a pair.
- `template<typename _Fn, typename... _Targs>`
`constexpr auto nvwa::fmap (_Fn &&f, const std::tuple< _Targs... > &args)`
Applies a function to all elements of a tuple, and makes the results a tuple.
- `template<template< typename, typename > class _OutCont = std::vector, template< typename > class _Alloc = std::allocator, typename _Fn, class _Rng >`
`constexpr auto nvwa::fmap (_Fn &&f, _Rng &&inputs) -> decltype(detail::adl_begin(inputs), detail::adl_end(inputs), _OutCont< std::decay_t< decltype(f(*detail::adl_begin(inputs)))>>, _Alloc< std::decay_t< decltype(f(*detail::adl_begin(inputs)))>>>())`
Applies a function to each element in the input range.
- `template<typename _Rs, typename _Fn, typename... _Targs>`
`constexpr auto nvwa::reduce (_Fn &&f, const std::tuple< _Targs... > &args, _Rs &&value)`
Applies a function cumulatively to all elements of a tuple.
- `template<typename _Fn, class _Rng >`
`constexpr auto nvwa::reduce (_Fn &&f, _Rng &&inputs)`
Applies a function cumulatively to elements in the input range.
- `template<typename _Rs, typename _Fn, typename _Iter >`
`constexpr _Rs && nvwa::reduce (_Fn &&f, _Rs &&value, _Iter begin, _Iter end)`
Applies a function cumulatively to a range.
- `template<typename _Rs, typename _Fn, class _Rng >`
`constexpr auto nvwa::reduce (_Fn &&f, _Rng &&inputs, _Rs &&initval) -> decltype(f(initval, *detail::adl_begin(inputs)))`
Applies a function cumulatively to elements in the input range.

- `template<typename _T1, typename _T2, typename _Fn >`
`constexpr auto nvwa::wrap_args_as_pair (_Fn &&f)`
Makes a two-argument function accept a pair instead.
- `template<typename _Tuple, typename _Fn >`
`constexpr auto nvwa::wrap_args_as_tuple (_Fn &&f)`
Makes a function accept a tuple as its arguments.
- `template<typename _Tp >`
`constexpr _Tp nvwa::pipeline (_Tp &&data)`
Returns the data intact to terminate the recursion.
- `template<typename _Tp, typename _Fn, typename... _Fargs>`
`decltype(auto) constexpr nvwa::pipeline (_Tp &&data, _Fn &&f, _Fargs &&... args)`
Applies the functions in the arguments to the data consecutively.
- `auto nvwa::compose ()`
Constructs a function (object) that composes the passed functions.
- `template<typename _Fn >`
`auto nvwa::compose (_Fn f)`
Constructs a function (object) that composes the passed functions.
- `template<typename _Fn, typename... _Fargs>`
`auto nvwa::compose (_Fn f, _Fargs... args)`
Constructs a function (object) that composes the passed functions.
- `template<typename _Rs, typename _Tp >`
`std::function< _Rs(_Tp)> nvwa::fix_simple (std::function< _Rs(std::function< _Rs(_Tp)>, _Tp)> f)`
Generates the fixed point using the simple fixed-point combinator.
- `template<typename _Rs, typename _Tp >`
`std::function< _Rs(_Tp)> nvwa::fix_simple (std::function< std::function< _Rs(_Tp)>(std::function< _Rs(↵
_↵
_Tp)>>)> f)`
Generates the fixed point using the simple fixed-point combinator.
- `template<typename _Rs, typename _Tp >`
`std::function< _Rs(_Tp)> nvwa::fix_curry (std::function< std::function< _Rs(_Tp)>(std::function< _Rs(↵
_↵
_Tp)>>)> f)`
Generates the fixed point using the Curry-style fixed-point combinator.
- `template<typename _Rs, typename... _Targs>`
`auto nvwa::make_curry (std::function< _Rs(_Targs...)> f)`
Makes a curried function.
- `template<typename _Rs, typename... _Targs>`
`auto nvwa::make_curry (_Rs(*f)(_Targs...))`
Makes a curried function.
- `template<typename _FnType, typename _Fn >`
`auto nvwa::make_curry (_Fn &&f)`
Makes a curried function.

7.14.1 Detailed Description

Utility templates for functional programming style.

Using this file requires a C++14-compliant compiler.

Date

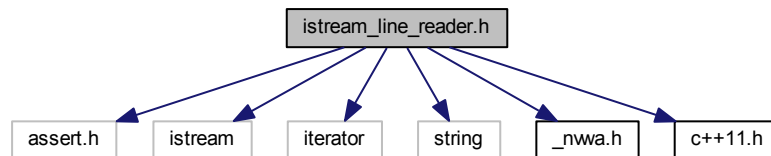
2018-01-02

7.15 istream_line_reader.h File Reference

Header file for istream_line_reader, an easy-to-use line-based istream reader.

```
#include <assert.h>
#include <istream>
#include <iterator>
#include <string>
#include "_nvwa.h"
#include "c++11.h"
```

Include dependency graph for istream_line_reader.h:



Classes

- class [nvwa::istream_line_reader](#)
Class to allow iteration over all lines from an input stream.
- class [nvwa::istream_line_reader::iterator](#)
Iterator that contains the line content.

Namespaces

- [nvwa](#)
Namespace of the nvwa project.

7.15.1 Detailed Description

Header file for istream_line_reader, an easy-to-use line-based istream reader.

This class allows using istreams in a Pythonic way (if your compiler supports C++11 or later), e.g.:

```
for (auto& line : nvwa::istream_line_reader(std::cin)) {
    // Process line
}
```

This code can be used without C++11, but using it would be less convenient then.

It was first published in a [blog](#), and has since been modified to satisfy the `InputIterator` concept, along with other minor changes.

Date

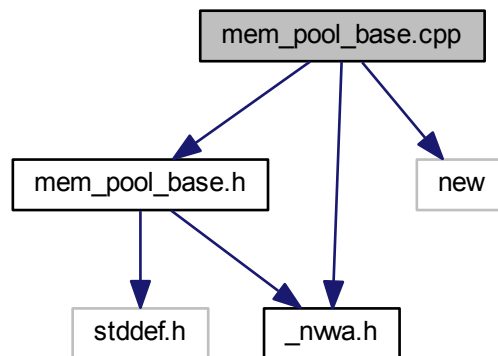
2017-03-23

7.16 mem_pool_base.cpp File Reference

Implementation for the memory pool base.

```
#include "mem_pool_base.h"  
#include <new>  
#include "_nwa.h"
```

Include dependency graph for mem_pool_base.cpp:



Namespaces

- [nvwa](#)

Namespace of the nvwa project.

7.16.1 Detailed Description

Implementation for the memory pool base.

Date

2017-09-09

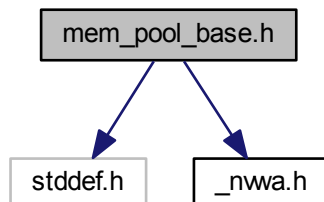
7.17 mem_pool_base.h File Reference

Header file for the memory pool base.

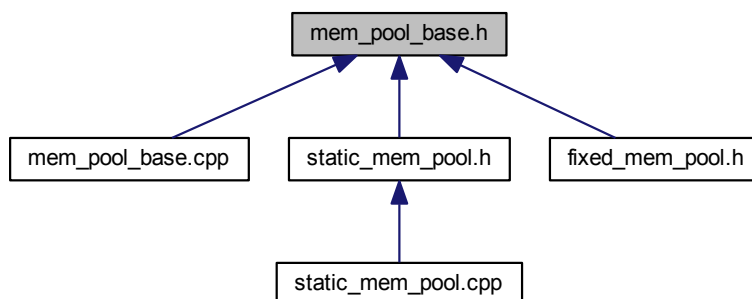
```
#include <stddef.h>
```

```
#include "_nvwa.h"
```

Include dependency graph for mem_pool_base.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [nvwa::mem_pool_base](#)
Base class for memory pools.
- struct [nvwa::mem_pool_base::_Block_list](#)
Structure to store the next available memory block.

Namespaces

- [nvwa](#)
Namespace of the nvwa project.

7.17.1 Detailed Description

Header file for the memory pool base.

Date

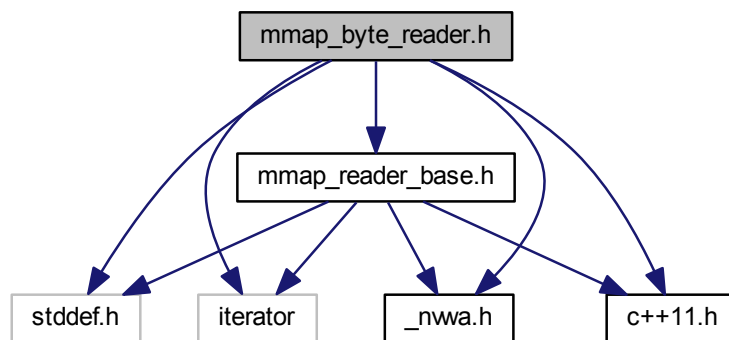
2013-10-06

7.18 mmap_byte_reader.h File Reference

Header file for `mmap_byte_reader`, an easy-to-use byte-based file reader.

```
#include <stddef.h>
#include <iterator>
#include "_nwa.h"
#include "c++11.h"
#include "mmap_reader_base.h"
```

Include dependency graph for `mmap_byte_reader.h`:



Classes

- class `nvwa::basic_mmap_byte_reader< _Tp >`
Class template to allow iteration over all bytes of a mmappable file.
- class `nvwa::basic_mmap_byte_reader< _Tp >::iterator`
Iterator over the bytes.

Namespaces

- `nvwa`
Namespace of the nvwa project.

7.18.1 Detailed Description

Header file for `mmap_byte_reader`, an easy-to-use byte-based file reader.

It is implemented with memory-mapped file APIs.

Date

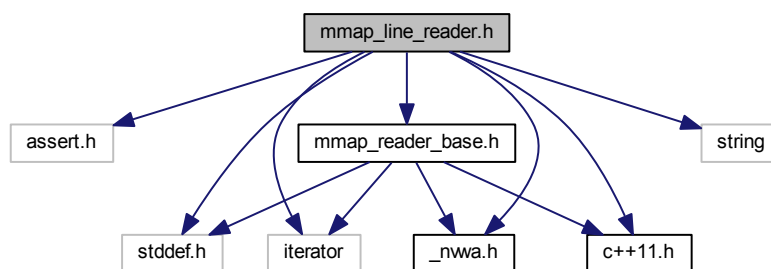
2017-09-12

7.19 mmap_line_reader.h File Reference

Header file for mmap_line_reader and mmap_line_reader_sv, easy-to-use line-based file readers.

```
#include <assert.h>
#include <stddef.h>
#include <iterator>
#include "_nwa.h"
#include "c++11.h"
#include "mmap_reader_base.h"
#include <string>
```

Include dependency graph for mmap_line_reader.h:



Classes

- class [nvwa::basic_mmap_line_reader< _Tp >](#)
Class template to allow iteration over all lines of a mmappable file.
- class [nvwa::basic_mmap_line_reader< _Tp >::iterator](#)
Iterator that contains the line content.

Namespaces

- [nvwa](#)
Namespace of the nvwa project.

7.19.1 Detailed Description

Header file for mmap_line_reader and mmap_line_reader_sv, easy-to-use line-based file readers.

It is implemented with memory-mapped file APIs.

Date

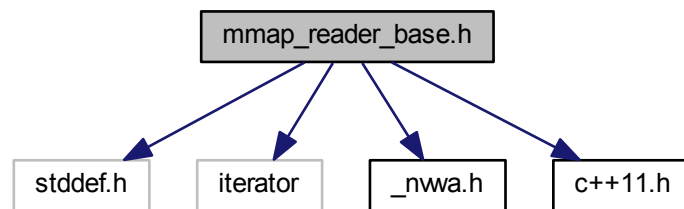
2017-09-10

7.20 mmap_reader_base.h File Reference

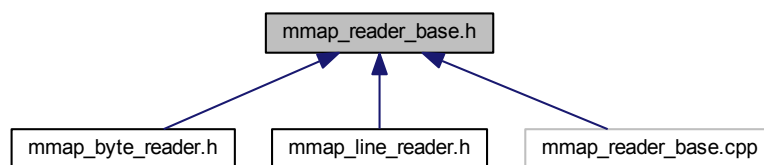
Header file for mmap_reader_base, common base for mmap-based file readers.

```
#include <stddef.h>
#include <iterator>
#include "_nwa.h"
#include "c++11.h"
```

Include dependency graph for mmap_reader_base.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- [nvwa](#)

Namespace of the nvwa project.

7.20.1 Detailed Description

Header file for mmap_reader_base, common base for mmap-based file readers.

It currently supports POSIX and Win32.

Date

2017-09-12

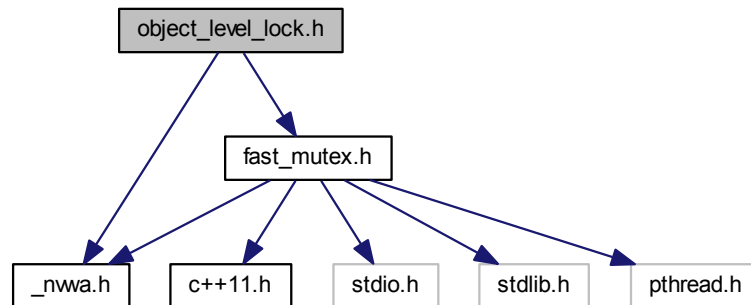
7.21 object_level_lock.h File Reference

In essence Loki ObjectLevelLockable re-engineered to use a fast_mutex class.

```
#include "fast_mutex.h"
```

```
#include "_nvwa.h"
```

Include dependency graph for object_level_lock.h:



Classes

- class `nvwa::object_level_lock<_Host>`
Helper class for object-level locking.
- class `nvwa::object_level_lock<_Host>::lock`
Type that provides locking/unlocking semantics.

Namespaces

- `nvwa`
Namespace of the nvwa project.

7.21.1 Detailed Description

In essence Loki ObjectLevelLockable re-engineered to use a fast_mutex class.

Check also Andrei Alexandrescu's article "[Multithreading and the C++ Type System](#)" for the ideas behind.

Date

2013-03-01

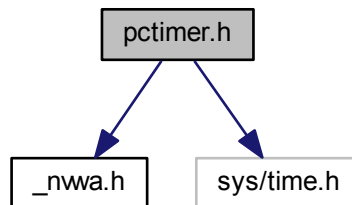
7.22 pctime.h File Reference

Function to get a high-resolution timer for Win32/Cygwin/Unix.

```
#include "_nwa.h"
```

```
#include <sys/time.h>
```

Include dependency graph for pctime.h:



Namespaces

- [nvwa](#)

Namespace of the nvwa project.

7.22.1 Detailed Description

Function to get a high-resolution timer for Win32/Cygwin/Unix.

Date

2015-09-20

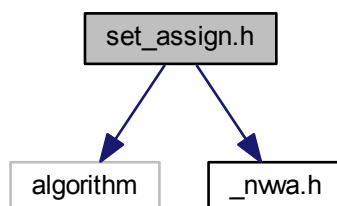
7.23 set_assign.h File Reference

Definition of template functions set_assign_union and set_assign_difference.

```
#include <algorithm>
```

```
#include "_nwa.h"
```

Include dependency graph for set_assign.h:



Namespaces

- [nvwa](#)

Namespace of the nvwa project.

7.23.1 Detailed Description

Definition of template functions `set_assign_union` and `set_assign_difference`.

Date

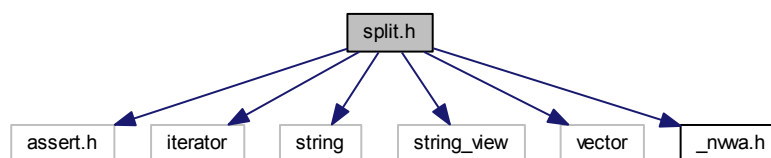
2013-03-01

7.24 split.h File Reference

Header file for an efficient, lazy split function, when using ranges seems an overkill.

```
#include <assert.h>
#include <iterator>
#include <string>
#include <string_view>
#include <vector>
#include "_nvwa.h"
```

Include dependency graph for split.h:



Classes

- class [nvwa::basic_split_view<_StringType, _DelimiterType>](#)
Class to allow iteration over split items from the input.
- class [nvwa::basic_split_view<_StringType, _DelimiterType>::iterator](#)
Iterator over the split items.

Namespaces

- [nvwa](#)

Namespace of the nvwa project.

Functions

- `template<typename _StringType, typename _DelimiterType >`
`constexpr basic_split_view<_StringType, _DelimiterType > nvwa::split(const _StringType &src, _DelimiterType delimiter) noexcept`

Splits a string (or string_view) into lazy views.

7.24.1 Detailed Description

Header file for an efficient, lazy split function, when using ranges seems an overkill.

It allows using split in a Pythonic way, e.g.:

```
for (auto& word : nvwa::split(a_long_string_or_string_view, ' ')) {
    // Process word
}
```

Date

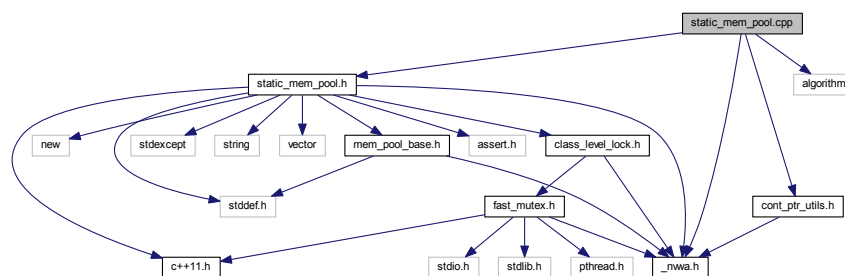
2017-12-31

7.25 static_mem_pool.cpp File Reference

Non-template and non-inline code for the 'static' memory pool.

```
#include "static_mem_pool.h"
#include <algorithm>
#include "_nvwa.h"
#include "cont_ptr_utils.h"
```

Include dependency graph for static_mem_pool.cpp:



Namespaces

- [nvwa](#)

Namespace of the nvwa project.

7.25.1 Detailed Description

Non-template and non-inline code for the 'static' memory pool.

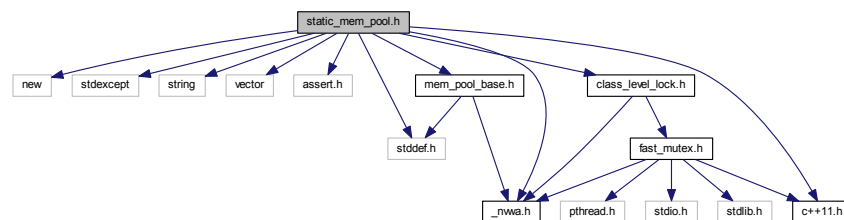
Date

2017-09-09

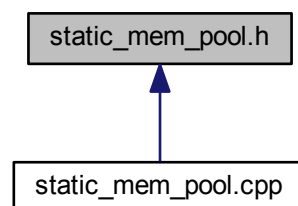
7.26 static_mem_pool.h File Reference

Header file for the 'static' memory pool.

```
#include <new>
#include <stdexcept>
#include <string>
#include <vector>
#include <assert.h>
#include <stddef.h>
#include "_nvw.h"
#include "c++11.h"
#include "class_level_lock.h"
#include "mem_pool_base.h"
Include dependency graph for static_mem_pool.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [nvwa::static_mem_pool_set](#)
Singleton class to maintain a set of existing instantiations of [static_mem_pool](#).
- class [nvwa::static_mem_pool<_Sz, _Gid>](#)
Singleton class template to manage the allocation/deallocation of memory blocks of one specific size.

Namespaces

- [nvwa](#)
Namespace of the nvwa project.

Macros

- `#define DECLARE_STATIC_MEM_POOL\(_Cls\)`
Declares the normal (throwing) allocation and deallocation functions.
- `#define DECLARE_STATIC_MEM_POOL__NOTHROW\(_Cls\)`
Declares the nothrow allocation and deallocation functions.
- `#define DECLARE_STATIC_MEM_POOL_GROUPED\(_Cls, _Gid\)`
Declares the normal (throwing) allocation and deallocation functions.
- `#define DECLARE_STATIC_MEM_POOL_GROUPED__NOTHROW\(_Cls, _Gid\)`
Declares the nothrow allocation and deallocation functions.

7.26.1 Detailed Description

Header file for the 'static' memory pool.

Date

2014-11-29

7.26.2 Macro Definition Documentation

7.26.2.1 DECLARE_STATIC_MEM_POOL

```
#define DECLARE_STATIC_MEM_POOL(  
    _Cls )
```

Value:

```
public: \
static void* operator new(size_t size) \
{ \
    assert(size == sizeof(_Cls)); \
    void* ptr; \
    ptr = NVWA::static_mem_pool<sizeof(_Cls)>:: \
        instance_known().allocate(); \
    if (ptr == _NULLPTR) \
        throw std::bad_alloc(); \
    return ptr; \
} \
static void operator delete(void* ptr) \
{ \
    if (ptr) \
        NVWA::static_mem_pool<sizeof(_Cls)>:: \
            instance_known().deallocate(ptr); \
}
```

Declares the normal (throwing) allocation and deallocation functions.

This macro uses the default group.

Parameters

<code>_Cls</code>	class to use the static_mem_pool
-------------------	----------------------------------

See also

[DECLARE_STATIC_MEM_POOL__NOSTHROW](#)
[DECLARE_STATIC_MEM_POOL_GROUPED](#)
[DECLARE_STATIC_MEM_POOL_GROUPED__NOSTHROW](#)

7.26.2.2 DECLARE_STATIC_MEM_POOL__NOSTHROW

```
#define DECLARE_STATIC_MEM_POOL__NOSTHROW(  
    _Cls )
```

Value:

```
public: \
    static void* operator new(size_t size) _NOSTHROW \
    { \
        assert(size == sizeof(_Cls)); \
        return NVWA::static_mem_pool<sizeof(_Cls)>:: \
            instance_known().allocate(); \
    } \
    static void operator delete(void* ptr) \
    { \
        if (ptr) \
            NVWA::static_mem_pool<sizeof(_Cls)>:: \
                instance_known().deallocate(ptr); \
    }
```

Declares the nothrow allocation and deallocation functions.

This macro uses the default group.

Parameters

<code>_Cls</code>	class to use the static_mem_pool
-------------------	----------------------------------

See also

[DECLARE_STATIC_MEM_POOL](#)
[DECLARE_STATIC_MEM_POOL_GROUPED](#)
[DECLARE_STATIC_MEM_POOL_GROUPED__NOSTHROW](#)

7.26.2.3 DECLARE_STATIC_MEM_POOL_GROUPED

```
#define DECLARE_STATIC_MEM_POOL_GROUPED(  
    _Cls,  
    _Gid )
```

Value:

```

public: \
    static void* operator new(size_t size) \
    { \
        assert(size == sizeof(_Cls)); \
        void* ptr; \
        ptr = NVWA::static_mem_pool<sizeof(_Cls), (_Gid)>:: \
            instance_known().allocate(); \
        if (ptr == _NULLPTR) \
            throw std::bad_alloc(); \
        return ptr; \
    } \
    static void operator delete(void* ptr) \
    { \
        if (ptr) \
            NVWA::static_mem_pool<sizeof(_Cls), (_Gid)>:: \
                instance_known().deallocate(ptr); \
    }

```

Declares the normal (throwing) allocation and deallocation functions.

Users need to specify a group ID.

Parameters

<code>_Cls</code>	class to use the <code>static_mem_pool</code>
<code>_Gid</code>	group ID (negative to protect multi-threaded access)

See also

[DECLARE_STATIC_MEM_POOL](#)
[DECLARE_STATIC_MEM_POOL__NOTHROW](#)
[DECLARE_STATIC_MEM_POOL_GROUPED__NOTHROW](#)

7.26.2.4 DECLARE_STATIC_MEM_POOL_GROUPED__NOTHROW

```

#define DECLARE_STATIC_MEM_POOL_GROUPED__NOTHROW(
    _Cls,
    _Gid )

```

Value:

```

public: \
    static void* operator new(size_t size) _NOEXCEPT \
    { \
        assert(size == sizeof(_Cls)); \
        return NVWA::static_mem_pool<sizeof(_Cls), (_Gid)>:: \
            instance_known().allocate(); \
    } \
    static void operator delete(void* ptr) \
    { \
        if (ptr) \
            NVWA::static_mem_pool<sizeof(_Cls), (_Gid)>:: \
                instance_known().deallocate(ptr); \
    }

```

Declares the nothrow allocation and deallocation functions.

Users need to specify a group ID.

Parameters

<code>_Cls</code>	class to use the <code>static_mem_pool</code>
<code>_Gid</code>	group ID (negative to protect multi-threaded access)

See also

[DECLARE_STATIC_MEM_POOL](#)
[DECLARE_STATIC_MEM_POOL__NOSTHROW](#)
[DECLARE_STATIC_MEM_POOL_GROUPED](#)

7.27 tree.h File Reference

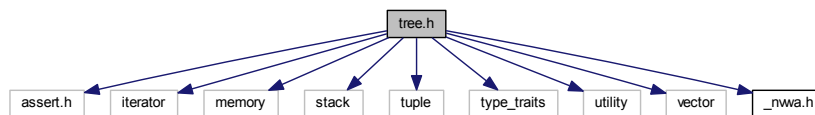
A generic tree class template and the traversal utilities.

```

#include <assert.h>
#include <iterator>
#include <memory>
#include <stack>
#include <tuple>
#include <type_traits>
#include <utility>
#include <vector>
#include "_nvwa.h"

```

Include dependency graph for tree.h:



Classes

- struct [nvwa::smart_ptr<_Tp, _Policy>](#)
Declaration of policy class to generate the smart pointer type.
- struct [nvwa::smart_ptr<_Tp, storage_policy::unique>](#)
Partial specialization to get `std::unique_ptr`.
- struct [nvwa::smart_ptr<_Tp, storage_policy::shared>](#)
Partial specialization to get `std::shared_ptr`.
- class [nvwa::tree<_Tp, _Policy>](#)
Basic tree (node) class template that owns all its children.
- class [nvwa::breadth_first_iteration<_Tree>](#)
Iteration class for breadth-first traversal.
- class [nvwa::depth_first_iteration<_Tree>](#)
Iteration class for depth-first traversal.
- class [nvwa::in_order_iteration<_Tree>](#)
Iteration class for in-order traversal.

Namespaces

- [nvwa](#)

Namespace of the nvwa project.

Enumerations

- enum [nvwa::storage_policy](#) { [nvwa::storage_policy::unique](#), [nvwa::storage_policy::shared](#) }

Policy class for how to store members.

Functions

- `template<storage_policy _Policy = NVWA_TREE_DEFAULT_STORAGE_POLICY, typename _Tp >
tree< typename std::decay< _Tp >::type, _Policy >::tree_ptr nvwa::create_tree (_Tp &&value)`
Creates a tree without any children.
- `template<storage_policy _Policy = NVWA_TREE_DEFAULT_STORAGE_POLICY, typename _Tp , typename... Args>
tree< typename std::decay< _Tp >::type, _Policy >::tree_ptr nvwa::create_tree (_Tp &&value, Args &&...
args)`
Creates a tree with children.

7.27.1 Detailed Description

A generic tree class template and the traversal utilities.

Using this file requires a C++11-compliant compiler.

Date

2017-05-14

Index

- `_DEBUG_NEW_ALIGNMENT`
 - `debug_new.cpp`, [99](#)
- `_DEBUG_NEW_CALLER_ADDRESS`
 - `debug_new.cpp`, [100](#)
- `_DEBUG_NEW_ERROR_ACTION`
 - `debug_new.cpp`, [100](#)
- `_DEBUG_NEW_FILENAME_LEN`
 - `debug_new.cpp`, [100](#)
- `_DEBUG_NEW_PROGNAME`
 - `debug_new.cpp`, [100](#)
- `_DEBUG_NEW_REDEFINE_NEW`
 - `debug_new.cpp`, [100](#)
- `_DEBUG_NEW_REMEMBER_STACK_TRACE`
 - `debug_new.cpp`, [101](#)
- `_DEBUG_NEW_STD_OPER_NEW`
 - `debug_new.cpp`, [101](#)
- `_DEBUG_NEW_TAILCHECK`
 - `debug_new.cpp`, [101](#)
- `_DEBUG_NEW_TYPE`
 - `debug_new.h`, [109](#)
- `_DEBUG_NEW_USE_ADDR2LINE`
 - `debug_new.cpp`, [101](#)
- `_Element`
 - `nvwa::bool_array::Element`, [32](#)
- `_FAST_MUTEX_ASSERT`
 - `fast_mutex.h`, [113](#)
- `_FAST_MUTEX_CHECK_INITIALIZATION`
 - `fast_mutex.h`, [113](#)
- `_M_process`
 - `nvwa::debug_new_recorder`, [54](#)
- `_S_alloc_cnt`
 - `nvwa::fixed_mem_pool`, [72](#)
- `_S_bit_ordinal`
 - `nvwa::bool_array`, [50](#)
- `_S_first_avail_ptr`
 - `nvwa::fixed_mem_pool`, [72](#)
- `_S_mem_pool_ptr`
 - `nvwa::fixed_mem_pool`, [72](#)
- `__VOLATILE`
 - `fast_mutex.h`, [113](#)
- `__debug_new_count`
 - `nvwa`, [29](#)
- `_nvwa.h`, [91](#)
- `~debug_new_counter`
 - `nvwa::debug_new_counter`, [53](#)
- `~fc_queue`
 - `nvwa::fc_queue`, [61](#)
- `~file_line_reader`
 - `nvwa::file_line_reader`, [68](#)
- `~iterator`
 - `nvwa::file_line_reader::iterator`, [76](#)
- `add`
 - `nvwa::static_mem_pool_set`, [89](#)
- `alloc_mem`
 - `nvwa`, [15](#)
- `alloc_sys`
 - `nvwa::mem_pool_base`, [80](#)
- `allocate`
 - `nvwa::fixed_mem_pool`, [70](#)
 - `nvwa::static_mem_pool`, [87](#)
- `apply`
 - `nvwa`, [15](#)
- `at`
 - `nvwa::bool_array`, [42](#)
- `BUFFER_SIZE`
 - `nvwa`, [29](#)
- `back`
 - `nvwa::fc_queue`, [61](#)
- `bad_alloc_handler`
 - `nvwa::fixed_mem_pool`, [70](#)
- `basic_split_view`
 - `nvwa::basic_split_view`, [37](#)
- `bool_array`
 - `nvwa::bool_array`, [41](#), [42](#)
- `bool_array.cpp`, [92](#)
- `bool_array.h`, [92](#)
- `byte`
 - `nvwa::bool_array`, [40](#)
- `c++11.h`, [94](#)
- `capacity`
 - `nvwa::fc_queue`, [61](#)
- `check_leaks`
 - `nvwa`, [16](#)
- `check_mem_corruption`
 - `nvwa`, [16](#)
- `class_level_lock.h`, [94](#)
- `compose`
 - `nvwa`, [16](#), [17](#)
- `cont_ptr_utils.h`, [96](#)
- `contains`
 - `nvwa::fc_queue`, [62](#)
- `copy_to_bitmap`
 - `nvwa::bool_array`, [43](#)
- `count`
 - `nvwa::bool_array`, [43](#)
- `create`

- nvwa::bool_array, 44
- create_tree
 - nvwa, 17, 18
- DEBUG_NEW
 - debug_new.h, 109
- DECLARE_FIXED_MEM_POOL__NOTHROW
 - fixed_mem_pool.h, 119
- DECLARE_FIXED_MEM_POOL__THROW_NOCHECK
 - fixed_mem_pool.h, 120
- DECLARE_FIXED_MEM_POOL
 - fixed_mem_pool.h, 118
- DECLARE_STATIC_MEM_POOL__NOTHROW
 - static_mem_pool.h, 135
- DECLARE_STATIC_MEM_POOL__GROUPED__NOTHROW
 - static_mem_pool.h, 136
- DECLARE_STATIC_MEM_POOL__GROUPED
 - static_mem_pool.h, 135
- DECLARE_STATIC_MEM_POOL
 - static_mem_pool.h, 134
- dealloc_sys
 - nvwa::mem_pool_base, 80
- deallocate
 - nvwa::fixed_mem_pool, 70
 - nvwa::static_mem_pool, 87
- debug_new.cpp, 97
 - _DEBUG_NEW_ALIGNMENT, 99
 - _DEBUG_NEW_CALLER_ADDRESS, 100
 - _DEBUG_NEW_ERROR_ACTION, 100
 - _DEBUG_NEW_FILENAME_LEN, 100
 - _DEBUG_NEW_PROGNAME, 100
 - _DEBUG_NEW_REDEFINE_NEW, 100
 - _DEBUG_NEW_REMEMBER_STACK_TRACE, 101
 - _DEBUG_NEW_STD_OPER_NEW, 101
 - _DEBUG_NEW_TAILCHECK, 101
 - _DEBUG_NEW_USE_ADDR2LINE, 101
 - operator delete, 102
 - operator delete[], 103
 - operator new, 104, 105
 - operator new[], 105, 106
- debug_new.h, 107
 - _DEBUG_NEW_TYPE, 109
 - DEBUG_NEW, 109
 - operator delete, 109
 - operator delete[], 110
 - operator new, 110
 - operator new[], 111
- debug_new_recorder
 - nvwa::debug_new_recorder, 53
- deinitialize
 - nvwa::fixed_mem_pool, 70
- empty
 - nvwa::fc_queue, 62
- fast_mutex.h, 111
 - _FAST_MUTEX_ASSERT, 113
 - _FAST_MUTEX_CHECK_INITIALIZATION, 113
 - __VOLATILE, 113
- fc_queue
 - nvwa::fc_queue, 59, 60
- fc_queue.h, 114
- file_line_reader
 - nvwa::file_line_reader, 67
- file_line_reader.cpp, 115
- file_line_reader.h, 116
- find
 - nvwa::bool_array, 44
- find_until
 - nvwa::bool_array, 45
- fix_curry
 - nvwa, 18
- fix_simple
 - nvwa, 19
- fixed_mem_pool.h, 117
 - DECLARE_FIXED_MEM_POOL__NOTHROW, 119
 - DECLARE_FIXED_MEM_POOL__THROW_NOCHECK, 120
 - DECLARE_FIXED_MEM_POOL, 118
- fmap
 - nvwa, 19, 20
- free_pointer
 - nvwa, 21
- front
 - nvwa::fc_queue, 62, 63
- full
 - nvwa::fc_queue, 63
- functional.h, 120
- get_8bits
 - nvwa::bool_array, 46
- get_alloc_count
 - nvwa::fixed_mem_pool, 71
- get_allocator
 - nvwa::fc_queue, 63
- get_num_bytes_from_bits
 - nvwa::bool_array, 46
- initialize
 - nvwa::bool_array, 46
 - nvwa::fixed_mem_pool, 71
- instance
 - nvwa::static_mem_pool, 87
 - nvwa::static_mem_pool_set, 89
- instance_known
 - nvwa::static_mem_pool, 87
- is_initialized
 - nvwa::fixed_mem_pool, 71
- is_leak_whitelisted
 - nvwa, 21
- istream_line_reader.h, 123
- iterator
 - nvwa::file_line_reader::iterator, 76

- leak_whitelist_callback
 - nvwa, 29
- leak_whitelist_callback_t
 - nvwa, 13
- lift_optional
 - nvwa, 22
- make_curry
 - nvwa, 22, 23
- mem_pool_base.cpp, 124
- mem_pool_base.h, 124
- merge_and
 - nvwa::bool_array, 47
- merge_or
 - nvwa::bool_array, 47
- mmap_byte_reader.h, 126
- mmap_line_reader.h, 127
- mmap_reader_base.h, 128
- new_output_fp
 - nvwa, 29
- new_progname
 - nvwa, 29
- npos
 - nvwa::bool_array, 50
- nvwa, 9
 - __debug_new_count, 29
 - alloc_mem, 15
 - apply, 15
 - BUFFER_SIZE, 29
 - check_leaks, 16
 - check_mem_corruption, 16
 - compose, 16, 17
 - create_tree, 17, 18
 - fix_curry, 18
 - fix_simple, 19
 - fmap, 19, 20
 - free_pointer, 21
 - is_leak_whitelisted, 21
 - leak_whitelist_callback, 29
 - leak_whitelist_callback_t, 13
 - lift_optional, 22
 - make_curry, 22, 23
 - new_output_fp, 29
 - new_progname, 29
 - PLATFORM_MEM_ALIGNMENT, 29
 - pipeline, 23
 - print_position, 24
 - print_position_from_addr, 24
 - print_stacktrace, 24
 - reduce, 25, 26
 - shared, 14
 - split, 27
 - stacktrace_print_callback, 30
 - stacktrace_print_callback_t, 14
 - storage_policy, 14
 - swap, 27
 - unique, 14
 - wrap_args_as_pair, 28
 - wrap_args_as_tuple, 28
 - nvwa::bad_optional_access, 34
 - nvwa::basic_mmap_byte_reader< _Tp >, 34
 - nvwa::basic_mmap_byte_reader< _Tp >::iterator, 73
 - nvwa::basic_mmap_line_reader
 - read, 35
 - strip_type, 35
 - nvwa::basic_mmap_line_reader< _Tp >, 34
 - nvwa::basic_mmap_line_reader< _Tp >::iterator, 74
 - nvwa::basic_split_view
 - basic_split_view, 37
 - to_vector, 37
 - to_vector_sv, 37
 - nvwa::basic_split_view< _StringType, _DelimiterType >, 36
 - nvwa::basic_split_view< _StringType, _DelimiterType >::iterator, 75
 - nvwa::bool_array, 38
 - _S_bit_ordinal, 50
 - at, 42
 - bool_array, 41, 42
 - byte, 40
 - copy_to_bitmap, 43
 - count, 43
 - create, 44
 - find, 44
 - find_until, 45
 - get_8bits, 46
 - get_num_bytes_from_bits, 46
 - initialize, 46
 - merge_and, 47
 - merge_or, 47
 - npos, 50
 - operator=, 48
 - operator[], 48
 - reset, 49
 - set, 49
 - size, 49
 - size_type, 41
 - swap, 50
 - nvwa::bool_array::_Element
 - _Element, 32
 - operator bool, 32
 - operator=, 33
 - nvwa::bool_array::_Element< _Byte_type >, 32
 - nvwa::breadth_first_iteration< _Tree >, 50
 - nvwa::class_level_lock< _Host, _RealLock >, 51
 - nvwa::class_level_lock< _Host, _RealLock >::lock, 78
 - nvwa::class_level_lock< _Host, false >, 52
 - nvwa::class_level_lock< _Host, false >::lock, 79
 - nvwa::debug_new_counter, 52
 - ~debug_new_counter, 53
 - nvwa::debug_new_recorder, 53
 - _M_process, 54
 - debug_new_recorder, 53
 - operator->, 54
 - nvwa::delete_object, 54
 - nvwa::depth_first_iteration< _Tree >, 55

- nvwa::dereference, 55
- nvwa::dereference_less, 56
- nvwa::fast_mutex, 56
- nvwa::fast_mutex_autolock, 57
- nvwa::fc_queue
 - ~fc_queue, 61
 - back, 61
 - capacity, 61
 - contains, 62
 - empty, 62
 - fc_queue, 59, 60
 - front, 62, 63
 - full, 63
 - get_allocator, 63
 - operator=, 64
 - pop, 65
 - push, 65
 - size, 65
 - swap, 66
- nvwa::fc_queue< _Tp, _Alloc >, 57
- nvwa::file_line_reader, 66
 - ~file_line_reader, 68
 - file_line_reader, 67
 - read, 68
 - strip_type, 67
- nvwa::file_line_reader::iterator, 75
 - ~iterator, 76
 - iterator, 76
 - operator=, 77
 - swap, 77
- nvwa::fixed_mem_pool
 - _S_alloc_cnt, 72
 - _S_first_avail_ptr, 72
 - _S_mem_pool_ptr, 72
 - allocate, 70
 - bad_alloc_handler, 70
 - deallocate, 70
 - deinitialize, 70
 - get_alloc_count, 71
 - initialize, 71
 - is_initialized, 71
- nvwa::fixed_mem_pool< _Tp >, 68
- nvwa::fixed_mem_pool< _Tp >::alignment, 33
- nvwa::fixed_mem_pool< _Tp >::block_size, 38
- nvwa::in_order_iteration< _Tree >, 72
- nvwa::istream_line_reader, 73
- nvwa::istream_line_reader::iterator, 75
- nvwa::mem_pool_base, 79
 - alloc_sys, 80
 - dealloc_sys, 80
 - recycle, 81
- nvwa::mem_pool_base::_Block_list, 31
- nvwa::new_ptr_list_t, 81
- nvwa::object_level_lock< _Host >, 82
- nvwa::object_level_lock< _Host >::lock, 78
- nvwa::optional< _Tp >, 83
- nvwa::output_object< _OutputStrm, _StringType >, 84
- nvwa::smart_ptr< _Tp, _Policy >, 84
- nvwa::smart_ptr< _Tp, storage_policy::shared >, 84
- nvwa::smart_ptr< _Tp, storage_policy::unique >, 85
- nvwa::static_mem_pool
 - allocate, 87
 - deallocate, 87
 - instance, 87
 - instance_known, 87
 - recycle, 88
- nvwa::static_mem_pool< _Sz, _Gid >, 85
- nvwa::static_mem_pool_set, 88
 - add, 89
 - instance, 89
 - recycle, 89
- nvwa::tree< _Tp, _Policy >, 90
- object_level_lock.h, 129
- operator bool
 - nvwa::bool_array::_Element, 32
- operator delete
 - debug_new.cpp, 102
 - debug_new.h, 109
- operator delete[]
 - debug_new.cpp, 103
 - debug_new.h, 110
- operator new
 - debug_new.cpp, 104, 105
 - debug_new.h, 110
- operator new[]
 - debug_new.cpp, 105, 106
 - debug_new.h, 111
- operator-> *
 - nvwa::debug_new_recorder, 54
- operator=
 - nvwa::bool_array, 48
 - nvwa::bool_array::_Element, 33
 - nvwa::fc_queue, 64
 - nvwa::file_line_reader::iterator, 77
- operator[]
 - nvwa::bool_array, 48
- PLATFORM_MEM_ALIGNMENT
 - nvwa, 29
- pctimer.h, 130
- pipeline
 - nvwa, 23
- pop
 - nvwa::fc_queue, 65
- print_position
 - nvwa, 24
- print_position_from_addr
 - nvwa, 24
- print_stacktrace
 - nvwa, 24
- push
 - nvwa::fc_queue, 65
- read
 - nvwa::basic_mmap_line_reader, 35
 - nvwa::file_line_reader, 68

- recycle
 - nvwa::mem_pool_base, [81](#)
 - nvwa::static_mem_pool, [88](#)
 - nvwa::static_mem_pool_set, [89](#)
- reduce
 - nvwa, [25](#), [26](#)
- reset
 - nvwa::bool_array, [49](#)
- set
 - nvwa::bool_array, [49](#)
- set_assign.h, [130](#)
- shared
 - nvwa, [14](#)
- size
 - nvwa::bool_array, [49](#)
 - nvwa::fc_queue, [65](#)
- size_type
 - nvwa::bool_array, [41](#)
- split
 - nvwa, [27](#)
- split.h, [131](#)
- stacktrace_print_callback
 - nvwa, [30](#)
- stacktrace_print_callback_t
 - nvwa, [14](#)
- static_mem_pool.cpp, [132](#)
- static_mem_pool.h, [133](#)
 - DECLARE_STATIC_MEM_POOL__NOTHROW, [135](#)
 - DECLARE_STATIC_MEM_POOL_GROUPED_↔_NOTHROW, [136](#)
 - DECLARE_STATIC_MEM_POOL_GROUPED, [135](#)
 - DECLARE_STATIC_MEM_POOL, [134](#)
- storage_policy
 - nvwa, [14](#)
- strip_type
 - nvwa::basic_mmap_line_reader, [35](#)
 - nvwa::file_line_reader, [67](#)
- swap
 - nvwa, [27](#)
 - nvwa::bool_array, [50](#)
 - nvwa::fc_queue, [66](#)
 - nvwa::file_line_reader::iterator, [77](#)
- to_vector
 - nvwa::basic_split_view, [37](#)
- to_vector_sv
 - nvwa::basic_split_view, [37](#)
- tree.h, [137](#)
- unique
 - nvwa, [14](#)
- wrap_args_as_pair
 - nvwa, [28](#)
- wrap_args_as_tuple
 - nvwa, [28](#)