

MemoryAllocationRoutines

5.3

Generated by Doxygen 1.8.14

Contents

1	Module Index	1
1.1	Modules	1
2	Namespace Index	3
2.1	Namespace List	3
3	Hierarchical Index	5
3.1	Class Hierarchy	5
4	Data Structure Index	7
4.1	Data Structures	7
5	File Index	9
5.1	File List	9
6	Module Documentation	11
6.1	Externally-usable macros	11
6.1.1	Detailed Description	12
6.1.2	Macro Definition Documentation	12
6.1.2.1	JEOD_ALLOC_CLASS_ARRAY	12
6.1.2.2	JEOD_ALLOC_CLASS_MULTI_POINTER_ARRAY	13
6.1.2.3	JEOD_ALLOC_CLASS_OBJECT	13
6.1.2.4	JEOD_ALLOC_CLASS_POINTER_ARRAY	14
6.1.2.5	JEOD_ALLOC_PRIM_ARRAY	14
6.1.2.6	JEOD_ALLOC_PRIM_OBJECT	15
6.1.2.7	JEOD_DELETE_2D	16

6.1.2.8	JEOD_DELETE_ARRAY	16
6.1.2.9	JEOD_DELETE_OBJECT	17
6.1.2.10	JEOD_DEREGISTER_CHECKPOINTABLE	17
6.1.2.11	JEOD_IS_ALLOCATED	18
6.1.2.12	JEOD_MEMORY_DEBUG	18
6.1.2.13	JEOD_REGISTER_CHECKPOINTABLE	19
6.1.2.14	JEOD_REGISTER_CLASS	19
6.1.2.15	JEOD_REGISTER_INCOMPLETE_CLASS	20
6.1.2.16	JEOD_REGISTER_NONEXPORTED_CLASS	20
6.2	Internal macros	21
6.2.1	Detailed Description	21
6.2.2	Macro Definition Documentation	21
6.2.2.1	JEOD_ALLOC_ARRAY_INTERNAL	21
6.2.2.2	JEOD_ALLOC_OBJECT_FILL	22
6.2.2.3	JEOD_ALLOC_OBJECT_INTERNAL	22
6.2.2.4	JEOD_ALLOC_POINTER_FILL	22
6.2.2.5	JEOD_ALLOC_PRIMITIVE_FILL	23
6.2.2.6	JEOD_CREATE_MEMORY	23
6.2.2.7	JEOD_DELETE_INTERNAL	23
6.3	Support classes	25
6.3.1	Detailed Description	25
6.3.2	Macro Definition Documentation	25
6.3.2.1	__STDC_LIMIT_MACROS	25
6.4	Models	26
6.4.1	Detailed Description	26
6.5	Utils	27
6.5.1	Detailed Description	27
6.6	Memory	28
6.6.1	Detailed Description	29

7 Namespace Documentation	31
7.1 jeod Namespace Reference	31
7.1.1 Detailed Description	32
7.1.2 Typedef Documentation	32
7.1.2.1 AllocTable	33
7.1.2.2 TypeTable	33
7.1.3 Function Documentation	33
7.1.3.1 jeod_alloc_construct_array()	33
7.1.3.2 jeod_alloc_destruct_array()	34
7.1.3.3 jeod_alloc_get_allocated_pointer()	34
7.1.4 Variable Documentation	35
7.1.4.1 MAGIC0	35
7.1.4.2 MAGIC1	35
8 Data Structure Documentation	37
8.1 jeod::JeodMemoryTypeDescriptor::attr Struct Reference	37
8.1.1 Detailed Description	37
8.2 jeod::JeodAllocHelperAllocatedPointer< T, is_poly > Class Template Reference	37
8.2.1 Detailed Description	37
8.2.2 Member Function Documentation	38
8.2.2.1 cast()	38
8.3 jeod::JeodAllocHelperAllocatedPointer< T, true > Class Template Reference	38
8.3.1 Detailed Description	39
8.3.2 Member Function Documentation	39
8.3.2.1 cast()	39
8.4 jeod::JeodAllocHelperConstructDestruct< T, is_class, is_abstract > Class Template Reference	39
8.4.1 Detailed Description	40
8.4.2 Member Function Documentation	40
8.4.2.1 construct()	40
8.4.2.2 destruct()	41
8.5 jeod::JeodAllocHelperConstructDestruct< T, false, is_abstract > Class Template Reference	41

8.5.1	Detailed Description	41
8.5.2	Member Function Documentation	42
8.5.2.1	construct()	42
8.5.2.2	destruct()	42
8.6	jeod::JeodAllocHelperConstructDestruct< T, true, false > Class Template Reference	43
8.6.1	Detailed Description	43
8.6.2	Member Function Documentation	43
8.6.2.1	construct()	43
8.6.2.2	destruct()	44
8.7	jeod::JeodMemoryItem Class Reference	44
8.7.1	Detailed Description	45
8.7.2	Member Enumeration Documentation	45
8.7.2.1	Flags	45
8.7.3	Constructor & Destructor Documentation	46
8.7.3.1	JeodMemoryItem() [1/2]	46
8.7.3.2	JeodMemoryItem() [2/2]	46
8.7.3.3	~JeodMemoryItem()	47
8.7.4	Member Function Documentation	47
8.7.4.1	construct_flags()	47
8.7.4.2	get_alloc_index()	47
8.7.4.3	get_checkpointed()	48
8.7.4.4	get_descriptor_index()	48
8.7.4.5	get_is_array()	49
8.7.4.6	get_is_guarded()	49
8.7.4.7	get_is_registered()	49
8.7.4.8	get_nelems()	50
8.7.4.9	get_placement_new()	50
8.7.4.10	get_unique_id()	50
8.7.4.11	is_structured_data()	51
8.7.4.12	set_is_registered()	51

8.7.4.13	set_unique_id()	51
8.7.5	Field Documentation	52
8.7.5.1	alloc_info_index	52
8.7.5.2	descriptor_index_hi	52
8.7.5.3	descriptor_index_lo	52
8.7.5.4	flags	53
8.7.5.5	nelems	53
8.7.5.6	unique_id	53
8.8	jeod::JeodMemoryManager Class Reference	54
8.8.1	Detailed Description	57
8.8.2	Member Typedef Documentation	59
8.8.2.1	AllocTable	59
8.8.2.2	TypeTable	59
8.8.3	Member Enumeration Documentation	60
8.8.3.1	DebugLevel	60
8.8.3.2	NameType	60
8.8.4	Constructor & Destructor Documentation	60
8.8.4.1	JeodMemoryManager() [1/3]	60
8.8.4.2	JeodMemoryManager() [2/3]	61
8.8.4.3	JeodMemoryManager() [3/3]	61
8.8.4.4	~JeodMemoryManager()	61
8.8.5	Member Function Documentation	61
8.8.5.1	add_allocation_atomic()	62
8.8.5.2	add_string_atomic()	62
8.8.5.3	allocate_memory()	63
8.8.5.4	begin_atomic_block()	64
8.8.5.5	check_master()	64
8.8.5.6	create_memory()	65
8.8.5.7	create_memory_internal()	66
8.8.5.8	delete_oldest_alloc_entry_atomic()	66

8.8.5.9	deregister_container()	67
8.8.5.10	destroy_memory()	68
8.8.5.11	destroy_memory_internal()	68
8.8.5.12	end_atomic_block()	69
8.8.5.13	find_alloc_entry_atomic()	69
8.8.5.14	free_memory()	70
8.8.5.15	generate_shutdown_report()	71
8.8.5.16	get_alloc_id_atomic()	71
8.8.5.17	get_string_atomic()	72
8.8.5.18	get_type_descriptor() [1/2]	72
8.8.5.19	get_type_descriptor() [2/2]	73
8.8.5.20	get_type_descriptor_atomic() [1/2]	74
8.8.5.21	get_type_descriptor_atomic() [2/2]	74
8.8.5.22	get_type_descriptor_nolock()	75
8.8.5.23	get_type_entry_atomic() [1/2]	75
8.8.5.24	get_type_entry_atomic() [2/2]	76
8.8.5.25	get_type_index_nolock()	76
8.8.5.26	is_allocated()	77
8.8.5.27	is_allocated_internal()	78
8.8.5.28	is_table_empty()	78
8.8.5.29	operator=()	78
8.8.5.30	register_class()	79
8.8.5.31	register_container()	79
8.8.5.32	register_memory_internal()	80
8.8.5.33	reset_alloc_id_atomic()	81
8.8.5.34	restart_clear_memory()	81
8.8.5.35	restart_reallocate()	81
8.8.5.36	set_debug_level() [1/2]	82
8.8.5.37	set_debug_level() [2/2]	82
8.8.5.38	set_guard_enabled()	83

8.8.5.39	set_mode()	83
8.8.5.40	set_mode_internal()	83
8.8.6	Friends And Related Function Documentation	84
8.8.6.1	init_attrjeod__JeodMemoryManager	84
8.8.6.2	InputProcessor	84
8.8.7	Field Documentation	84
8.8.7.1	alloc_table	84
8.8.7.2	allocation_number	85
8.8.7.3	cur_data_size	85
8.8.7.4	debug_level	85
8.8.7.5	guard_enabled	86
8.8.7.6	Master	86
8.8.7.7	max_data_size	86
8.8.7.8	max_table_size	86
8.8.7.9	mode	87
8.8.7.10	mutex	87
8.8.7.11	sim_interface	87
8.8.7.12	string_table	87
8.8.7.13	type_table	88
8.9	jeod::JeodMemoryReflectiveTable Class Reference	88
8.9.1	Detailed Description	89
8.9.2	Constructor & Destructor Documentation	89
8.9.2.1	JeodMemoryReflectiveTable() [1/2]	89
8.9.2.2	JeodMemoryReflectiveTable() [2/2]	89
8.9.3	Member Function Documentation	89
8.9.3.1	add() [1/2]	89
8.9.3.2	add() [2/2]	89
8.9.3.3	operator=()	90
8.10	jeod::JeodMemoryTable< ValueType > Class Template Reference	90
8.10.1	Detailed Description	91

8.10.2	Member Typedef Documentation	92
8.10.2.1	const_value_iterator	92
8.10.2.2	NameIndex	93
8.10.2.3	ValueList	93
8.10.3	Constructor & Destructor Documentation	93
8.10.3.1	JeodMemoryTable() [1/2]	93
8.10.3.2	~JeodMemoryTable()	93
8.10.3.3	JeodMemoryTable() [2/2]	94
8.10.4	Member Function Documentation	94
8.10.4.1	add()	94
8.10.4.2	begin()	94
8.10.4.3	clone()	95
8.10.4.4	del()	95
8.10.4.5	end()	95
8.10.4.6	find()	96
8.10.4.7	get()	96
8.10.4.8	operator=()	97
8.10.5	Field Documentation	97
8.10.5.1	string_to_index	97
8.10.5.2	value_list	97
8.11	jeod::JeodMemoryTableClonable< ValueType > Class Template Reference	98
8.11.1	Detailed Description	98
8.11.2	Constructor & Destructor Documentation	98
8.11.2.1	JeodMemoryTableClonable() [1/2]	99
8.11.2.2	JeodMemoryTableClonable() [2/2]	99
8.11.3	Member Function Documentation	99
8.11.3.1	clone()	99
8.11.3.2	operator=()	99
8.12	jeod::JeodMemoryTableCopyable< ValueType > Class Template Reference	100
8.12.1	Detailed Description	100

8.12.2	Constructor & Destructor Documentation	100
8.12.2.1	JeodMemoryTableCopyable() [1/2]	101
8.12.2.2	JeodMemoryTableCopyable() [2/2]	101
8.12.3	Member Function Documentation	101
8.12.3.1	clone()	101
8.12.3.2	operator=()	101
8.13	jeod::JeodMemoryTypeDescriptor Class Reference	102
8.13.1	Detailed Description	104
8.13.2	Constructor & Destructor Documentation	104
8.13.2.1	JeodMemoryTypeDescriptor() [1/2]	104
8.13.2.2	~JeodMemoryTypeDescriptor()	104
8.13.2.3	JeodMemoryTypeDescriptor() [2/2]	105
8.13.3	Member Function Documentation	105
8.13.3.1	base_type()	105
8.13.3.2	buffer_end() [1/2]	105
8.13.3.3	buffer_end() [2/2]	106
8.13.3.4	buffer_size() [1/2]	106
8.13.3.5	buffer_size() [2/2]	106
8.13.3.6	clone()	107
8.13.3.7	construct_array()	107
8.13.3.8	delete_array()	107
8.13.3.9	delete_object()	108
8.13.3.10	destroy_memory()	108
8.13.3.11	destruct_array()	109
8.13.3.12	dimensionality()	109
8.13.3.13	get_attr()	109
8.13.3.14	get_name()	110
8.13.3.15	get_register_instances()	110
8.13.3.16	get_size()	110
8.13.3.17	get_typeid()	111

8.13.3.18	initialize_type_name()	111
8.13.3.19	is_structured()	111
8.13.3.20	most_derived_pointer() [1/2]	112
8.13.3.21	most_derived_pointer() [2/2]	112
8.13.3.22	operator=()	112
8.13.3.23	pointer_dimension()	112
8.13.3.24	set_check_for_registration_errors()	112
8.13.3.25	type_spec()	113
8.13.4	Field Documentation	113
8.13.4.1	check_for_registration_errors	113
8.13.4.2	name	114
8.13.4.3	obj_id	114
8.13.4.4	register_instances	114
8.13.4.5	size	114
8.14	jeod::JeodMemoryTypeDescriptorDerived< Type > Class Template Reference	115
8.14.1	Detailed Description	116
8.14.2	Member Typedef Documentation	116
8.14.2.1	Attributes	116
8.14.2.2	TypeDescriptor	116
8.14.3	Constructor & Destructor Documentation	116
8.14.3.1	JeodMemoryTypeDescriptorDerived() [1/2]	117
8.14.3.2	JeodMemoryTypeDescriptorDerived() [2/2]	117
8.14.3.3	~JeodMemoryTypeDescriptorDerived()	117
8.14.4	Member Function Documentation	117
8.14.4.1	clone()	117
8.14.4.2	construct_array()	118
8.14.4.3	delete_array()	118
8.14.4.4	delete_object()	118
8.14.4.5	destruct_array()	119
8.14.4.6	is_structured()	119

8.14.4.7	most_derived_pointer() [1/2]	119
8.14.4.8	most_derived_pointer() [2/2]	120
8.14.4.9	operator=()	120
8.15	jeod::JeodMemoryTypePreDescriptor Class Reference	121
8.15.1	Detailed Description	121
8.15.2	Constructor & Destructor Documentation	121
8.15.2.1	~JeodMemoryTypePreDescriptor()	121
8.15.3	Member Function Documentation	121
8.15.3.1	get_descriptor()	122
8.15.3.2	get_typeid()	122
8.16	jeod::JeodMemoryTypePreDescriptorDerived< Type > Class Template Reference	122
8.16.1	Detailed Description	123
8.16.2	Member Typedef Documentation	123
8.16.2.1	TypeDescriptor	123
8.16.3	Constructor & Destructor Documentation	124
8.16.3.1	JeodMemoryTypePreDescriptorDerived() [1/2]	124
8.16.3.2	JeodMemoryTypePreDescriptorDerived() [2/2]	124
8.16.3.3	~JeodMemoryTypePreDescriptorDerived()	124
8.16.4	Member Function Documentation	124
8.16.4.1	get_descriptor()	125
8.16.4.2	get_ref()	125
8.16.4.3	get_typeid()	126
8.16.5	Field Documentation	126
8.16.5.1	descriptor	126
8.16.5.2	is_exportable	126
8.17	jeod::JeodSimEngineAttributes< Type, is_class > Class Template Reference	126
8.17.1	Detailed Description	127
8.17.2	Member Function Documentation	127
8.17.2.1	attributes()	127
8.18	jeod::JeodSimEngineAttributes< Type *, false > Class Template Reference	127

8.18.1 Detailed Description	128
8.18.2 Member Function Documentation	128
8.18.2.1 attributes()	128
8.19 jeod::JeodSimEngineAttributes< Type, true > Class Template Reference	128
8.19.1 Detailed Description	129
8.19.2 Member Function Documentation	129
8.19.2.1 attributes()	129
8.20 jeod::JeodSimEngineAttributes< void *, false > Class Template Reference	129
8.20.1 Detailed Description	130
8.20.2 Member Function Documentation	130
8.20.2.1 attributes()	130
8.21 jeod::MemoryMessages Class Reference	130
8.21.1 Detailed Description	131
8.21.2 Constructor & Destructor Documentation	131
8.21.2.1 MemoryMessages() [1/2]	131
8.21.2.2 MemoryMessages() [2/2]	132
8.21.3 Member Function Documentation	132
8.21.3.1 operator=()	132
8.21.4 Friends And Related Function Documentation	132
8.21.4.1 init_attrjeod__MemoryMessages	132
8.21.4.2 InputProcessor	132
8.21.5 Field Documentation	132
8.21.5.1 corrupted_memory	132
8.21.5.2 debug	133
8.21.5.3 internal_error	133
8.21.5.4 invalid_size	133
8.21.5.5 lock_error	133
8.21.5.6 null_pointer	134
8.21.5.7 out_of_memory	134
8.21.5.8 registration_error	134
8.21.5.9 singleton_error	134
8.21.5.10 suspect_pointer	135
8.22 jeod::JeodMemoryManager::TypeEntry Struct Reference	135
8.22.1 Detailed Description	135
8.22.2 Constructor & Destructor Documentation	135
8.22.2.1 TypeEntry()	136
8.22.3 Field Documentation	136
8.22.3.1 index	136
8.22.3.2 tdesc	136

9 File Documentation	137
9.1 class_declarations.hh File Reference	137
9.1.1 Detailed Description	137
9.2 jeod_alloc.hh File Reference	137
9.2.1 Detailed Description	139
9.3 jeod_alloc_construct_destruct.hh File Reference	139
9.3.1 Detailed Description	140
9.4 jeod_alloc_get_allocated_pointer.hh File Reference	140
9.4.1 Detailed Description	141
9.5 memory_attributes_templates.hh File Reference	141
9.5.1 Detailed Description	142
9.6 memory_item.cc File Reference	142
9.6.1 Detailed Description	142
9.7 memory_item.hh File Reference	142
9.7.1 Detailed Description	143
9.8 memory_manager.cc File Reference	143
9.8.1 Detailed Description	143
9.8.2 Macro Definition Documentation	143
9.8.2.1 MAKE_DESCRIPTOR	144
9.9 memory_manager.hh File Reference	144
9.9.1 Detailed Description	145
9.10 memory_manager_hide_from_trick.hh File Reference	145
9.10.1 Detailed Description	145
9.11 memory_manager_protected.cc File Reference	145
9.11.1 Detailed Description	146
9.12 memory_manager_static.cc File Reference	146
9.12.1 Detailed Description	146
9.13 memory_messages.cc File Reference	146
9.13.1 Detailed Description	147
9.13.2 Macro Definition Documentation	147
9.13.2.1 MAKE_MEMORY_MESSAGE_CODE	147
9.14 memory_messages.hh File Reference	147
9.14.1 Detailed Description	147
9.15 memory_table.hh File Reference	148
9.15.1 Detailed Description	148
9.16 memory_type.cc File Reference	148
9.16.1 Detailed Description	149
9.17 memory_type.hh File Reference	149
9.17.1 Detailed Description	149
Index	151

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Models	26
Utils	27
Memory	28
Externally-usable macros	11
Internal macros	21
Support classes	25

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

jeod	Namespace jeod	31
----------------------	--------------------------	--------------------

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

jeod::JeodMemoryTypeDescriptor::attr	37
jeod::JeodAllocHelperAllocatedPointer< T, is_poly >	37
jeod::JeodAllocHelperAllocatedPointer< T, true >	38
jeod::JeodAllocHelperConstructDestruct< T, is_class, is_abstract >	39
jeod::JeodAllocHelperConstructDestruct< T, false, is_abstract >	41
jeod::JeodAllocHelperConstructDestruct< T, true, false >	43
jeod::JeodMemoryItem	44
jeod::JeodMemoryManager	54
jeod::JeodMemoryTable< ValueType >	90
jeod::JeodMemoryTableClonable< ValueType >	98
jeod::JeodMemoryTableCopyable< ValueType >	100
jeod::JeodMemoryTable< JeodMemoryTypeDescriptor >	90
jeod::JeodMemoryTableClonable< JeodMemoryTypeDescriptor >	98
jeod::JeodMemoryTable< std::string >	90
jeod::JeodMemoryTableCopyable< std::string >	100
jeod::JeodMemoryReflectiveTable	88
jeod::JeodMemoryTypeDescriptor	102
jeod::JeodMemoryTypeDescriptorDerived< Type >	115
jeod::JeodMemoryTypePreDescriptor	121
jeod::JeodMemoryTypePreDescriptorDerived< Type >	122
jeod::JeodSimEngineAttributes< Type, is_class >	126
jeod::JeodSimEngineAttributes< Type *, false >	127
jeod::JeodSimEngineAttributes< Type, true >	128
jeod::JeodSimEngineAttributes< void *, false >	129
jeod::MemoryMessages	130
jeod::JeodMemoryManager::TypeEntry	135

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

jeod::JeodMemoryTypeDescriptor::attr	37
The simulation engine attributes that describe the type. <code>trick_io(**)</code>	
jeod::JeodAllocHelperAllocatedPointer< T, is_poly >	37
Class template that provides a static function <i>cast</i> that casts a pointer to an object of type <i>T</i> to a <code>void*</code> pointer	
jeod::JeodAllocHelperAllocatedPointer< T, true >	38
Partial instantiation of JeodAllocHelperAllocatedPointer for polymorphic classes	
jeod::JeodAllocHelperConstructDestruct< T, is_class, is_abstract >	39
Class template that provides static functions <i>construct</i> and <i>destruct</i> that construct an array of objects	
jeod::JeodAllocHelperConstructDestruct< T, false, is_abstract >	41
Partial instantiation for non-classes	
jeod::JeodAllocHelperConstructDestruct< T, true, false >	43
Partial instantiation for non-abstract classes	
jeod::JeodMemoryItem	44
A JeodMemoryItem contains metadata about some chunk of allocated memory	
jeod::JeodMemoryManager	54
This class provides the interface between the macros in jeod_alloc.hh and the rest of the JEOD memory model	
jeod::JeodMemoryReflectiveTable	88
A JeodMemoryReflectiveTable maps strings to themselves	
jeod::JeodMemoryTable< ValueType >	90
A JeodMemoryTable maps strings to values with a coordinated map/vector pair	
jeod::JeodMemoryTableClonable< ValueType >	98
A JeodMemoryTableClonable is a JeodMemoryTable that implements the required <code>clone()</code> functionality by invoking the <i>ValueType</i> 's <code>clone()</code> method to create a clone of the input value	
jeod::JeodMemoryTableCopyable< ValueType >	100
A JeodMemoryTableCopyable is a JeodMemoryTable that implements the required <code>clone()</code> functionality by invoking the <i>ValueType</i> 's copy constructor to create a clone of the input value	
jeod::JeodMemoryTypeDescriptor	102
Abstract class for managing data allocated as some specific type	
jeod::JeodMemoryTypeDescriptorDerived< Type >	115
Extends JeodMemoryTypeDescriptor to describe a specific type	
jeod::JeodMemoryTypePreDescriptor	121
Abstract class for describing a type without necessarily needing to create a JeodMemoryTypeDescriptor of that type	

jeod::JeodMemoryTypePreDescriptorDerived< Type >	
A JeodMemoryTypePreDescriptorDerived describes a <i>Type</i>	122
jeod::JeodSimEngineAttributes< Type, is_class >	
Class template to construct a simulation engine attributes object that represents some type	126
jeod::JeodSimEngineAttributes< Type *, false >	
Partial template instantiation of JeodSimEngineAttributes for a pointer type	127
jeod::JeodSimEngineAttributes< Type, true >	
Partial template instantiation of JeodSimEngineAttributes for a class	128
jeod::JeodSimEngineAttributes< void *, false >	
Template specialization of JeodSimEngineAttributes for void*	129
jeod::MemoryMessages	
Declares messages associated with the integration test model	130
jeod::JeodMemoryManager::TypeEntry	
The type table is indexed by an integer and contains type descriptors	135

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

class_declarations.hh	Forward declarations of classes defined in the utils/memory model	137
jeod_alloc.hh	Define JEOD memory allocation macros	137
jeod_alloc_construct_destruct.hh	Define templates for use by jeod_alloc.hh	139
jeod_alloc_get_allocated_pointer.hh	Define function template <code>jeod_alloc_get_allocated_pointer</code>	140
memory_attributes_templates.hh	Define the class template <code>JeodSimEngineAttributes</code>	141
memory_item.cc	Implement the <code>JeodMemoryItem</code> class	142
memory_item.hh	Define the class <code>JeodMemoryItem</code>	142
memory_manager.cc	Implement the <code>JeodMemoryManager</code> class	143
memory_manager.hh	Define the <code>JeodMemoryManager</code> class, the central agent of the memory model	144
memory_manager_hide_from_trick.hh	Trick doesn't understand these	145
memory_manager_protected.cc	Implement those <code>JeodMemoryManager</code> member functions that access data members that need to be treated with care to make the memory manager thread safe	145
memory_manager_static.cc	Implement the static methods of the <code>JeodMemoryManager</code> class	146
memory_messages.cc	Implement the class <code>MemoryMessages</code>	146
memory_messages.hh	Define the class <code>MemoryMessages</code> , the class that specifies the message IDs used in the memory model	147
memory_table.hh	Define classes for representing data types	148
memory_type.cc	Implement destructors for the classes for representing data types	148
memory_type.hh	Define the abstract class <code>JeodMemoryTypeDescriptor</code> and templates that create instantiable classes that derive from <code>JeodMemoryTypeDescriptor</code>	149

Chapter 6

Module Documentation

6.1 Externally-usable macros

The supported use of the JEOD memory model is via those macros advertised as externally-usable.

Macros

- `#define JEOD_MEMORY_DEBUG 2`
Specifies the level of checking performed by the JEOD memory model.
- `#define JEOD_REGISTER_CLASS(type) jeod::JeodMemoryManager::register_class(jeod::JeodMemoryTypePreDescriptorDer
_ref())`
Register the type type with the memory manager.
- `#define JEOD_REGISTER_INCOMPLETE_CLASS(type) JEOD_REGISTER_CLASS(type)`
Register the incomplete class type with the memory manager.
- `#define JEOD_REGISTER_NONEXPORTED_CLASS(type) jeod::JeodMemoryManager::register_class(jeod::JeodMemoryType
_ref())`
Register the type type with the memory manager, but with the class marked as not exportable to the simulation engine.
- `#define JEOD_REGISTER_CHECKPOINTABLE(owner, elem_name)`
Register the data member elem_name of the owner as a Checkpointable object.
- `#define JEOD_DEREGISTER_CHECKPOINTABLE(owner, elem_name)`
Register the data member elem_name of the owner as a Checkpointable object.
- `#define JEOD_ALLOC_CLASS_MULTI_POINTER_ARRAY(nelem, type, asters) JEOD_ALLOC_ARRAY_INTERNAL(type
asters, nelem, JEOD_ALLOC_POINTER_FILL, JEOD_REGISTER_CLASS(type asters))`
Allocate an array of nelem multi-level pointers to the specified type.
- `#define JEOD_ALLOC_CLASS_POINTER_ARRAY(nelem, type) JEOD_ALLOC_CLASS_MULTI_POINTER_ARRAY(nelem,
type, *)`
Allocate an array of nelem pointers to the specified type.
- `#define JEOD_ALLOC_CLASS_ARRAY(nelem, type) JEOD_ALLOC_ARRAY_INTERNAL(type, nelem,
JEOD_ALLOC_OBJECT_FILL, JEOD_REGISTER_CLASS(type))`
Allocate an array of nelem instances of the specified structured type.
- `#define JEOD_ALLOC_PRIM_ARRAY(nelem, type) JEOD_ALLOC_ARRAY_INTERNAL(type, nelem,
JEOD_ALLOC_PRIMITIVE_FILL, JEOD_REGISTER_CLASS(type))`
Allocate nelem elements of the specified primitive type.
- `#define JEOD_ALLOC_CLASS_OBJECT(type, constr) JEOD_ALLOC_OBJECT_INTERNAL(type,
JEOD_ALLOC_OBJECT_FILL, constr, JEOD_REGISTER_CLASS(type))`
*Allocate **one** instance of the specified class.*

- `#define JEOD_ALLOC_PRIM_OBJECT(type, initial) JEOD_ALLOC_OBJECT_INTERNAL(type, JEOD_ALLOC_PRIMITIVE_FILL, (initial), JEOD_REGISTER_CLASS(type))`
Allocate **one** instance of the specified type.
- `#define JEOD_IS_ALLOCATED(ptr) jeod::JeodMemoryManager::is_allocated(jeod::jeod_alloc_get_allocated_pointer(ptr), __FILE__, __LINE__)`
Determine if *ptr* was allocated by some `JEOD_ALLOC_XXX_ARRAY` macro.
- `#define JEOD_DELETE_ARRAY(ptr) JEOD_DELETE_INTERNAL(ptr, true)`
Free memory at *ptr* that was earlier allocated with some `JEOD_ALLOC_XXX_ARRAY` macro.
- `#define JEOD_DELETE_OBJECT(ptr) JEOD_DELETE_INTERNAL(ptr, false)`
Free memory at *ptr* that was earlier allocated with some `JEOD_ALLOC_XXX_OBJECT` macro.
- `#define JEOD_DELETE_2D(ptr, size, is_array)`

6.1.1 Detailed Description

The supported use of the JEOD memory model is via those macros advertised as externally-usable.

These externally-usable macros expand into invocations of internal macros, which in turn expand into calls to methods of classes defined in the memory model.

6.1.2 Macro Definition Documentation

6.1.2.1 JEOD_ALLOC_CLASS_ARRAY

```
#define JEOD_ALLOC_CLASS_ARRAY(  
    nelem,  
    type ) JEOD_ALLOC_ARRAY_INTERNAL(type, nelem, JEOD_ALLOC_OBJECT_FILL, JEOD_REGISTER_CLASS(type))
```

Allocate an array of *nelem* instances of the specified structured *type*.

The default constructor is invoked to initialize each allocated object.

Returns

Allocated array of specified type.

Parameters

<i>nelem</i>	Size of the array.
<i>type</i>	The underlying type, which must be a structured type.

Example:

```
Foo ** foo_array = JEOD_ALLOC_CLASS_ARRAY(2, Foo);
```

This allocates two objects of the class `Foo`.

Definition at line 394 of file jeod_alloc.hh.

6.1.2.2 JEOD_ALLOC_CLASS_MULTI_POINTER_ARRAY

```
#define JEOD_ALLOC_CLASS_MULTI_POINTER_ARRAY(  
    nelem,  
    type,  
    asters ) JEOD_ALLOC_ARRAY_INTERNAL(type asters, nelem, JEOD_ALLOC_POINTER_FILL,  
JEOD_REGISTER_CLASS(type asters))
```

Allocate an array of *nelem* multi-level pointers to the specified *type*.

The *asters* are asterisks that specify the pointer level. The allocated memory is initialized via *new*.

Returns

Allocated array of specified type.

Parameters

<i>nelem</i>	Size of the array.
<i>type</i>	The underlying type, which must be a structured type.
<i>asters</i>	A bunch of asterisks.

Example:

```
Foo *** foo_array = JEOD_ALLOC_CLASS_MULTI_POINTER_ARRAY(2, Foo, **);
```

This allocates two pointers-to-pointers to the class *Foo*. Note that this does not allocate either the *Foo* objects or pointers to the *Foo* objects.

Definition at line 361 of file jeod_alloc.hh.

6.1.2.3 JEOD_ALLOC_CLASS_OBJECT

```
#define JEOD_ALLOC_CLASS_OBJECT(  
    type,  
    constr ) JEOD_ALLOC_OBJECT_INTERNAL(type, JEOD_ALLOC_OBJECT_FILL, constr, JEOD_REGISTER_CLASS(ty
```

Allocate **one** instance of the specified class.

The supplied constructor arguments, *constr*, are used as arguments to *new*. The default constructor will be invoked if the *constr* argument is the empty list; a non-default constructor will be invoked for a non-empty list.

Returns

Pointer to allocated object.

Parameters

<i>type</i>	The underlying type, which must be a structured type.
<i>constr</i>	Constructor arguments, enclosed in parentheses.

Example:

```
Foo * foo = JEOD_ALLOC_CLASS_OBJECT(Foo, (bar, baz));
```

This allocates a new object of type `Foo`, invoking the `Foo::Foo(bar, baz)` constructor.

Definition at line 431 of file `jeod_alloc.hh`.

6.1.2.4 JEOD_ALLOC_CLASS_POINTER_ARRAY

```
#define JEOD_ALLOC_CLASS_POINTER_ARRAY(  
    nelem,  
    type ) JEOD_ALLOC_CLASS_MULTI_POINTER_ARRAY(nelem, type, *)
```

Allocate an array of *nelem* pointers to the specified *type*.

The allocated memory is initialized via `new`.

Returns

Allocated array of specified type.

Parameters

<i>nelem</i>	Size of the array.
<i>type</i>	The underlying type, which must be a structured type.

Example:

```
Foo ** foo_array = JEOD_ALLOC_CLASS_POINTER_ARRAY(2, Foo);
```

This allocates two pointers to the class `Foo`. Note that this does not allocate the `Foo` objects themselves.

Definition at line 378 of file `jeod_alloc.hh`.

6.1.2.5 JEOD_ALLOC_PRIM_ARRAY

```
#define JEOD_ALLOC_PRIM_ARRAY(  
    nelem,  
    type ) JEOD_ALLOC_ARRAY_INTERNAL(type, nelem, JEOD_ALLOC_PRIMITIVE_FILL, JEOD_REGISTER_CLASS(type))
```

Allocate *nelem* elements of the specified primitive *type*.

The allocated array is zero-filled.

Returns

Allocated array of specified type.

Parameters

<i>nelem</i>	Size of the array.
<i>type</i>	The underlying type, which must be a C++ primitive type.

Example:

```
double * double_array = JEOD_ALLOC_PRIM_ARRAY(2, double);
```

This allocates an array of two doubles.

Definition at line 410 of file jeod_alloc.hh.

6.1.2.6 JEOD_ALLOC_PRIM_OBJECT

```
#define JEOD_ALLOC_PRIM_OBJECT(  
    type,  
    initial ) JEOD_ALLOC_OBJECT_INTERNAL(type, JEOD_ALLOC_PRIMITIVE_FILL, (initial),  
    JEOD_REGISTER_CLASS(type))
```

Allocate **one** instance of the specified *type*.

The object is initialized with the supplied *initial* value.

Returns

Pointer to allocated primitive.

Parameters

<i>type</i>	The underlying type, which must be a C++ primitive type.
<i>initial</i>	Initial value.

Example:

```
double * foo = JEOD_ALLOC_PRIM_OBJECT(double, 3.14159265358979323846);
```

This allocates a double and initializes it to pi.

Definition at line 447 of file jeod_alloc.hh.

6.1.2.7 JEOD_DELETE_2D

```
#define JEOD_DELETE_2D(  
    ptr,  
    size,  
    is_array )
```

Value:

```
if(ptr != nullptr)  
{  
    for(unsigned int ii = 0; ii < size; ++ii)  
    {  
        if(is_array)  
        {  
            JEOD_DELETE_ARRAY(ptr[ii]);  
        }  
        else  
        {  
            JEOD_DELETE_OBJECT(ptr[ii]);  
        }  
    }  
    JEOD_DELETE_ARRAY(ptr);  
}
```

Definition at line 501 of file jeod_alloc.hh.

6.1.2.8 JEOD_DELETE_ARRAY

```
#define JEOD_DELETE_ARRAY(  
    ptr ) JEOD_DELETE_INTERNAL(ptr, true)
```

Free memory at *ptr* that was earlier allocated with some JEOD_ALLOC_XXX_ARRAY macro.

Parameters

<i>ptr</i>	Memory to be released.
------------	------------------------

Example:

```
Foo * foo_array = JEOD_ALLOC_CLASS_ARRAY(2, Foo);  
...  
JEOD_DELETE_ARRAY(foo_array);
```

The above allocates a chunk of memory and then frees it.

Definition at line 484 of file jeod_alloc.hh.

6.1.2.9 JEOD_DELETE_OBJECT

```
#define JEOD_DELETE_OBJECT(  
    ptr ) JEOD_DELETE_INTERNAL(ptr, false)
```

Free memory at *ptr* that was earlier allocated with some JEOD_ALLOC_XXX_OBJECT macro.

Parameters

<i>ptr</i>	Memory to be released.
------------	------------------------

Example:

```
Foo * foo1 = JEOD_ALLOC_CLASS_OBJECT(Foo, ());  
...  
JEOD_DELETE_OBJECT(foo1);
```

The above allocates a chunk of memory and then frees it.

Definition at line 499 of file jeod_alloc.hh.

6.1.2.10 JEOD_DEREGISTER_CHECKPOINTABLE

```
#define JEOD_DEREGISTER_CHECKPOINTABLE(  
    owner,  
    elem_name )
```

Value:

```
jeod::JeodMemoryManager::deregister_container(  
    jeod::jeod_alloc_get_allocated_pointer(owner),  
    \                                     typeid(*owner),  
    \                                     #elem_name,  
    \                                     elem_name)
```

Register the data member *elem_name* of the *owner* as a Checkpointable object.

Assumptions and Limitations:

- The *owner* must be a pointer, typically this.
- The *owner* class must have been registered with the memory manager via JEOD_REGISTER_CLASS.
- The *elem_name* must identify a data member of the *owner* whose type derives from Checkpointable.

Parameters

<i>owner</i>	The object that contains the Checkpointable object.
<i>elem_name</i>	The Checkpointable object.

Definition at line 309 of file jeod_alloc.hh.

6.1.2.11 JEOD_IS_ALLOCATED

```
#define JEOD_IS_ALLOCATED(
    ptr ) jeod::JeodMemoryManager::is_allocated(jeod::jeod_alloc_get_allocated_pointer(ptr),
    __FILE__, __LINE__)
```

Determine if *ptr* was allocated by some JEOD_ALLOC_XXX_ARRAY macro.

Returns

true if *ptr* was allocated by this module, false otherwise.

Parameters

<i>ptr</i>	Memory to be checked.
------------	-----------------------

Example:

```
char * name;
...
if (JEOD_IS_ALLOCATED(name)) {
    JEOD_DELETE_ARRAY(name);
    name = NULL;
}
```

The above deletes the memory at *name*, but only if that memory was allocated by the JEOD memory management module.

Definition at line 468 of file jeod_alloc.hh.

6.1.2.12 JEOD_MEMORY_DEBUG

```
#define JEOD_MEMORY_DEBUG 2
```

Specifies the level of checking performed by the JEOD memory model.

- 0 - JEOD memory management off
- 1 - Error checking only
- 2 - Summary checking
- 3 - Blow-by-block account of allocation, deallocation.

Definition at line 134 of file jeod_alloc.hh.

6.1.2.13 JEOD_REGISTER_CHECKPOINTABLE

```
#define JEOD_REGISTER_CHECKPOINTABLE(
    owner,
    elem_name )
```

Value:

```
jeod::JeodMemoryManager::register_container(
    jeod::jeod_alloc_get_allocated_pointer(owner),
    \
    \
    \
    typeid(*owner),
    #elem_name,
    elem_name)
```

Register the data member *elem_name* of the *owner* as a Checkpointable object.

Assumptions and Limitations:

- The *owner* must be a pointer, typically this.
- The *owner* class must have been registered with the memory manager via JEOD_REGISTER_CLASS.
- The *elem_name* must identify a data member of the *owner* whose type derives from Checkpointable.

Parameters

<i>owner</i>	The object that contains the Checkpointable object.
<i>elem_name</i>	The Checkpointable object.

Definition at line 288 of file jeod_alloc.hh.

6.1.2.14 JEOD_REGISTER_CLASS

```
#define JEOD_REGISTER_CLASS(
    type ) jeod::JeodMemoryManager::register_class(jeod::JeodMemoryTypePreDescriptorDerived<type>(t
_ref())
```

Register the type *type* with the memory manager.

Parameters

<i>type</i>	Data type (C token, not a string).
-------------	------------------------------------

Definition at line 249 of file jeod_alloc.hh.

6.1.2.15 JEOD_REGISTER_INCOMPLETE_CLASS

```
#define JEOD_REGISTER_INCOMPLETE_CLASS(  
    type ) JEOD_REGISTER_CLASS(type)
```

Register the incomplete class *type* with the memory manager.

The type mechanism now does the "right thing" with types. This macro is deprecated.

Parameters

<i>type</i>	Data type (C token, not a string).
-------------	------------------------------------

Definition at line 260 of file jeod_alloc.hh.

6.1.2.16 JEOD_REGISTER_NONEXPORTED_CLASS

```
#define JEOD_REGISTER_NONEXPORTED_CLASS(  
    type ) jeod::JeodMemoryManager::register_class(jeod::JeodMemoryTypePreDescriptorDerived<type>(f  
_ref())
```

Register the type *type* with the memory manager, but with the class marked as not exportable to the simulation engine.

Instances of a non-exported class allocated with JEOD_ALLOC_xxx will not be registered with the simulation engine.

Parameters

<i>type</i>	Data type (C token, not a string).
-------------	------------------------------------

Definition at line 270 of file jeod_alloc.hh.

6.2 Internal macros

The internal macros act as the bridge between the externally-usable memory model macros and the publicly-visible memory model class methods.

Macros

- `#define JEOD_ALLOC_OBJECT_FILL 0xdf`
Fill pattern for non-primitive types.
- `#define JEOD_ALLOC_PRIMITIVE_FILL 0`
Fill pattern for primitive types.
- `#define JEOD_ALLOC_POINTER_FILL 0`
Fill pattern for pointer types.
- `#define JEOD_CREATE_MEMORY(is_array, nelem, fill, tentry) jeod::JeodMemoryManager::create_memory(is_array, nelem, fill, tentry, __FILE__, __LINE__)`
Allocate and register memory to be populated via placement new.
- `#define JEOD_ALLOC_ARRAY_INTERNAL(type, nelem, fill, tentry) new(JEOD_CREATE_MEMORY(true, nelem, fill, tentry)) type[nelem]`
Allocate nelem elements of pointers to the specified structured type.
- `#define JEOD_ALLOC_OBJECT_INTERNAL(type, fill, constr, tentry) new(JEOD_CREATE_MEMORY(false, 1, fill, tentry)) type constr`
Allocate an instance of the specified class using the specified constructor arguments.
- `#define JEOD_DELETE_INTERNAL(ptr, is_array)`
Free memory allocated with some JEOD_ALLOC macro.

6.2.1 Detailed Description

The internal macros act as the bridge between the externally-usable memory model macros and the publicly-visible memory model class methods.

6.2.2 Macro Definition Documentation

6.2.2.1 JEOD_ALLOC_ARRAY_INTERNAL

```
#define JEOD_ALLOC_ARRAY_INTERNAL(  
    type,  
    nelem,  
    fill,  
    tentry ) new(JEOD_CREATE_MEMORY(true, nelem, fill, tentry)) type[nelem]
```

Allocate nelem elements of pointers to the specified structured type.

Parameters

<i>type</i>	Data type.
<i>nelem</i>	Size of the array.
<i>fill</i>	Fill pattern.
<i>tentry</i>	JEOD type descriptor entry.

Definition at line 193 of file jeod_alloc.hh.

6.2.2.2 JEOD_ALLOC_OBJECT_FILL

```
#define JEOD_ALLOC_OBJECT_FILL 0xdf
```

Fill pattern for non-primitive types.

This is a nasty fill pattern that forces JEOD developers to write constructors that initialize every element of a class.

Definition at line 153 of file jeod_alloc.hh.

6.2.2.3 JEOD_ALLOC_OBJECT_INTERNAL

```
#define JEOD_ALLOC_OBJECT_INTERNAL(  
    type,  
    fill,  
    constr,  
    tentry ) new(JEOD_CREATE_MEMORY(false, 1, fill, tentry)) type constr
```

Allocate an instance of the specified class using the specified constructor arguments.

Parameters

<i>type</i>	Data type.
<i>fill</i>	Fill pattern.
<i>constr</i>	Constructor arguments, enclosed in parentheses.
<i>tentry</i>	JEOD type descriptor entry.

Definition at line 205 of file jeod_alloc.hh.

6.2.2.4 JEOD_ALLOC_POINTER_FILL

```
#define JEOD_ALLOC_POINTER_FILL 0
```

Fill pattern for pointer types.

Pointer types are initialized to null pointers. Note that this may change in the future. JEOD developers are strongly encouraged to initialize pointer arrays after allocating them.

Definition at line 171 of file jeod_alloc.hh.

6.2.2.5 JEOD_ALLOC_PRIMITIVE_FILL

```
#define JEOD_ALLOC_PRIMITIVE_FILL 0
```

Fill pattern for primitive types.

Primitive types are initialized to all zero.

Definition at line 161 of file jeod_alloc.hh.

6.2.2.6 JEOD_CREATE_MEMORY

```
#define JEOD_CREATE_MEMORY(
    is_array,
    nelem,
    fill,
    tentry ) jeod::JeodMemoryManager::create_memory(is_array, nelem, fill, tentry, ↵
__FILE__, __LINE__)
```

Allocate and register memory to be populated via placement new.

Parameters

<i>is_array</i>	Allocated as an array?
<i>nelem</i>	Size of the array.
<i>fill</i>	Fill pattern.
<i>tentry</i>	JEOD type descriptor entry.

Definition at line 181 of file jeod_alloc.hh.

6.2.2.7 JEOD_DELETE_INTERNAL

```
#define JEOD_DELETE_INTERNAL(
    ptr,
    is_array )
```

Value:

```
do
{
    \
    if(ptr != nullptr && JEOD_IS_ALLOCATED(ptr))
    {
        \
        jeod::JeodMemoryManager::destroy_memory(
        jeod::jeod_alloc_get_allocated_pointer(ptr), \
        \
        is_array,
        \
        \
        __FILE__,
        \
    }
```

```
                                __LINE__);
    ptr = nullptr;
}
} while(0)
```

Free memory allocated with some JEOD_ALLOC macro.

Depends on

`dynamic_cast<void*>(ptr)`

yielding a pointer to the most derived object pointed to by *ptr*. See ISO/IEC 14882:2003 section 5.2.7.7.

Parameters

<i>ptr</i>	Memory to be released.
<i>is_array</i>	True for DELETE_ARRAY, false for DELETE_OBJECT.

Definition at line 218 of file jeod_alloc.hh.

6.3 Support classes

The memory model classes are the workhorses of the JEOD memory model.

Namespaces

- [jeod](#)
Namespace jeod.

Macros

- [#define __STDC_LIMIT_MACROS](#)

6.3.1 Detailed Description

The memory model classes are the workhorses of the JEOD memory model.

6.3.2 Macro Definition Documentation

6.3.2.1 __STDC_LIMIT_MACROS

```
#define __STDC_LIMIT_MACROS
```

Definition at line 35 of file `memory_manager_protected.cc`.

6.4 Models

Modules

- [Utils](#)

6.4.1 Detailed Description

6.5 Utils

Modules

- [Memory](#)

6.5.1 Detailed Description

6.6 Memory

Modules

- [Externally-usable macros](#)

The supported use of the JEOD memory model is via those macros advertised as externally-usable.

- [Internal macros](#)

The internal macros act as the bridge between the externally-usable memory model macros and the publicly-visible memory model class methods.

- [Support classes](#)

The memory model classes are the workhorses of the JEOD memory model.

Files

- file [class_declarations.hh](#)

Forward declarations of classes defined in the utils/memory model.

- file [jeod_alloc.hh](#)

Define JEOD memory allocation macros.

- file [jeod_alloc_construct_destruct.hh](#)

Define templates for use by [jeod_alloc.hh](#).

- file [jeod_alloc_get_allocated_pointer.hh](#)

Define function template [jeod_alloc_get_allocated_pointer](#).

- file [memory_attributes_templates.hh](#)

Define the class template [JeodSimEngineAttributes](#).

- file [memory_item.hh](#)

Define the class [JeodMemoryItem](#).

- file [memory_manager.hh](#)

Define the [JeodMemoryManager](#) class, the central agent of the memory model.

- file [memory_manager_hide_from_trick.hh](#)

Trick doesn't understand these.

- file [memory_messages.hh](#)

Define the class [MemoryMessages](#), the class that specifies the message IDs used in the memory model.

- file [memory_table.hh](#)

Define classes for representing data types.

- file [memory_type.hh](#)

Define the abstract class [JeodMemoryTypeDescriptor](#) and templates that create instantiable classes that derive from [JeodMemoryTypeDescriptor](#).

- file [memory_item.cc](#)

Implement the [JeodMemoryItem](#) class.

- file [memory_manager.cc](#)

Implement the [JeodMemoryManager](#) class.

- file [memory_manager_protected.cc](#)

Implement those [JeodMemoryManager](#) member functions that access data members that need to be treated with care to make the memory manager thread safe.

- file [memory_manager_static.cc](#)

Implement the static methods of the [JeodMemoryManager](#) class.

- file [memory_messages.cc](#)

Implement the class [MemoryMessages](#).

- file [memory_type.cc](#)

Implement destructors for the classes for representing data types.

Namespaces

- [jeod](#)

Namespace jeod.

6.6.1 Detailed Description

Chapter 7

Namespace Documentation

7.1 jeod Namespace Reference

Namespace jeod.

Data Structures

- class [JeodAllocHelperAllocatedPointer](#)
Class template that provides a static function cast that casts a pointer to an object of type `T` to a `void` pointer.*
- class [JeodAllocHelperAllocatedPointer< T, true >](#)
Partial instantiation of [JeodAllocHelperAllocatedPointer](#) for polymorphic classes.
- class [JeodAllocHelperConstructDestruct](#)
Class template that provides static functions `construct` and `destruct` that construct an array of objects.
- class [JeodAllocHelperConstructDestruct< T, false, is_abstract >](#)
Partial instantiation for non-classes.
- class [JeodAllocHelperConstructDestruct< T, true, false >](#)
Partial instantiation for non-abstract classes.
- class [JeodMemoryItem](#)
A [JeodMemoryItem](#) contains metadata about some chunk of allocated memory.
- class [JeodMemoryManager](#)
This class provides the interface between the macros in [jeod_alloc.hh](#) and the rest of the JEOD memory model.
- class [JeodMemoryReflectiveTable](#)
A [JeodMemoryReflectiveTable](#) maps strings to themselves.
- class [JeodMemoryTable](#)
A [JeodMemoryTable](#) maps strings to values with a coordinated map/vector pair.
- class [JeodMemoryTableClonable](#)
A [JeodMemoryTableClonable](#) is a [JeodMemoryTable](#) that implements the required [clone\(\)](#) functionality by invoking the `ValueType`'s [clone\(\)](#) method to create a clone of the input value.
- class [JeodMemoryTableCopyable](#)
A [JeodMemoryTableCopyable](#) is a [JeodMemoryTable](#) that implements the required [clone\(\)](#) functionality by invoking the `ValueType`'s copy constructor to create a clone of the input value.
- class [JeodMemoryTypeDescriptor](#)
Abstract class for managing data allocated as some specific type.
- class [JeodMemoryTypeDescriptorDerived](#)
Extends [JeodMemoryTypeDescriptor](#) to describe a specific type.

- class [JeodMemoryTypePreDescriptor](#)
Abstract class for describing a type without necessarily needing to create a [JeodMemoryTypeDescriptor](#) of that type.
- class [JeodMemoryTypePreDescriptorDerived](#)
A [JeodMemoryTypePreDescriptorDerived](#) describes a Type.
- class [JeodSimEngineAttributes](#)
Class template to construct a simulation engine attributes object that represents some type.
- class [JeodSimEngineAttributes< Type *, false >](#)
Partial template instantiation of [JeodSimEngineAttributes](#) for a pointer type.
- class [JeodSimEngineAttributes< Type, true >](#)
Partial template instantiation of [JeodSimEngineAttributes](#) for a class.
- class [JeodSimEngineAttributes< void *, false >](#)
Template specialization of [JeodSimEngineAttributes](#) for void.*
- class [MemoryMessages](#)
Declares messages associated with the integration test model.

Typedefs

- using [AllocTable](#) = std::map< const void *, [JeodMemoryItem](#) >
An AllocTable maps memory addresses to memory descriptions.
- using [TypeTable](#) = [JeodMemoryTableClonable](#)< [JeodMemoryTypeDescriptor](#) >
The type type itself is a memory table with copy implemented by clone().

Functions

- template<typename T >
void * [jeod_alloc_construct_array](#) (std::size_t nelem, void *addr)
Construct an array of objects of type T.
- template<typename T >
void [jeod_alloc_destruct_array](#) (std::size_t nelem, void *addr)
Destruct an array of objects of type T.
- template<typename T >
void * [jeod_alloc_get_allocated_pointer](#) (T *pointer)
Cast a pointer to some object to a pointer to void such that a pointer to a polymorphic object, downcast to a base class pointer, becomes a pointer to the original object, but also such that a pointer to an instance of a non-polymorphic class or a pointer to a non-class type is handled correctly.*

Variables

- static constexpr uint32_t [MAGIC0](#) = 0x2203992c
- static constexpr uint32_t [MAGIC1](#) = 0x6c052d84

7.1.1 Detailed Description

Namespace jeod.

7.1.2 Typedef Documentation

7.1.2.1 AllocTable

```
using jeod::AllocTable = typedef std::map<const void *, JeodMemoryItem>
```

An AllocTable maps memory addresses to memory descriptions.

Definition at line 76 of file memory_manager_hide_from_trick.hh.

7.1.2.2 TypeTable

```
using jeod::TypeTable = typedef JeodMemoryTableClonable<JeodMemoryTypeDescriptor>
```

The type type itself is a memory table with copy implemented by clone().

Definition at line 81 of file memory_manager_hide_from_trick.hh.

7.1.3 Function Documentation

7.1.3.1 jeod_alloc_construct_array()

```
template<typename T >
void* jeod::jeod_alloc_construct_array (
    std::size_t nelem,
    void * addr ) [inline]
```

Construct an array of objects of type *T*.

Template Parameters

<i>T</i>	Pointed-to type.
----------	------------------

Parameters

<i>nelem</i>	Number of elements in the array
<i>addr</i>	Address to be constructed

Returns

Constructed array.

Definition at line 202 of file jeod_alloc_construct_destruct.hh.

7.1.3.2 jeod_alloc_destruct_array()

```
template<typename T >
void jeod::jeod_alloc_destruct_array (
    std::size_t nelem,
    void * addr ) [inline]
```

Destruct an array of objects of type *T*.

Template Parameters

<i>T</i>	Pointed-to type.
----------	------------------

Parameters

<i>nelem</i>	Number of elements in the array
<i>addr</i>	Address to be destructed

Definition at line 214 of file `jeod_alloc_construct_destruct.hh`.

7.1.3.3 jeod_alloc_get_allocated_pointer()

```
template<typename T >
void* jeod::jeod_alloc_get_allocated_pointer (
    T * pointer ) [inline]
```

Cast a pointer to some object to a pointer to void* such that a pointer to a polymorphic object, downcast to a base class pointer, becomes a pointer to the original object, but also such that a pointer to an instance of a non-polymorphic class or a pointer to a non-class type is handled correctly.

Template Parameters

<i>T</i>	Pointed-to type.
----------	------------------

Parameters

<i>pointer</i>	Pointer to be cast to void*.
----------------	------------------------------

Usage:

```
jeod_alloc_get_allocated_pointer(pointer)
```

Note that the template parameter does not need to be specified. The compiler automatically determines the type.

Assumptions and limitations:

- The function argument *pointer* is a pointer.
- The pointer is not *cv* qualified (i.e., neither *const* nor *volatile*).
- Inheritance is public for polymorphic classes.
- `jeod_alloc_get_allocated_pointer(&array[1])`
does not return a pointer to

`&array[0]`

.

Definition at line 153 of file `jeod_alloc_get_allocated_pointer.hh`.

References `jeod::JeodAllocHelperAllocatedPointer< T, is_poly >::cast()`.

Referenced by `jeod::JeodMemoryTypeDescriptorDerived< Type >::most_derived_pointer()`.

7.1.4 Variable Documentation

7.1.4.1 MAGIC0

```
constexpr uint32_t jeod::MAGIC0 = 0x2203992c [static]
```

Definition at line 55 of file `memory_manager.cc`.

Referenced by `jeod::JeodMemoryManager::allocate_memory()`, and `jeod::JeodMemoryManager::free_memory()`.

7.1.4.2 MAGIC1

```
constexpr uint32_t jeod::MAGIC1 = 0x6c052d84 [static]
```

Definition at line 56 of file `memory_manager.cc`.

Referenced by `jeod::JeodMemoryManager::allocate_memory()`, and `jeod::JeodMemoryManager::free_memory()`.

Chapter 8

Data Structure Documentation

8.1 jeod::JeodMemoryTypeDescriptor::attr Struct Reference

The simulation engine attributes that describe the type.trick_io(**)

```
#include <memory_type.hh>
```

8.1.1 Detailed Description

The simulation engine attributes that describe the type.trick_io(**)

Definition at line 348 of file memory_type.hh.

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

- [memory_type.hh](#)

8.2 jeod::JeodAllocHelperAllocatedPointer< T, is_poly > Class Template Reference

Class template that provides a static function *cast* that casts a pointer to an object of type *T* to a void* pointer.

```
#include <jeod_alloc_get_allocated_pointer.hh>
```

Static Public Member Functions

- static void * [cast](#) (T *pointer)
Cast a pointer to a non-polymorphic class via an implicit cast.

8.2.1 Detailed Description

```
template<typename T, bool is_poly>  
class jeod::JeodAllocHelperAllocatedPointer< T, is_poly >
```

Class template that provides a static function *cast* that casts a pointer to an object of type *T* to a void* pointer.

Template Parameters

<i>T</i>	Type
<i>is_poly</i>	True if the type T is a polymorphic class.

This class template is intended for used by `jeod_alloc_get_allocated_pointer`. Use in any other context is caveat emptor.

This template provides a default implementation for non-polymorphic classes (`is_poly == false`) that uses implicit cast. The partial template instantiation that immediately follows provides n an implementation that uses `dynamic_cast` when `is_poly` is true.

Definition at line 92 of file `jeod_alloc_get_allocated_pointer.hh`.

8.2.2 Member Function Documentation

8.2.2.1 `cast()`

```
template<typename T , bool is_poly>
static void* jeod::JeodAllocHelperAllocatedPointer< T, is_poly >::cast (
    T * pointer ) [inline], [static]
```

Cast a pointer to a non-polymorphic class via an implicit cast.

Returns

Input pointer cast to void* via implicit cast.

Parameters

<i>pointer</i>	Pointer
----------------	---------

Definition at line 100 of file `jeod_alloc_get_allocated_pointer.hh`.

Referenced by `jeod::jeod_alloc_get_allocated_pointer()`.

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

- [jeod_alloc_get_allocated_pointer.hh](#)

8.3 `jeod::JeodAllocHelperAllocatedPointer< T, true >` Class Template Reference

Partial instantiation of [JeodAllocHelperAllocatedPointer](#) for polymorphic classes.

```
#include <jeod_alloc_get_allocated_pointer.hh>
```

Static Public Member Functions

- static void * [cast](#) (T *pointer)

Cast a pointer to a non-class object via `dynamic_cast`.

8.3.1 Detailed Description

```
template<typename T>
class jeod::JeodAllocHelperAllocatedPointer< T, true >
```

Partial instantiation of [JeodAllocHelperAllocatedPointer](#) for polymorphic classes.

This class uses the fact that `dynamic_cast<void*>(ptr)` results in a pointer to the most derived object pointed to by `ptr`. See ISO/IEC 14882:2003 section 5.2.7.7 for details.

Definition at line 112 of file `jeod_alloc_get_allocated_pointer.hh`.

8.3.2 Member Function Documentation

8.3.2.1 [cast\(\)](#)

```
template<typename T >
static void* jeod::JeodAllocHelperAllocatedPointer< T, true >::cast (
    T * pointer ) [inline], [static]
```

Cast a pointer to a non-class object via `dynamic_cast`.

Returns

Input pointer cast to `void*` via `dynamic_cast`.

Parameters

<i>pointer</i>	Pointer
----------------	---------

Definition at line 120 of file `jeod_alloc_get_allocated_pointer.hh`.

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

- [jeod_alloc_get_allocated_pointer.hh](#)

8.4 jeod::JeodAllocHelperConstructDestruct< T, is_class, is_abstract > Class Template Reference

Class template that provides static functions *construct* and *destruct* that construct an array of objects.

```
#include <jeod_alloc_construct_destruct.hh>
```

Static Public Member Functions

- static void * **construct** (std::size_t nelem, void *addr)
Construct an array of objects.
- static void **destruct** (std::size_t nelem, void *addr)
Destruct an array of objects.

8.4.1 Detailed Description

```
template<typename T, bool is_class, bool is_abstract>
class jeod::JeodAllocHelperConstructDestruct< T, is_class, is_abstract >
```

Class template that provides static functions *construct* and *destruct* that construct an array of objects.

Template Parameters

<i>T</i>	Type
<i>is_class</i>	True if the type T is a class.
<i>is_abstract</i>	True if the type T is an abstract class.

This class template is intended for used by `jeod_alloc_construct_array` and `jeod_alloc_destruct_array`. Use in any other context is caveat emptor.

This template provides do-nothing implementations, which is about all one can do for arrays of abstract objects (which can't exist).

Definition at line 102 of file `jeod_alloc_construct_destruct.hh`.

8.4.2 Member Function Documentation

8.4.2.1 `construct()`

```
template<typename T, bool is_class, bool is_abstract>
static void* jeod::JeodAllocHelperConstructDestruct< T, is_class, is_abstract >::construct (
    std::size_t nelem,
    void * addr ) [inline], [static]
```

Construct an array of objects.

Returns

Constructed array.

Parameters

<i>nelem</i>	Number of elements in the array
<i>addr</i>	Address to be constructed

Definition at line 111 of file jeod_alloc_construct_destruct.hh.

8.4.2.2 destruct()

```
template<typename T, bool is_class, bool is_abstract>
static void jeod::JeodAllocHelperConstructDestruct< T, is_class, is_abstract >::destruct (
    std::size_t nelem,
    void * addr ) [inline], [static]
```

Destruct an array of objects.

Parameters

<i>nelem</i>	Number of elements in the array
<i>addr</i>	Address to be destructed

Definition at line 121 of file jeod_alloc_construct_destruct.hh.

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

- [jeod_alloc_construct_destruct.hh](#)

8.5 jeod::JeodAllocHelperConstructDestruct< T, false, is_abstract > Class Template Reference

Partial instantiation for non-classes.

```
#include <jeod_alloc_construct_destruct.hh>
```

Static Public Member Functions

- static void * [construct](#) (std::size_t nelem, void *addr)
Construct an array of objects.
- static void [destruct](#) (std::size_t nelem, void *addr)
Destruct an array of objects.

8.5.1 Detailed Description

```
template<typename T, bool is_abstract>
class jeod::JeodAllocHelperConstructDestruct< T, false, is_abstract >
```

Partial instantiation for non-classes.

Definition at line 130 of file jeod_alloc_construct_destruct.hh.

8.5.2 Member Function Documentation

8.5.2.1 `construct()`

```
template<typename T , bool is_abstract>
static void* jeod::JeodAllocHelperConstructDestruct< T, false, is_abstract >::construct (
    std::size_t nelem,
    void * addr ) [inline], [static]
```

Construct an array of objects.

Returns

Constructed array.

Parameters

<i>nelem</i>	Number of elements in the array
<i>addr</i>	Address to be constructed

Definition at line 139 of file `jeod_alloc_construct_destruct.hh`.

8.5.2.2 `destruct()`

```
template<typename T , bool is_abstract>
static void jeod::JeodAllocHelperConstructDestruct< T, false, is_abstract >::destruct (
    std::size_t nelem,
    void * addr ) [inline], [static]
```

Destruct an array of objects.

Parameters

<i>nelem</i>	Number of elements in the array
<i>addr</i>	Address to be destructed

Definition at line 150 of file `jeod_alloc_construct_destruct.hh`.

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

- [jeod_alloc_construct_destruct.hh](#)

8.6 jeod::JeodAllocHelperConstructDestruct< T, true, false > Class Template Reference

Partial instantiation for non-abstract classes.

```
#include <jeod_alloc_construct_destruct.hh>
```

Static Public Member Functions

- static void * [construct](#) (std::size_t nelem, void *addr)
Construct an array of objects.
- static void [destruct](#) (std::size_t nelem, void *addr)
Destruct an array of objects.

8.6.1 Detailed Description

```
template<typename T>
class jeod::JeodAllocHelperConstructDestruct< T, true, false >
```

Partial instantiation for non-abstract classes.

Definition at line 159 of file jeod_alloc_construct_destruct.hh.

8.6.2 Member Function Documentation

8.6.2.1 construct()

```
template<typename T >
static void* jeod::JeodAllocHelperConstructDestruct< T, true, false >::construct (
    std::size_t nelem,
    void * addr ) [inline], [static]
```

Construct an array of objects.

Returns

Constructed array.

Parameters

<i>nelem</i>	Number of elements in the array
<i>addr</i>	Address to be constructed

Definition at line 168 of file jeod_alloc_construct_destruct.hh.

8.6.2.2 destruct()

```
template<typename T >
static void jeod::JeodAllocHelperConstructDestruct< T, true, false >::destruct (
    std::size_t nelem,
    void * addr ) [inline], [static]
```

Destruct an array of objects.

Parameters

<i>nelem</i>	Number of elements in the array
<i>addr</i>	Address to be destructed

Definition at line 178 of file jeod_alloc_construct_destruct.hh.

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

- [jeod_alloc_construct_destruct.hh](#)

8.7 jeod::JeodMemoryItem Class Reference

A [JeodMemoryItem](#) contains metadata about some chunk of allocated memory.

```
#include <memory_item.hh>
```

Public Types

- enum [Flags](#) {
[PlacementNew](#) = 1, [IsArray](#) = 2, [IsGuarded](#) = 4, [IsStructured](#) = 8,
[IsRegistered](#) = 16, [CheckPointed](#) = 32 }
Identifies by name the bit flags the comprise a JeodMemoryItem::flag.

Public Member Functions

- [JeodMemoryItem](#) ()=default
- [JeodMemoryItem](#) (bool placement_new, bool is_array, bool is_guarded, bool is_structured, unsigned int nelems_in, unsigned int type_idx, unsigned int alloc_idx)
Construct a JeodMemoryItem.
- [~JeodMemoryItem](#) ()=default
- void [set_unique_id](#) (uint32_t id)
Set the unique identifier.
- void [set_is_registered](#) (bool value)
Set the is_registered flag.
- uint32_t [get_nelems](#) () const
Access the array size.
- uint32_t [get_alloc_index](#) () const
Access the allocation information index.

- uint32_t [get_unique_id](#) () const
Access the unique identifier.
- uint32_t [get_descriptor_index](#) () const
Access the type descriptor index.
- bool [get_is_array](#) () const
Access the is_array flag.
- bool [get_is_guarded](#) () const
Access the is_guarded flag.
- bool [get_placement_new](#) () const
Access the placement_new flag.
- bool [is_structured_data](#) () const
Is the associated data a structure/class?
- bool [get_is_registered](#) () const
Access the checkpointed flag.
- bool [get_checkpointed](#) () const
Access the checkpointed flag.

Static Private Member Functions

- static uint8_t [construct_flags](#) (bool placement_new, bool is_array, bool is_guarded, bool is_structured)
Construct the flags for a new [JeodMemoryItem](#).

Private Attributes

- uint32_t [nelems](#) {}
Number of elements in the allocated array.
- uint32_t [alloc_info_index](#) {}
Allocation information index, max of 2^{32-2} tracked locations.
- uint32_t [unique_id](#) {}
Unique identifier, max of 2^{32-2} allocations (zero is not used).
- uint16_t [descriptor_index_hi](#) {}
High order bits of the descriptor index.
- uint8_t [descriptor_index_lo](#) {}
Low order bits of the descriptor index.
- uint8_t [flags](#) {}
Flags indicating whether.

8.7.1 Detailed Description

A [JeodMemoryItem](#) contains metadata about some chunk of allocated memory.

This is a simple datatype that contains POD elements only. All data members are private and are accessible only through getters; the members are essentially constant. The only way to change the values is via a wholesale copy.

Definition at line 86 of file `memory_item.hh`.

8.7.2 Member Enumeration Documentation

8.7.2.1 Flags

```
enum jeod::JeodMemoryItem::Flags
```

Identifies by name the bit flags the comprise a `JeodMemoryItem::flag`.

Enumerator

PlacementNew	Was the item constructed with placement new? There is no functional placement delete in C++.
IsArray	Was the item an array constructed via new []? This addresses the delete[] versus delete issue.
IsGuarded	Is the allocated buffer surrounded by guard words? This flag is always false in regular new mode.
IsStructured	Is the item a class (versus a primitive type)? Classes add several other twists.
IsRegistered	Has the item been registered with the simulation engine?
CheckPointed	Reserved for future work, as are flag bits 6 and 7 (64 and 128).

Definition at line 93 of file `memory_item.hh`.

8.7.3 Constructor & Destructor Documentation

8.7.3.1 JeodMemoryItem() [1/2]

```
jeod::JeodMemoryItem::JeodMemoryItem ( ) [default]
```

8.7.3.2 JeodMemoryItem() [2/2]

```
jeod::JeodMemoryItem::JeodMemoryItem (
    bool placement_new,
    bool is_array,
    bool is_guarded,
    bool is_structured,
    unsigned int nelems_in,
    unsigned int type_idx,
    unsigned int alloc_idx )
```

Construct a [JeodMemoryItem](#).

The data are essentially constant, so the only viable way to set elements to meaningful values is via this non-default constructor.

Parameters

in	<i>placement_new</i>	Constructed with placement new?
in	<i>is_array</i>	Constructed with new []?
in	<i>is_guarded</i>	Is the item an array?
in	<i>is_structured</i>	Is the item a structured data type?
in	<i>nelems_in</i>	Number of elements
in	<i>type_idx</i>	Type descriptor (index)
in	<i>alloc_idx</i>	Macro invocation info (index)

Definition at line 86 of file `memory_item.cc`.

8.7.3.3 ~JeodMemoryItem()

```
jeod::JeodMemoryItem::~JeodMemoryItem ( ) [default]
```

8.7.4 Member Function Documentation

8.7.4.1 construct_flags()

```
uint8_t jeod::JeodMemoryItem::construct_flags (
    bool placement_new,
    bool is_array,
    bool is_guarded,
    bool is_structured ) [static], [private]
```

Construct the flags for a new [JeodMemoryItem](#).

Returns

Constructed flags

Parameters

in	<i>placement_new</i>	Constructed with placement new?
in	<i>is_array</i>	Constructed with new []?
in	<i>is_guarded</i>	Is the item an array?
in	<i>is_structured</i>	Is the item a structured data type?

Definition at line 47 of file `memory_item.cc`.

References `flags`, `IsArray`, `IsGuarded`, `IsStructured`, and `PlacementNew`.

8.7.4.2 get_alloc_index()

```
uint32_t jeod::JeodMemoryItem::get_alloc_index ( ) const [inline]
```

Access the allocation information index.

Returns

Allocation information index

Definition at line 246 of file memory_item.hh.

References alloc_info_index.

Referenced by jeod::JeodMemoryManager::destroy_memory_internal(), jeod::JeodMemoryManager::generate_shutdown_report(), and jeod::JeodMemoryManager::restart_clear_memory().

8.7.4.3 get_checkpointed()

```
bool jeod::JeodMemoryItem::get_checkpointed ( ) const [inline]
```

Access the checkpointed flag.

Returns

Checkpointed?

Definition at line 309 of file memory_item.hh.

References CheckPointed, and flags.

8.7.4.4 get_descriptor_index()

```
uint32_t jeod::JeodMemoryItem::get_descriptor_index ( ) const [inline]
```

Access the type descriptor index.

Returns

Type descriptor index

Definition at line 255 of file memory_item.hh.

References descriptor_index_hi, and descriptor_index_lo.

8.7.4.5 get_is_array()

```
bool jeod::JeodMemoryItem::get_is_array ( ) const [inline]
```

Access the is_array flag.

Returns

Allocated as an array?

Definition at line 273 of file memory_item.hh.

References flags, and IsArray.

Referenced by jeod::JeodMemoryManager::destroy_memory_internal(), jeod::JeodMemoryManager::restart_↔clear_memory(), and jeod::JeodMemoryTypeDescriptor::type_spec().

8.7.4.6 get_is_guarded()

```
bool jeod::JeodMemoryItem::get_is_guarded ( ) const [inline]
```

Access the is_guarded flag.

Returns

Is allocated memory guarded?

Definition at line 282 of file memory_item.hh.

References flags, and IsGuarded.

Referenced by jeod::JeodMemoryManager::destroy_memory_internal(), and jeod::JeodMemoryManager::restart_↔_clear_memory().

8.7.4.7 get_is_registered()

```
bool jeod::JeodMemoryItem::get_is_registered ( ) const [inline]
```

Access the checkpointed flag.

Returns

Registered with sim engine?

Definition at line 300 of file memory_item.hh.

References flags, and IsRegistered.

Referenced by jeod::JeodMemoryManager::destroy_memory_internal(), jeod::JeodMemoryManager::restart_↔clear_memory(), and jeod::JeodMemoryManager::~JeodMemoryManager().

8.7.4.8 `get_nelems()`

```
uint32_t jeod::JeodMemoryItem::get_nelems ( ) const [inline]
```

Access the array size.

Returns

Array size

Definition at line 228 of file `memory_item.hh`.

References `nelems`.

Referenced by `jeod::JeodMemoryManager::add_allocation_atomic()`, `jeod::JeodMemoryTypeDescriptor::buffer↵_end()`, `jeod::JeodMemoryTypeDescriptor::buffer_size()`, `jeod::JeodMemoryManager::destroy_memory_internal()`, `jeod::JeodMemoryManager::restart_clear_memory()`, and `jeod::JeodMemoryTypeDescriptor::type_spec()`.

8.7.4.9 `get_placement_new()`

```
bool jeod::JeodMemoryItem::get_placement_new ( ) const [inline]
```

Access the `placement_new` flag.

Returns

Allocated for placement new?

Definition at line 264 of file `memory_item.hh`.

References `flags`, and `PlacementNew`.

Referenced by `jeod::JeodMemoryManager::destroy_memory_internal()`, and `jeod::JeodMemoryManager::restart↵_clear_memory()`.

8.7.4.10 `get_unique_id()`

```
uint32_t jeod::JeodMemoryItem::get_unique_id ( ) const [inline]
```

Access the unique identifier.

Returns

Unique identifier

Definition at line 237 of file `memory_item.hh`.

References `unique_id`.

8.7.4.11 is_structured_data()

```
bool jeod::JeodMemoryItem::is_structured_data ( ) const [inline]
```

Is the associated data a structure/class?

Returns

True for structured data

Definition at line 291 of file memory_item.hh.

References flags, and IsStructured.

8.7.4.12 set_is_registered()

```
void jeod::JeodMemoryItem::set_is_registered (
    bool value )
```

Set the is_registered flag.

Parameters

in	<i>value</i>	New value of the is_registered flag
----	--------------	-------------------------------------

Definition at line 132 of file memory_item.cc.

References flags, and IsRegistered.

8.7.4.13 set_unique_id()

```
void jeod::JeodMemoryItem::set_unique_id (
    uint32_t id )
```

Set the unique identifier.

Parameters

in	<i>id</i>	Unique identifier
----	-----------	-------------------

Definition at line 105 of file memory_item.cc.

References jeod::MemoryMessages::internal_error, and unique_id.

Referenced by jeod::JeodMemoryManager::register_memory_internal().

8.7.5 Field Documentation

8.7.5.1 alloc_info_index

```
uint32_t jeod::JeodMemoryItem::alloc_info_index {} [private]
```

Allocation information index, max of $2^{32}-2$ tracked locations.

The allocation information is a string of the form "file.cc:line#" that indicates where in the code the data was allocated. The underlying string is maintained in the global memory manager's string table.`trick_units(-)`

Definition at line 190 of file `memory_item.hh`.

Referenced by `get_alloc_index()`.

8.7.5.2 descriptor_index_hi

```
uint16_t jeod::JeodMemoryItem::descriptor_index_hi {} [private]
```

High order bits of the descriptor index.

The `descriptor_index` specifies the type decriptor that describes the data. The underlying descriptor is maintained in the global memory manager's type descriptor table.`trick_units(-)`

Definition at line 205 of file `memory_item.hh`.

Referenced by `get_descriptor_index()`.

8.7.5.3 descriptor_index_lo

```
uint8_t jeod::JeodMemoryItem::descriptor_index_lo {} [private]
```

Low order bits of the descriptor index.

`trick_units(-)`

Definition at line 210 of file `memory_item.hh`.

Referenced by `get_descriptor_index()`.

8.7.5.4 flags

```
uint8_t jeod::JeodMemoryItem::flags {} [private]
```

Flags indicating whether.

- The data was constructed with default new or placement new
- The data was allocated as an array or as a single object
- The allocated are guarded
- The data is a structured or non-structured data type
- The data has been checkpointed (future)
- Plus three more future-use spares.trick_units(-)

Definition at line 221 of file memory_item.hh.

Referenced by construct_flags(), get_checkpointed(), get_is_array(), get_is_guarded(), get_is_registered(), get_placement_new(), is_structured_data(), and set_is_registered().

8.7.5.5 nelems

```
uint32_t jeod::JeodMemoryItem::nelems {} [private]
```

Number of elements in the allocated array.

trick_units(-)

Definition at line 182 of file memory_item.hh.

Referenced by get_nelems().

8.7.5.6 unique_id

```
uint32_t jeod::JeodMemoryItem::unique_id {} [private]
```

Unique identifier, max of $2^{32}-2$ allocations (zero is not used).

The unique identifier forms the basis of the item name sent to the simulation engine for this memory item.trick_units(-)

Definition at line 197 of file memory_item.hh.

Referenced by get_unique_id(), and set_unique_id().

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

- [memory_item.hh](#)
- [memory_item.cc](#)

8.8 jeod::JeodMemoryManager Class Reference

This class provides the interface between the macros in [jeod_alloc.hh](#) and the rest of the JEOD memory model.

```
#include <memory_manager.hh>
```

Data Structures

- struct [TypeEntry](#)

The type table is indexed by an integer and contains type descriptors.

Public Types

- enum [DebugLevel](#) { [Debug_off](#) = 0, [Summary_only](#) = 1, [Error_details](#) = 2, [Full_details](#) = 3 }

The memory manager as a whole and individual operations have a debug level.

- enum [NameType](#) { [Typeid_type_name](#) = 0, [Demangled_type_name](#) = 1 }

The type lookup by type name needs to know whether the provided name is a typeid name or a demangled name.

Public Member Functions

- [JeodMemoryManager](#) ()=delete
- [JeodMemoryManager](#) (const [JeodMemoryManager](#) &)=delete
- [JeodMemoryManager](#) & operator= (const [JeodMemoryManager](#) &)=delete
- [JeodMemoryManager](#) ([JeodMemoryInterface](#) &)
Construct a MemoryManager object.
- virtual [~JeodMemoryManager](#) ()
Destruct a MemoryManager object.
- void [restart_clear_memory](#) ()
Wipe out all allocated memory in anticipation of restoring the memory in some previously recording checkpoint file.
- void [restart_reallocate](#) (const std::string &mangled_type_name, uint32_t unique_id, uint32_t nelements, bool is_array)
Restore one chunk of allocated memory per a checkpoint file entry.

Static Public Member Functions

- static const [TypeEntry](#) [register_class](#) ([JeodMemoryTypePreDescriptor](#) &tdesc)
Register a class with the memory manager.
- static const [JeodMemoryTypeDescriptor](#) * [get_type_descriptor](#) (const std::type_info &typeid_info)
Get a type descriptor from the memory manager's type table.
- static const [JeodMemoryTypeDescriptor](#) * [get_type_descriptor](#) ([NameType](#) name_type, const std::string &name)
Get a type descriptor from the memory manager's type table.
- static void * [create_memory](#) (bool is_array, unsigned int nelems, int fill, const [TypeEntry](#) &tentry, const char *file, unsigned int line)
Allocate memory and register the allocated memory with JEOD.
- static bool [is_allocated](#) (const void *addr, const char *file, unsigned int line)
Query whether some address was allocated by JEOD.
- static void [destroy_memory](#) (void *addr, bool delete_array, const char *file, unsigned int line)

Destroy memory previously registered with JEOD.

- static void [register_container](#) (const void *container, const std::type_info &container_type, const std::string &elem_name, JeodCheckpointable &checkpointable)

Register a checkpointable object with the memory manager.

- static void [deregister_container](#) (const void *container, const std::type_info &container_type, const std::string &elem_name, JeodCheckpointable &checkpointable)

Deregister all checkpointable object contained within some object.

- static void [set_mode](#) (JeodSimulationInterface::Mode new_mode)

Set the memory manager's simulation interface mode.

- static void [set_debug_level](#) (unsigned int level)

Set the debug level.

- static void [set_debug_level](#) (DebugLevel level)

Set the debug level.

- static void [set_guard_enabled](#) (bool value)

Set the guard_enabled flag.

- static bool [is_table_empty](#) ()

Query whether all allocated memory has been freed.

Private Types

- using [AllocTable](#) = std::map< const void *, [JeodMemoryItem](#) >
An AllocTable maps memory addresses to memory descriptions.
- using [TypeTable](#) = [JeodMemoryTableClonable](#)< [JeodMemoryTypeDescriptor](#) >
The type type itself is a memory table with copy implemented by clone().

Private Member Functions

- void [generate_shutdown_report](#) ()
Generate a shutdown report.
- void * [create_memory_internal](#) (bool is_array, unsigned int nelems, int fill, const [TypeEntry](#) &tentry, const char *file, unsigned int line)
Allocate memory for use with placement new and register that memory with the memory manager and with the simulation engine.
- void [register_memory_internal](#) (const void *addr, uint32_t unique_id, bool placement_new, bool is_array, unsigned int nelems, const [TypeEntry](#) &tentry, const char *file, unsigned int line)
Allocate memory if that was not already done by the caller and register the memory with JEOD and with an external agent.
- bool [is_allocated_internal](#) (const void *addr, const char *file, unsigned int line)
Query whether some address was allocated by JEOD.
- void [destroy_memory_internal](#) (void *addr, bool delete_array, const char *file, unsigned int line)
Destroy a chunk of memory and knowledge about it.
- void [set_mode_internal](#) (JeodSimulationInterface::Mode new_mode)
Set the mode and perform mode transitions.
- void [begin_atomic_block](#) () const
Prepare for a set of operations that must be done atomically.
- void [end_atomic_block](#) (bool ignore_errors) const
End an atomic set of operations.
- const [TypeEntry](#) [get_type_entry_atomic](#) ([JeodMemoryTypePreDescriptor](#) &tdesc)
Return the type entry for the input type descriptor, adding the descriptor to the type table if the type has not yet been registered.

- const [TypeEntry](#) [get_type_entry_atomic](#) ([NameType](#) name_type, const std::string &type_name) const
Retrieve the descriptor for the specified type from the type table.
- bool [get_type_index_nolock](#) (const [JeodMemoryTypeDescriptor](#) &tdesc, uint32_t *idx)
Retrieve the index for the specified type from the type table, adding an entry if needed.
- const [JeodMemoryTypeDescriptor](#) * [get_type_descriptor_atomic](#) (const std::type_info &typeid_info) const
Retrieve the descriptor for the specified type from the type table.
- const [JeodMemoryTypeDescriptor](#) & [get_type_descriptor_atomic](#) (unsigned int idx) const
Retrieve the descriptor for the specified type from the type table.
- const [JeodMemoryTypeDescriptor](#) & [get_type_descriptor_nolock](#) (const [JeodMemoryItem](#) &item) const
Retrieve the descriptor for the specified type from the type table.
- const std::string & [get_string_atomic](#) (unsigned int idx) const
Retrieve the specified string from the string table.
- unsigned int [add_string_atomic](#) (const std::string &str)
Add a location identifier string to the string table.
- uint32_t [get_alloc_id_atomic](#) (const char *file, unsigned int line)
Create a unique identifier for an allocation.
- void [reset_alloc_id_atomic](#) (uint32_t unique_id)
Reset the unique identifier for a restart.
- void [find_alloc_entry_atomic](#) (const void *addr, bool delete_entry, const char *file, unsigned int line, void *&found_addr, [JeodMemoryItem](#) &found_item, const [JeodMemoryTypeDescriptor](#) *&found_type)
Find the allocation table entry that matches the input address, and delete it if delete_entry is true.
- void [add_allocation_atomic](#) (const void *addr, const [JeodMemoryItem](#) &item, const [JeodMemoryTypeDescriptor](#) &tdesc, const char *file, unsigned int line)
Add the specified addr/item pair to the table.
- void [delete_oldest_alloc_entry_atomic](#) (void *&addr, [JeodMemoryItem](#) &item, const [JeodMemoryTypeDescriptor](#) *&type)
Find and delete the alloc table entry with the smallest unique id, setting the provided references with info about the deleted item.
- void * [allocate_memory](#) (std::size_t nelems, std::size_t elem_size, bool guard, int fill) const
Allocate memory.
- void [free_memory](#) (void *addr, std::size_t length, bool guard, unsigned int alloc_idx, const char *file, unsigned int line) const
Release memory.

Static Private Member Functions

- static bool [check_master](#) (bool error_is_fatal, int line)
Many of the static methods are a pass-through to a private non-static method, with the static method testing that the pass-through is valid.

Private Attributes

- [JeodMemoryInterface](#) & [sim_interface](#)
The interface to the simulation engine's memory manager.
- [DebugLevel](#) [debug_level](#) {[Error_details](#)}
Debugging level.
- size_t [cur_data_size](#) {}
Number of allocated user bytes (excludes management overhead).
- size_t [max_data_size](#) {}
Maximum value attained by cur_data_size.

- unsigned int [max_table_size](#) {}
Maximum value attained by `alloc_table.size()`.
- unsigned int [allocation_number](#) {}
Number of allocations.
- [AllocTable](#) [alloc_table](#)
Maps memory addresses to the descriptions of those addresses.
- [TypeTable](#) [type_table](#)
Maps typeid names to type descriptors.
- [JeodMemoryReflectiveTable](#) [string_table](#)
Maps unique strings to themselves.
- pthread_mutex_t [mutex](#) {}
Mutex that synchronizes access to the tables.
- JeodSimulationInterface::Mode [mode](#) {JeodSimulationInterface::Construction}
Simulation interface mode.
- bool [guard_enabled](#) {true}
Data can be guarded if this is set.

Static Private Attributes

- static [JeodMemoryManager](#) * [Master](#) = nullptr
The singleton instance of the [JeodMemoryManager](#) class.

Friends

- class [InputProcessor](#)
- void [init_attrjeod__JeodMemoryManager](#) ()

8.8.1 Detailed Description

This class provides the interface between the macros in [jeod_alloc.hh](#) and the rest of the JEOD memory model.

The public interface is via the publicly visible static methods. All nonstatic member functions are private. Each public static method relays the method call to the singleton memory manager via a correspondingly named private member function.

Singleton

The class is intended to be a singleton. The private static member [JeodMemoryManager::Master](#) points to this singular instance. The constructor sets that static member if it is null. The constructor issues a non-fatal error when multiple instances of the class are created.

Thread Safety

This class contains objects that must be accessed and updated in a thread-safe manner. The member data that must be used atomically are

- [JeodMemoryManager::alloc_table](#) - Maps memory addresses to memory items
- [JeodMemoryManager::type_table](#) - Maps RTTI names to type descriptors
- [JeodMemoryManager::string_table](#) - Maps unique strings to themselves.
- [JeodMemoryManager::cur_data_size](#) - Current size of allocated data.

- [JeodMemoryManager::max_data_size](#) - Maximum of the above.
- [JeodMemoryManager::max_table_size](#) - Maximum allocation table size.
- [JeodMemoryManager::allocation_number](#) - Number of allocations made.

To ensure the constraint is satisfied, access to the these elements is protected by means of a mutex and is limited to a small number of methods. A pair of methods, [JeodMemoryManager::begin_atomic_block](#) and [JeodMemoryManager::end_atomic_block](#) systematize the use of the mutex. The methods that operate on the protected data are

- Constructor and destructor.
The constructor operates on the protected data before it creates the mutex and marks the [JeodMemoryManager](#) object as usable. The destructor marks the object as unusable and destroys the mutex before operating on the protected data.
- [JeodMemoryManager::generate_shutdown_report](#), which is called by the destructor after it has destroyed the mutex.
- Methods whose names end with `_atomic`. These methods use the `begin_atomic_block` / `end_atomic_block` paradigm to ensure that the operations are carried out atomically.
- Methods whose names end with `_nolock`. These methods operate on protected data but do so without atomic protection. These methods are called only by `_atomic` methods from within their atomic protection block.

Forbidden Word - Mutable

The data member [JeodMemoryManager::mutex](#) is mutable, a forbidden word per the JEOD coding standards. The coding standards allow for waivers to the standards if the exception is justified. This section provides the explanation needed to enable the use of that word in this case.

The *mutable* keyword tells the compiler to ignore modifications to mutable elements in an otherwise *const* method. The *mutex* is mutable because, although its value does change with a successful lock, it is restored to its prelock value with an unlock. A method that could otherwise qualify as a *const* method can still be a *const* method by marking the mutex as mutable. Mutexes are one of the well-accepted types of data that typically marked as mutable.

Assumptions and Constraints on the Simulation Developer

This class places restrictions on the simulation developer.

- The simulation's `MessageHandler` object must be constructed prior to constructing the simulation's [JeodMemoryManager](#) object.
- The simulation's `MessageHandler` object must not be destroyed prior to constructing the simulation's [JeodMemoryManager](#) object.
- The simulation's [JeodMemoryManager](#) object must be constructed prior to invoking any of the JEOD_↔
ALLOC_XXX macros in other models.
- The simulation's [JeodMemoryManager](#) object must not be destroyed before other models release their allocated memory.

The recommended solution is to create an instance of a compliant `SimulationInterface` before creating any other models and to destroy that `SimulationInterface` object after destroying all other models. A simple way to achieve this in a Trick-07 simulation is to define a `TrickSim` object that contains a `TrickSimulationInterface` element and to place this `sim` object immediately after the `sys sim` object.

Assumptions and Constraints on the Simulation Engine

This class makes certain assumptions of the behavior of the simulation engine.

- The simulation engine will not spawn threads that use the JEOD memory model to allocate memory until after the SimulationInterface object has been constructed.
- The simulation engine will join all threads that use the JEOD memory model prior to destroying the SimulationInterface object.

The Trick-07 and Trick-10 simulation engines satisfies these constraints.

Assumptions and Constraints on the Simulation Developer

This class places certain limitations on the architecture of a JEOD-based simulation.

- The [JeodMemoryManager](#) destructor uses the simulation's message handler to report errors discovered during destruction and may eventually use the simulation's simulation engine memory interface to revoke the registration of memory allocated by JEOD that has not been freed. This in turn means that: – The simulation's message handler and simulation engine memory interface must be destructed after destructing the memory manager. – The destructors for those objects cannot use the memory manager.
- The JEOD memory allocation and deallocation macros expand into calls to memory manager methods. The memory manager must be viable (post construction, pre destruction) for these calls to function properly. This in turn means that the memory manager must be constructed very early in the overall construction process and destructed very late in the overall destruction process.
- The supported solution to both of these issues is to use a compliant derived class of the JeodSimulation↔Interface class and to ensure that this composite object created early and destroyed late. In a Trick-07 simulation, this can be accomplished simply by placing a declaration of an object of type JeodTrickSim↔Interface near the top of an S_define file. The recommended placement is just after the Trick system sim object.

Definition at line 212 of file memory_manager.hh.

8.8.2 Member Typedef Documentation

8.8.2.1 AllocTable

```
using jeod::JeodMemoryManager::AllocTable = std::map<const void *, JeodMemoryItem> [private]
```

An AllocTable maps memory addresses to memory descriptions.

Definition at line 350 of file memory_manager.hh.

8.8.2.2 TypeTable

```
using jeod::JeodMemoryManager::TypeTable = JeodMemoryTableClonable<JeodMemoryTypeDescriptor> [private]
```

The type type itself is a memory table with copy implemented by clone().

Definition at line 355 of file memory_manager.hh.

8.8.3 Member Enumeration Documentation

8.8.3.1 DebugLevel

```
enum jeod::JeodMemoryManager::DebugLevel
```

The memory manager as a whole and individual operations have a debug level.

The debug levels and the message handler must be set to a sufficiently high level to enable and see the debugging output.

Enumerator

Debug_off	Debugging is off.
Summary_only	Summary information; Allocation data are not stored.
Error_details	Allocation data stored and used with error messages.
Full_details	Blow-by-blow accounting of all transactions.

Definition at line 226 of file memory_manager.hh.

8.8.3.2 NameType

```
enum jeod::JeodMemoryManager::NameType
```

The type lookup by type name needs to know whether the provided name is a typeid name or a demangled name.

Enumerator

Typeid_type_name	Name is from a std::type_info.name()
Demangled_type_name	Name is what people might use.

Definition at line 238 of file memory_manager.hh.

8.8.4 Constructor & Destructor Documentation

8.8.4.1 JeodMemoryManager() [1/3]

```
jeod::JeodMemoryManager::JeodMemoryManager ( ) [delete]
```

8.8.4.2 JeodMemoryManager() [2/3]

```
jeod::JeodMemoryManager::JeodMemoryManager (
    const JeodMemoryManager & ) [explicit], [delete]
```

8.8.4.3 JeodMemoryManager() [3/3]

```
jeod::JeodMemoryManager::JeodMemoryManager (
    JeodMemoryInterface & interface ) [explicit]
```

Construct a MemoryManager object.

Parameters

<i>in, out</i>	<i>interface</i>	The memory interface with the simulation engine
----------------	------------------	---

Definition at line 62 of file `memory_manager.cc`.

References `MAKE_DESCRIPTOR`, `Master`, `mutex`, and `jeod::MemoryMessages::singleton_error`.

8.8.4.4 ~JeodMemoryManager()

```
jeod::JeodMemoryManager::~JeodMemoryManager ( ) [virtual]
```

Destruct a MemoryManager object.

Assumptions and Limitations

- In a multi-threaded environment,
 - This destructor shall be called once and once only to destroy the singleton [JeodMemoryManager](#) object.
 - The thread that calls this destructor shall wait until all other threads that access JEOD memory have finished, either by default or by force.

Note that this is a constraint on the simulation engine, not on JEOD.

Definition at line 136 of file `memory_manager.cc`.

References `alloc_table`, `generate_shutdown_report()`, `jeod::JeodMemoryItem::get_is_registered()`, `get_type_↔descriptor_nolock()`, `Master`, `mutex`, and `sim_interface`.

8.8.5 Member Function Documentation

8.8.5.1 add_allocation_atomic()

```
void jeod::JeodMemoryManager::add_allocation_atomic (
    const void * addr,
    const JeodMemoryItem & item,
    const JeodMemoryTypeDescriptor & tdesc,
    const char * file,
    unsigned int line ) [private]
```

Add the specified addr/item pair to the table.

Assumptions and Limitations

- Operations on the map must be atomic. This method satisfies that requirement.
- The specified address must not already be in the table.

Parameters

in	<i>addr</i>	Newly allocated memory
in	<i>item</i>	Description of that memory
in	<i>tdesc</i>	Description of the type
in	<i>file</i>	Source file containing JEOD_ALLOC
in	<i>line</i>	Line number containing JEOD_ALLOC

Definition at line 700 of file memory_manager_protected.cc.

References `alloc_table`, `begin_atomic_block()`, `jeod::JeodMemoryTypeDescriptor::buffer_end()`, `jeod::JeodMemoryTypeDescriptor::buffer_size()`, `jeod::MemoryMessages::corrupted_memory`, `cur_data_size`, `end_atomic_block()`, `jeod::JeodMemoryItem::get_nelems()`, `get_type_descriptor_nolock()`, `max_data_size`, and `max_table_size`.

Referenced by `register_memory_internal()`.

8.8.5.2 add_string_atomic()

```
unsigned int jeod::JeodMemoryManager::add_string_atomic (
    const std::string & str ) [private]
```

Add a location identifier string to the string table.

Assumptions and Limitations

- Operations on the map must be atomic. This method satisfies that requirement.

Returns

String table index

Parameters

in	<i>str</i>	String to add
----	------------	---------------

Definition at line 188 of file `memory_manager_protected.cc`.

References `jeod::JeodMemoryReflectiveTable::add()`, `begin_atomic_block()`, `end_atomic_block()`, and `string_table`.

Referenced by `register_memory_internal()`.

8.8.5.3 allocate_memory()

```
void * jeod::JeodMemoryManager::allocate_memory (
    std::size_t nelems,
    std::size_t elem_size,
    bool guard,
    int fill ) const [private]
```

Allocate memory.

Assumptions and Limitations

- This is a low-level allocation function. It does not
 - Register the allocated memory with JEOD or with an external agent.
 - Construct the newly-allocated memory.
- The returned address should not be released using the C free function or C++ delete operator. Failure to obey this restriction will result in big problems.

Returns

Allocated memory

Parameters

in	<i>nelems</i>	Number of elements
in	<i>elem_size</i>	Size of each element
in	<i>guard</i>	Allocate guard bytes if set
in	<i>fill</i>	Fill pattern (ref. <code>memset</code>)

Definition at line 700 of file `memory_manager.cc`.

References `jeod::MAGIC0`, `jeod::MAGIC1`, and `jeod::MemoryMessages::out_of_memory`.

Referenced by `create_memory_internal()`, and `restart_reallocate()`.

8.8.5.4 begin_atomic_block()

```
void jeod::JeodMemoryManager::begin_atomic_block ( ) const [private]
```

Prepare for a set of operations that must be done atomically.

Assumptions and Limitations

- This method must be used in conjunction with end_atomic_block.

```
try {
    begin_atomic_block ();
    operate_on_protected_members();
    end_atomic_block (false);
}
catch (...) {
    end_atomic_block (true);
    throw;
}
```

- See the class header for a detailed description. Purpose: (Prepare for a set of operations that must be done atomically.) Assumptions and limitations: This method must be used in conjunction with end_↵atomic_block.

```
try {
    begin_atomic_block ();
    operate_on_protected_members();
    end_atomic_block (false);
}
catch (...) {
    end_atomic_block (true);
    throw;
}
```

(See the class header for a detailed description.)

Definition at line 97 of file memory_manager_protected.cc.

References jeod::MemoryMessages::lock_error, and mutex.

Referenced by add_allocation_atomic(), add_string_atomic(), delete_oldest_alloc_entry_atomic(), find_alloc_↵entry_atomic(), get_alloc_id_atomic(), get_string_atomic(), get_type_descriptor_atomic(), get_type_entry_atomic(), and reset_alloc_id_atomic().

8.8.5.5 check_master()

```
bool jeod::JeodMemoryManager::check_master (
    bool error_is_fatal,
    int line ) [static], [private]
```

Many of the static methods are a pass-through to a private non-static method, with the static method testing that the pass-through is valid.

This method performs that test and handles the failure response.

Returns

True if Master is not null

Parameters

in	<i>error_is_fatal</i>	True => call fail
in	<i>line</i>	LINE

Definition at line 59 of file memory_manager_static.cc.

References Master, and jeod::MemoryMessages::singleton_error.

Referenced by create_memory(), deregister_container(), destroy_memory(), get_type_descriptor(), is_allocated(), is_table_empty(), register_class(), register_container(), set_debug_level(), set_guard_enabled(), and set_mode().

8.8.5.6 create_memory()

```
void * jeod::JeodMemoryManager::create_memory (
    bool is_array,
    unsigned int nelems,
    int fill,
    const TypeEntry & tentry,
    const char * file,
    unsigned int line ) [static]
```

Allocate memory and register the allocated memory with JEOD.

Assumptions and Limitations

- This method must not be called before the singleton memory manager has been created or after it has been destroyed. A fatal error results when this is not true.
- The allocated memory is not constructed by this method. The calling routine should initialize the memory with placement new.
- Access to this method is through the JEOD memory allocation macros. Use in any other context is caveat emptor.

Returns

Allocated memory

Parameters

in	<i>is_array</i>	Memory constructed by new[] if set
in	<i>nelems</i>	Number of elements to be allocated
in	<i>fill</i>	Byte fill pattern
in	<i>tentry</i>	Type entry
in	<i>file</i>	Source file containing JEOD_ALLOC
in	<i>line</i>	Line number containing JEOD_ALLOC

Definition at line 243 of file memory_manager_static.cc.

References `check_master()`, `create_memory_internal()`, and `Master`.

8.8.5.7 `create_memory_internal()`

```
void * jeod::JeodMemoryManager::create_memory_internal (
    bool is_array,
    unsigned int nelems,
    int fill,
    const TypeEntry & tentry,
    const char * file,
    unsigned int line ) [private]
```

Allocate memory for use with placement new and register that memory with the memory manager and with the simulation engine.

Assumptions and Limitations

- This method will be invoked via the JEOD memory allocation macros. Use in any other context is a case of caveat emptor.
- The type descriptor index must index the type descriptor that describes the type to be created.
- The memory is allocated but not constructed. Construction is the responsibility of the caller. The JEOD memory allocation macros construct the allocated memory via placement new.

Returns

Allocated memory

Parameters

in	<i>is_array</i>	Memory constructed by <code>new[]</code> if set
in	<i>nelems</i>	Number of elements to be allocated
in	<i>fill</i>	Byte fill pattern
in	<i>tentry</i>	Type entry
in	<i>file</i>	Source file containing <code>JEOD_ALLOC</code>
in	<i>line</i>	Line number containing <code>JEOD_ALLOC</code>

Definition at line 387 of file `memory_manager.cc`.

References `allocate_memory()`, `jeod::JeodMemoryTypeDescriptor::get_size()`, `guard_enabled`, `register_memory_internal()`, and `jeod::JeodMemoryManager::TypeEntry::tdesc`.

Referenced by `create_memory()`.

8.8.5.8 `delete_oldest_alloc_entry_atomic()`

```
void jeod::JeodMemoryManager::delete_oldest_alloc_entry_atomic (
    void *& addr,
```

```
JeodMemoryItem & item,
const JeodMemoryTypeDescriptor *& type ) [private]
```

Find and delete the alloc table entry with the smallest unique id, setting the provided references with info about the deleted item.

The addr and type are set to NULL if the table is empty.

Assumptions and Limitations

- Operations on the map must be atomic. This method satisfies that requirement.
- If the restore doesn't work the sim will be knee deep in alligators.

Parameters

out	<i>addr</i>	Address found in table
out	<i>item</i>	Descriptor for above
out	<i>type</i>	Type descriptor

Definition at line 802 of file memory_manager_protected.cc.

References `alloc_table`, `allocation_number`, `begin_atomic_block()`, `jeod::JeodMemoryTypeDescriptor::buffer_size()`, `cur_data_size`, `end_atomic_block()`, and `get_type_descriptor_nolock()`.

Referenced by `restart_clear_memory()`.

8.8.5.9 deregister_container()

```
void jeod::JeodMemoryManager::deregister_container (
    const void * container,
    const std::type_info & container_type,
    const std::string & elem_name,
    JeodCheckpointable & checkpointable ) [static]
```

Deregister all checkpointable object contained within some object.

Assumptions and Limitations

- This method must not be called before the singleton memory manager has been created or after it has been destroyed. A fatal error results when this is not true.

Parameters

in	<i>container</i>	Object container
in	<i>container_type</i>	Container type info
in	<i>elem_name</i>	Element name
in, out	<i>checkpointable</i>	Checkpointable object

Definition at line 368 of file `memory_manager_static.cc`.

References `check_master()`, `get_type_descriptor_atomic()`, `Master`, `jeod::MemoryMessages::null_pointer`, and `sim_interface`.

8.8.5.10 `destroy_memory()`

```
void jeod::JeodMemoryManager::destroy_memory (
    void * addr,
    bool delete_array,
    const char * file,
    unsigned int line ) [static]
```

Destroy memory previously registered with JEOD.

Assumptions and Limitations

- This method must not be called before the singleton memory manager has been created or after it has been destroyed. A fatal error results when this is not true.
- The provided memory shall not be used in any way after calling this method. This method destructs and frees that memory.
- Access to this method is through the JEOD memory allocation macros. Use in any other context is caveat emptor.

Parameters

in, out	<i>addr</i>	Memory to be destroyed
in	<i>delete_array</i>	DELETE_ARRAY (true) vs. DELETE_OBJECT
in	<i>file</i>	Source file containing delete
in	<i>line</i>	Line number containing delete

Definition at line 300 of file `memory_manager_static.cc`.

References `check_master()`, `destroy_memory_internal()`, and `Master`.

8.8.5.11 `destroy_memory_internal()`

```
void jeod::JeodMemoryManager::destroy_memory_internal (
    void * addr,
    bool delete_array,
    const char * file,
    unsigned int line ) [private]
```

Destroy a chunk of memory and knowledge about it.

This includes

- De-registering the memory with JEOD and with an external agent.
- Invoking the destructor in the case of a structured type.
- Releasing the memory to the system.

Parameters

in, out	<i>addr</i>	Memory to be destroyed
in	<i>delete_array</i>	DELETE_ARRAY (true) vs. DELETE_OBJECT
in	<i>file</i>	Source file containing delete
in	<i>line</i>	Line number containing delete

Definition at line 549 of file `memory_manager.cc`.

References `jeod::JeodMemoryTypeDescriptor::buffer_size()`, `jeod::MemoryMessages::debug`, `debug_level`, `jeod::JeodMemoryTypeDescriptor::destroy_memory()`, `find_alloc_entry_atomic()`, `free_memory()`, `jeod::JeodMemoryItem::get_alloc_index()`, `jeod::JeodMemoryItem::get_is_array()`, `jeod::JeodMemoryItem::get_is_guarded()`, `jeod::JeodMemoryItem::get_is_registered()`, `jeod::JeodMemoryItem::get_nelems()`, `jeod::JeodMemoryItem::get_placement_new()`, `get_string_atomic()`, `jeod::MemoryMessages::null_pointer`, `sim_interface`, `jeod::MemoryMessages::suspect_pointer`, and `jeod::JeodMemoryTypeDescriptor::type_spec()`.

Referenced by `destroy_memory()`.

8.8.5.12 `end_atomic_block()`

```
void jeod::JeodMemoryManager::end_atomic_block (
    bool ignore_errors ) const [private]
```

End an atomic set of operations.

Parameters

in	<i>ignore_errors</i>	Ignore errors from unlock?
----	----------------------	----------------------------

Definition at line 117 of file `memory_manager_protected.cc`.

References `jeod::MemoryMessages::lock_error`, and `mutex`.

Referenced by `add_allocation_atomic()`, `add_string_atomic()`, `delete_oldest_alloc_entry_atomic()`, `find_alloc_entry_atomic()`, `get_alloc_id_atomic()`, `get_string_atomic()`, `get_type_descriptor_atomic()`, `get_type_entry_atomic()`, and `reset_alloc_id_atomic()`.

8.8.5.13 `find_alloc_entry_atomic()`

```
void jeod::JeodMemoryManager::find_alloc_entry_atomic (
    const void * addr,
    bool delete_entry,
    const char * file,
    unsigned int line,
    void *& found_addr,
    JeodMemoryItem & found_item,
    const JeodMemoryTypeDescriptor *& found_type ) [private]
```

Find the allocation table entry that matches the input address, and delete it if `delete_entry` is true.

The matching is strict. A match occurs only if the input address is a key in the allocation table. An error is reported if the input address is inside the allocated space corresponding to one of the allocation table entries.

Output values:

- Entry not found:
 - The *found_addr* and *found_type* are set to NULL.
 - The *found_item* is not touched.
- Entry found:
 - The *found_addr* is set to the key of the found entry.
 - The *found_item* is copied from the value of the found entry.
 - The *found_type* points to the *type_descriptor* entry for the found item's type.

Assumptions and Limitations

- Operations on the map must be atomic. This method satisfies that requirement.

Parameters

in	<i>addr</i>	Address
in	<i>delete_entry</i>	Indicates entry is to be deleted
in	<i>file</i>	Source file containing JEOD_XXX
in	<i>line</i>	Line number containing JEOD_XXX
out	<i>found_addr</i>	Address found in table
out	<i>found_item</i>	Descriptor for above
out	<i>found_type</i>	Type descriptor

Definition at line 587 of file `memory_manager_protected.cc`.

References `alloc_table`, `begin_atomic_block()`, `jeod::JeodMemoryTypeDescriptor::buffer_size()`, `cur_data_size`, `end_atomic_block()`, `jeod::JeodMemoryTypeDescriptor::get_name()`, `get_type_descriptor_nolock()`, and `jeod::MemoryMessages::suspect_pointer`.

Referenced by `destroy_memory_internal()`, and `is_allocated_internal()`.

8.8.5.14 free_memory()

```
void jeod::JeodMemoryManager::free_memory (
    void * addr,
    std::size_t length,
    bool guard,
    unsigned int alloc_idx,
    const char * file,
    unsigned int line ) const [private]
```

Release memory.

Assumptions and Limitations

- This is a low-level de-allocation function. It does not
 - De-register the memory with JEOD or with an external agent.
 - Destruct the memory.

Parameters

in, out	<i>addr</i>	Memory to be freed
in	<i>length</i>	Buffer size
in	<i>guard</i>	Memory was guarded if set
in	<i>alloc_idx</i>	Allocation index
in	<i>file</i>	Source file containing delete
in	<i>line</i>	Line number containing delete

Definition at line 775 of file `memory_manager.cc`.

References `jeod::MemoryMessages::corrupted_memory`, `get_string_atomic()`, `jeod::MAGIC0`, and `jeod::MAGIC1`.

Referenced by `destroy_memory_internal()`, and `restart_clear_memory()`.

8.8.5.15 generate_shutdown_report()

```
void jeod::JeodMemoryManager::generate_shutdown_report ( ) [private]
```

Generate a shutdown report.

Assumptions and Limitations

- This method is to be called by the destructor only. It freely accesses tabular data, the assumption being that the mutex and flags that protect that data are now gone.

Definition at line 180 of file `memory_manager.cc`.

References `alloc_table`, `jeod::MemoryMessages::corrupted_memory`, `jeod::MemoryMessages::debug`, `debug_↵_level`, `jeod::JeodMemoryTable< ValueType >::get()`, `jeod::JeodMemoryItem::get_alloc_index()`, `get_type_↵descriptor_nolock()`, `max_data_size`, `max_table_size`, `string_table`, and `jeod::JeodMemoryTypeDescriptor::type_↵spec()`.

Referenced by `~JeodMemoryManager()`.

8.8.5.16 get_alloc_id_atomic()

```
uint32_t jeod::JeodMemoryManager::get_alloc_id_atomic (
    const char * file,
    unsigned int line ) [private]
```

Create a unique identifier for an allocation.

Assumptions and Limitations

- Operations on the map must be atomic. This method satisfies that requirement.

Returns

Allocation ID

Parameters

in	<i>file</i>	Source file containing JEOD_ALLOC
in	<i>line</i>	Line number containing JEOD_ALLOC

Definition at line 490 of file `memory_manager_protected.cc`.

References `allocation_number`, `begin_atomic_block()`, `jeod::MemoryMessages::corrupted_memory`, and `end_atomic_block()`.

Referenced by `register_memory_internal()`.

8.8.5.17 `get_string_atomic()`

```
const std::string & jeod::JeodMemoryManager::get_string_atomic (
    unsigned int idx ) const [private]
```

Retrieve the specified string from the string table.

Assumptions and Limitations

- Operations on the map must be atomic. This method satisfies that requirement.

Returns

String table index

Parameters

in	<i>idx</i>	Class index
----	------------	-------------

Definition at line 154 of file `memory_manager_protected.cc`.

References `begin_atomic_block()`, `end_atomic_block()`, `jeod::JeodMemoryTable< ValueType >::get()`, `jeod::MemoryMessages::internal_error`, and `string_table`.

Referenced by `destroy_memory_internal()`, and `free_memory()`.

8.8.5.18 `get_type_descriptor()` [1/2]

```
const JeodMemoryTypeDescriptor * jeod::JeodMemoryManager::get_type_descriptor (
    const std::type_info & typeid_info ) [static]
```

Get a type descriptor from the memory manager's type table.

Assumptions and Limitations

- This method must not be called before the singleton memory manager has been created or after it has been destroyed. A fatal error results when this is not true.

Returns

Type descriptor

Parameters

in	<i>typeid_info</i>	C++ type descriptor
----	--------------------	---------------------

Definition at line 184 of file `memory_manager_static.cc`.

References `check_master()`, `get_type_descriptor_atomic()`, and `Master`.

Referenced by `jeod::JeodMemoryTypeDescriptor::base_type()`.

8.8.5.19 get_type_descriptor() [2/2]

```
const JeodMemoryTypeDescriptor * jeod::JeodMemoryManager::get_type_descriptor (
    JeodMemoryManager::NameType name_type,
    const std::string & type_name ) [static]
```

Get a type descriptor from the memory manager's type table.

Assumptions and Limitations

- This method must not be called before the singleton memory manager has been created or after it has been destroyed. A fatal error results when this is not true.

Returns

Type descriptor

Parameters

in	<i>name_type</i>	Typeid or demangled name
in	<i>type_name</i>	Type name

Definition at line 209 of file `memory_manager_static.cc`.

References `check_master()`, `get_type_entry_atomic()`, `Master`, and `jeod::JeodMemoryManager::TypeEntry::tdesc`.

8.8.5.20 `get_type_descriptor_atomic()` [1/2]

```
const JeodMemoryTypeDescriptor * jeod::JeodMemoryManager::get_type_descriptor_atomic (
    const std::type_info & typeid_info ) const [private]
```

Retrieve the descriptor for the specified type from the type table.

Assumptions and Limitations

- Operations on the map must be atomic. This method satisfies that requirement.

Returns

Type descriptor

Parameters

in	<i>typeid_info</i>	Type info
----	--------------------	-----------

Definition at line 333 of file `memory_manager_protected.cc`.

References `begin_atomic_block()`, `end_atomic_block()`, `jeod::JeodMemoryTable< ValueType >::find()`, `jeod::JeodMemoryTable< ValueType >::get()`, and `type_table`.

Referenced by `deregister_container()`, `get_type_descriptor()`, and `register_container()`.

8.8.5.21 `get_type_descriptor_atomic()` [2/2]

```
const JeodMemoryTypeDescriptor & jeod::JeodMemoryManager::get_type_descriptor_atomic (
    unsigned int idx ) const [private]
```

Retrieve the descriptor for the specified type from the type table.

Assumptions and Limitations

- The input index is non-zero. This assumption is enforced.
- Operations on the map must be atomic. This method satisfies that requirement.

Returns

Type descriptor

Parameters

in	<i>idx</i>	Type index
----	------------	------------

Definition at line 442 of file `memory_manager_protected.cc`.

References `begin_atomic_block()`, `end_atomic_block()`, `jeod::JeodMemoryTable< ValueType >::get()`, `jeod::MemoryMessages::internal_error`, and `type_table`.

8.8.5.22 `get_type_descriptor_nolock()`

```
const JeodMemoryTypeDescriptor & jeod::JeodMemoryManager::get_type_descriptor_nolock (
    const JeodMemoryItem & item ) const [inline], [private]
```

Retrieve the descriptor for the specified type from the type table.

Assumptions and Limitations

- The type is in the table. A core dump will result if it is not.
- Operations on the type table must be atomic. This method *does not* satisfy that requirement.

Returns

Type descriptor

Parameters

in	<i>item</i>	Memory descriptor
----	-------------	-------------------

Definition at line 561 of file `memory_manager.hh`.

Referenced by `add_allocation_atomic()`, `delete_oldest_alloc_entry_atomic()`, `find_alloc_entry_atomic()`, `generate_shutdown_report()`, and `~JeodMemoryManager()`.

8.8.5.23 `get_type_entry_atomic()` [1/2]

```
const JeodMemoryManager::TypeEntry jeod::JeodMemoryManager::get_type_entry_atomic (
    JeodMemoryTypePreDescriptor & tdesc ) [private]
```

Return the type entry for the input type descriptor, adding the descriptor to the type table if the type has not yet been registered.

Assumptions and Limitations

- The mangled name returned by the `std::type_info` name method is unique across all allocatable types and is invariant.
- Operations on the map must be atomic. This method satisfies that requirement.

Returns

Type descriptor index

Parameters

in	<i>tdesc</i>	Type pre-descriptor
----	--------------	---------------------

Definition at line 273 of file `memory_manager_protected.cc`.

References `jeod::JeodMemoryTable< ValueType >::add()`, `begin_atomic_block()`, `jeod::MemoryMessages::debug`, `debug_level`, `end_atomic_block()`, `jeod::JeodMemoryTable< ValueType >::find()`, `jeod::JeodMemoryTable< ValueType >::get()`, `jeod::JeodMemoryTypePreDescriptor::get_descriptor()`, `jeod::JeodMemoryTypeDescriptor↵::get_name()`, `jeod::JeodMemoryTypePreDescriptor::get_typeid()`, and `type_table`.

Referenced by `get_type_descriptor()`, `register_class()`, and `restart_reallocate()`.

8.8.5.24 get_type_entry_atomic() [2/2]

```
const JeodMemoryManager::TypeEntry jeod::JeodMemoryManager::get_type_entry_atomic (
    JeodMemoryManager::NameType name_type,
    const std::string & type_name ) const [private]
```

Retrieve the descriptor for the specified type from the type table.

Assumptions and Limitations

- Operations on the map must be atomic. This method satisfies that requirement.

Returns

Type entry

Parameters

in	<i>name_type</i>	Name type spec
in	<i>type_name</i>	Type name

Definition at line 373 of file `memory_manager_protected.cc`.

References `jeod::JeodMemoryTable< ValueType >::begin()`, `begin_atomic_block()`, `jeod::JeodMemoryTable< ValueType >::end()`, `end_atomic_block()`, `jeod::JeodMemoryTable< ValueType >::find()`, `jeod::JeodMemory↵Table< ValueType >::get()`, `jeod::JeodMemoryTypeDescriptor::get_name()`, `type_table`, and `Typeid_type_name`.

8.8.5.25 get_type_index_nolock()

```
bool jeod::JeodMemoryManager::get_type_index_nolock (
    const JeodMemoryTypeDescriptor & tdesc,
    uint32_t * idx ) [private]
```

Retrieve the index for the specified type from the type table, adding an entry if needed.

Assumptions and Limitations

- Operations on the type table must be atomic. This method *does not* satisfy that requirement.

Returns

True => table updated

Parameters

in	<i>tdesc</i>	Descriptor
out	<i>idx</i>	Type descriptor index

Definition at line 235 of file `memory_manager_protected.cc`.

References `jeod::JeodMemoryTable< ValueType >::add()`, `jeod::JeodMemoryTable< ValueType >::find()`, `jeod::JeodMemoryTypeDescriptor::get_typeid()`, and `type_table`.

8.8.5.26 is_allocated()

```
bool jeod::JeodMemoryManager::is_allocated (
    const void * addr,
    const char * file,
    unsigned int line ) [static]
```

Query whether some address was allocated by JEOD.

Assumptions and Limitations

- This method must not be called before the singleton memory manager has been created or after it has been destroyed. A fatal error results when this is not true.

Returns

True if allocated by JEOD

Parameters

in	<i>addr</i>	Memory to be queried
in	<i>file</i>	Source file containing query
in	<i>line</i>	Line number containing query

Definition at line 270 of file `memory_manager_static.cc`.

References `check_master()`, `is_allocated_internal()`, and `Master`.

8.8.5.27 is_allocated_internal()

```
bool jeod::JeodMemoryManager::is_allocated_internal (
    const void * addr,
    const char * file,
    unsigned int line ) [private]
```

Query whether some address was allocated by JEOD.

Returns

True if the address in question was allocated by JEOD

Parameters

in	<i>addr</i>	Memory to be queried
in	<i>file</i>	Source file containing query
in	<i>line</i>	Line number containing query

Definition at line 519 of file `memory_manager.cc`.

References `find_alloc_entry_atomic()`.

Referenced by `is_allocated()`.

8.8.5.28 is_table_empty()

```
bool jeod::JeodMemoryManager::is_table_empty ( ) [static]
```

Query whether all allocated memory has been freed.

Assumptions and Limitations

- Intended for testing use only. This method does not use a thread-safe query.

Returns

Has all memory been freed?

Definition at line 132 of file `memory_manager_static.cc`.

References `alloc_table`, `check_master()`, and `Master`.

8.8.5.29 operator=()

```
JeodMemoryManager& jeod::JeodMemoryManager::operator= (
    const JeodMemoryManager & ) [delete]
```

8.8.5.30 register_class()

```
const JeodMemoryManager::TypeEntry jeod::JeodMemoryManager::register_class (
    JeodMemoryTypePreDescriptor & tdesc ) [static]
```

Register a class with the memory manager.

Assumptions and Limitations

- This method must not be called before the singleton memory manager has been created or after it has been destroyed. A fatal error results when this is not true.
- Access to this method is through the JEOD memory allocation macros. Use in any other context is caveat emptor.

Returns

Type entry for the class

Parameters

in	<i>tdesc</i>	Type pre-descriptor
----	--------------	---------------------

Definition at line 158 of file `memory_manager_static.cc`.

References `check_master()`, `get_type_entry_atomic()`, and `Master`.

8.8.5.31 register_container()

```
void jeod::JeodMemoryManager::register_container (
    const void * container,
    const std::type_info & container_type,
    const std::string & elem_name,
    JeodCheckpointable & checkpointable ) [static]
```

Register a checkpointable object with the memory manager.

Assumptions and Limitations

- This method must not be called before the singleton memory manager has been created or after it has been destroyed. A fatal error results when this is not true.

Parameters

in	<i>container</i>	Object container
in	<i>container_type</i>	Container type info
in	<i>elem_name</i>	Element name
in, out	<i>checkpointable</i>	Checkpointable object

Definition at line 322 of file `memory_manager_static.cc`.

References `check_master()`, `get_type_descriptor_atomic()`, `Master`, `jeod::MemoryMessages::null_pointer`, and `sim_interface`.

8.8.5.32 `register_memory_internal()`

```
void jeod::JeodMemoryManager::register_memory_internal (
    const void * addr,
    uint32_t unique_id,
    bool placement_new,
    bool is_array,
    unsigned int nelems,
    const TypeEntry & tentry,
    const char * file,
    unsigned int line ) [private]
```

Allocate memory if that was not already done by the caller and register the memory with JEOD and with an external agent.

Assumptions and Limitations

- This method will be invoked via the JEOD memory allocation macros. Use in any other context is caveat emptor.
- The corresponding delete macro will be used to delete the memory. Using the C free function or the C++ delete operator can cause *big* problems.
- The delete macro will be expanded with the same placement new option as was used in the allocation macro that resulted in this call.
- The memory is not constructed. That is the job of the expansion of the JEOD_ALLOC macro.

Parameters

in	<i>addr</i>	Memory to be registered
in	<i>unique_id</i>	Unique id
in	<i>placement_new</i>	Was memory allocated by this model?
in	<i>is_array</i>	Was memory allocated as an array?
in	<i>nelems</i>	Array size
in	<i>tentry</i>	Type entry
in	<i>file</i>	Source file containing JEOD_ALLOC
in	<i>line</i>	Line number containing JEOD_ALLOC

Definition at line 421 of file `memory_manager.cc`.

References `add_allocation_atomic()`, `add_string_atomic()`, `jeod::JeodMemoryTypeDescriptor::buffer_size()`, `jeod::MemoryMessages::debug`, `debug_level`, `get_alloc_id_atomic()`, `jeod::JeodMemoryTypeDescriptor::get_register_instances()`, `jeod::JeodMemoryManager::TypeEntry::index`, `jeod::MemoryMessages::invalid_size`, `jeod::JeodMemoryTypeDescriptor::is_structured()`, `reset_alloc_id_atomic()`, `jeod::JeodMemoryItem::set_unique_id()`, `sim_interface`, `jeod::JeodMemoryManager::TypeEntry::tdesc`, and `jeod::JeodMemoryTypeDescriptor::type_spec()`.

Referenced by `create_memory_internal()`, and `restart_reallocate()`.

8.8.5.33 reset_alloc_id_atomic()

```
void jeod::JeodMemoryManager::reset_alloc_id_atomic (
    uint32_t unique_id ) [private]
```

Reset the unique identifier for a restart.

Assumptions and Limitations

- Operations on the map must be atomic. This method satisfies that requirement.

Parameters

in	<i>unique_id</i>	Unique id of a restored allocation
----	------------------	------------------------------------

Definition at line 537 of file `memory_manager_protected.cc`.

References `allocation_number`, `begin_atomic_block()`, and `end_atomic_block()`.

Referenced by `register_memory_internal()`.

8.8.5.34 restart_clear_memory()

```
void jeod::JeodMemoryManager::restart_clear_memory ( )
```

Wipe out all allocated memory in anticipation of restoring the memory in some previously recording checkpoint file.

Assumptions and Limitations

- If the restore doesn't work the sim will be knee deep in alligators.

Definition at line 272 of file `memory_manager.cc`.

References `allocation_number`, `jeod::JeodMemoryTypeDescriptor::buffer_size()`, `cur_data_size`, `delete_oldest_alloc_entry_atomic()`, `jeod::JeodMemoryTypeDescriptor::destroy_memory()`, `free_memory()`, `jeod::JeodMemoryItem::get_alloc_index()`, `jeod::JeodMemoryItem::get_is_array()`, `jeod::JeodMemoryItem::get_is_guarded()`, `jeod::JeodMemoryItem::get_is_registered()`, `jeod::JeodMemoryItem::get_nelems()`, `jeod::JeodMemoryItem::get_placement_new()`, `max_data_size`, `max_table_size`, and `sim_interface`.

8.8.5.35 restart_reallocate()

```
void jeod::JeodMemoryManager::restart_reallocate (
    const std::string & mangled_type_name,
    uint32_t unique_id,
    uint32_t nelements,
    bool is_array )
```

Restore one chunk of allocated memory per a checkpoint file entry.

Assumptions and Limitations

- This restores the allocation, but not the contents. The contents will soon be restored by the simulation engine.

Parameters

in	<i>mangled_type_name</i>	Mangled type name
in	<i>unique_id</i>	Unique id
in	<i>nelements</i>	Number of elements
in	<i>is_array</i>	True => an array

Definition at line 320 of file `memory_manager.cc`.

References `allocate_memory()`, `jeod::JeodMemoryTypeDescriptor::construct_array()`, `jeod::JeodMemoryTypeDescriptor::get_size()`, `get_type_entry_atomic()`, `guard_enabled`, `register_memory_internal()`, `jeod::MemoryMessages::suspect_pointer`, `jeod::JeodMemoryManager::TypeEntry::tdesc`, and `Typeid_type_name`.

8.8.5.36 set_debug_level() [1/2]

```
void jeod::JeodMemoryManager::set_debug_level (
    unsigned int level ) [static]
```

Set the debug level.

Parameters

in	<i>level</i>	New debug level
----	--------------	-----------------

Definition at line 98 of file `memory_manager_static.cc`.

References `Full_details`.

8.8.5.37 set_debug_level() [2/2]

```
void jeod::JeodMemoryManager::set_debug_level (
    DebugLevel level ) [static]
```

Set the debug level.

Parameters

in	<i>level</i>	New debug level
----	--------------	-----------------

Definition at line 84 of file `memory_manager_static.cc`.

References `check_master()`, `debug_level`, and `Master`.

8.8.5.38 set_guard_enabled()

```
void jeod::JeodMemoryManager::set_guard_enabled (
    bool value ) [static]
```

Set the guard_enabled flag.

Parameters

in	<i>value</i>	New value
----	--------------	-----------

Definition at line 114 of file memory_manager_static.cc.

References check_master(), guard_enabled, and Master.

8.8.5.39 set_mode()

```
void jeod::JeodMemoryManager::set_mode (
    JeodSimulationInterface::Mode new_mode ) [static]
```

Set the memory manager's simulation interface mode.

Assumptions and Limitations

- This method must not be called before the singleton memory manager has been created or after it has been destroyed. A fatal error results when this is not true.

Parameters

in	<i>new_mode</i>	New mode
----	-----------------	----------

Definition at line 411 of file memory_manager_static.cc.

References check_master(), Master, and set_mode_internal().

8.8.5.40 set_mode_internal()

```
void jeod::JeodMemoryManager::set_mode_internal (
    JeodSimulationInterface::Mode new_mode ) [private]
```

Set the mode and perform mode transitions.

Parameters

in	<i>new_mode</i>	New mode
----	-----------------	----------

Definition at line 665 of file memory_manager.cc.

References mode.

Referenced by set_mode().

8.8.6 Friends And Related Function Documentation

8.8.6.1 init_attrjeod__JeodMemoryManager

```
void init_attrjeod__JeodMemoryManager ( ) [friend]
```

8.8.6.2 InputProcessor

```
friend class InputProcessor [friend]
```

Definition at line 214 of file memory_manager.hh.

8.8.7 Field Documentation

8.8.7.1 alloc_table

```
AllocTable jeod::JeodMemoryManager::alloc_table [private]
```

Maps memory addresses to the descriptions of those addresses.

trick_io(**)

Definition at line 522 of file memory_manager.hh.

Referenced by add_allocation_atomic(), delete_oldest_alloc_entry_atomic(), find_alloc_entry_atomic(), generate_shutdown_report(), is_table_empty(), and ~JeodMemoryManager().

8.8.7.2 allocation_number

```
unsigned int jeod::JeodMemoryManager::allocation_number {} [private]
```

Number of allocations.

This always increments and can be adjusted upward on restarts.`trick_io(*o) trick_units(-)`

Definition at line 513 of file `memory_manager.hh`.

Referenced by `delete_oldest_alloc_entry_atomic()`, `get_alloc_id_atomic()`, `reset_alloc_id_atomic()`, and `restart_clear_memory()`.

8.8.7.3 cur_data_size

```
size_t jeod::JeodMemoryManager::cur_data_size {} [private]
```

Number of allocated user bytes (excludes management overhead).

`trick_io(*o) trick_units(-)`

Definition at line 497 of file `memory_manager.hh`.

Referenced by `add_allocation_atomic()`, `delete_oldest_alloc_entry_atomic()`, `find_alloc_entry_atomic()`, and `restart_clear_memory()`.

8.8.7.4 debug_level

```
DebugLevel jeod::JeodMemoryManager::debug_level {Error_details} [private]
```

Debugging level.

- 0 = Minimal output, errors only.
- 1 = Summary report, generated just before `exit(0)`.
- 2 = Report unfreed memory as well.
- 3 = Blow-by-blow report of each allocation and deallocation.`trick_units(-)`

Definition at line 492 of file `memory_manager.hh`.

Referenced by `destroy_memory_internal()`, `generate_shutdown_report()`, `get_type_entry_atomic()`, `register_memory_internal()`, and `set_debug_level()`.

8.8.7.5 guard_enabled

```
bool jeod::JeodMemoryManager::guard_enabled {true} [private]
```

Data can be guarded if this is set.

If not set, guards will never be established.`trick_units(-)`

Definition at line 548 of file `memory_manager.hh`.

Referenced by `create_memory_internal()`, `restart_reallocate()`, and `set_guard_enabled()`.

8.8.7.6 Master

```
JeodMemoryManager * jeod::JeodMemoryManager::Master = nullptr [static], [private]
```

The singleton instance of the `JeodMemoryManager` class.

The constructor sets this pointer.`trick_io(*o) trick_units(-)`

Definition at line 368 of file `memory_manager.hh`.

Referenced by `check_master()`, `create_memory()`, `deregister_container()`, `destroy_memory()`, `get_type_descriptor()`, `is_allocated()`, `is_table_empty()`, `JeodMemoryManager()`, `register_class()`, `register_container()`, `set_debug_level()`, `set_guard_enabled()`, `set_mode()`, and `~JeodMemoryManager()`.

8.8.7.7 max_data_size

```
size_t jeod::JeodMemoryManager::max_data_size {} [private]
```

Maximum value attained by `cur_data_size`.

`trick_io(*o) trick_units(-)`

Definition at line 502 of file `memory_manager.hh`.

Referenced by `add_allocation_atomic()`, `generate_shutdown_report()`, and `restart_clear_memory()`.

8.8.7.8 max_table_size

```
unsigned int jeod::JeodMemoryManager::max_table_size {} [private]
```

Maximum value attained by `alloc_table.size()`.

`trick_io(*o) trick_units(-)`

Definition at line 507 of file `memory_manager.hh`.

Referenced by `add_allocation_atomic()`, `generate_shutdown_report()`, and `restart_clear_memory()`.

8.8.7.9 mode

```
JeodSimulationInterface::Mode jeod::JeodMemoryManager::mode {JeodSimulationInterface::Construction}
[private]
```

Simulation interface mode.

trick_units(—)

Definition at line 542 of file memory_manager.hh.

Referenced by set_mode_internal().

8.8.7.10 mutex

```
pthread_mutex_t jeod::JeodMemoryManager::mutex {} [mutable], [private]
```

Mutex that synchronizes access to the tables.

trick_io(**)

Definition at line 537 of file memory_manager.hh.

Referenced by begin_atomic_block(), end_atomic_block(), JeodMemoryManager(), and ~JeodMemoryManager().

8.8.7.11 sim_interface

```
JeodMemoryInterface& jeod::JeodMemoryManager::sim_interface [private]
```

The interface to the simulation engine's memory manager.

trick_io(*o) trick_units(—)

Definition at line 483 of file memory_manager.hh.

Referenced by deregister_container(), destroy_memory_internal(), register_container(), register_memory_internal(), restart_clear_memory(), and ~JeodMemoryManager().

8.8.7.12 string_table

```
JeodMemoryReflectiveTable jeod::JeodMemoryManager::string_table [private]
```

Maps unique strings to themselves.

trick_io(**)

Definition at line 532 of file memory_manager.hh.

Referenced by add_string_atomic(), generate_shutdown_report(), and get_string_atomic().

8.8.7.13 type_table

`TypeTable` `jeod::JeodMemoryManager::type_table` [private]

Maps typeid names to type descriptors.

`trick_io(**)`

Definition at line 527 of file `memory_manager.hh`.

Referenced by `get_type_descriptor_atomic()`, `get_type_entry_atomic()`, and `get_type_index_nolock()`.

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

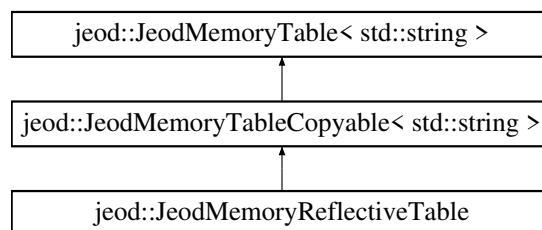
- [memory_manager.hh](#)
- [memory_manager.cc](#)
- [memory_manager_protected.cc](#)
- [memory_manager_static.cc](#)

8.9 jeod::JeodMemoryReflectiveTable Class Reference

A [JeodMemoryReflectiveTable](#) maps strings to themselves.

```
#include <memory_table.hh>
```

Inheritance diagram for `jeod::JeodMemoryReflectiveTable`:



Public Member Functions

- [JeodMemoryReflectiveTable](#) ()
Default constructor.
- [JeodMemoryReflectiveTable](#) (const [JeodMemoryReflectiveTable](#) &)=delete
- [JeodMemoryReflectiveTable](#) & operator= (const [JeodMemoryReflectiveTable](#) &)=delete
- unsigned int [add](#) (const std::string &keyval)
Add a key to the table.

Private Member Functions

- unsigned int [add](#) (const std::string &key, const std::string &val)
Not implemented.

Additional Inherited Members

8.9.1 Detailed Description

A [JeodMemoryReflectiveTable](#) maps strings to themselves.

Definition at line 413 of file `memory_table.hh`.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 JeodMemoryReflectiveTable() [1/2]

```
jeod::JeodMemoryReflectiveTable::JeodMemoryReflectiveTable ( ) [inline]
```

Default constructor.

Definition at line 420 of file `memory_table.hh`.

8.9.2.2 JeodMemoryReflectiveTable() [2/2]

```
jeod::JeodMemoryReflectiveTable::JeodMemoryReflectiveTable (
    const JeodMemoryReflectiveTable & ) [explicit], [delete]
```

8.9.3 Member Function Documentation

8.9.3.1 add() [1/2]

```
unsigned int jeod::JeodMemoryReflectiveTable::add (
    const std::string & key,
    const std::string & val ) [private]
```

Not implemented.

Referenced by `jeod::JeodMemoryManager::add_string_atomic()`.

8.9.3.2 add() [2/2]

```
unsigned int jeod::JeodMemoryReflectiveTable::add (
    const std::string & keyval ) [inline]
```

Add a key to the table.

A reflective table has values equal to keys.

Returns

Index number mapped by the key.

Parameters

in	keyval	Key (and value) to be added to the table.
----	--------	---

Definition at line 442 of file `memory_table.hh`.

References `jeod::JeodMemoryTable< ValueType >::add()`.

8.9.3.3 operator=()

```
JeodMemoryReflectiveTable& jeod::JeodMemoryReflectiveTable::operator= (
    const JeodMemoryReflectiveTable & ) [delete]
```

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

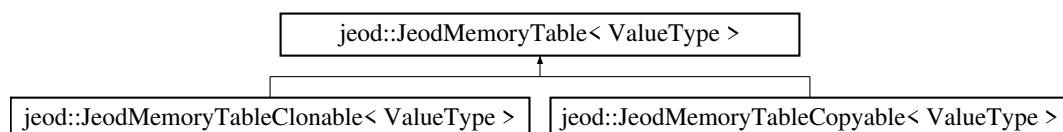
- [memory_table.hh](#)

8.10 jeod::JeodMemoryTable< ValueType > Class Template Reference

A [JeodMemoryTable](#) maps strings to values with a coordinated map/vector pair.

```
#include <memory_table.hh>
```

Inheritance diagram for `jeod::JeodMemoryTable< ValueType >`:

**Public Types**

- using [NameIndex](#) = `std::map< const std::string, unsigned int >`
Maps strings to an index number.
- using [ValueList](#) = `std::vector< const ValueType * >`
Maps index numbers to key values.
- using [const_value_iterator](#) = `typename ValueList::const_iterator`
Const iterator over values.

Public Member Functions

- [JeodMemoryTable](#) ()
Default constructor.
- virtual [~JeodMemoryTable](#) ()
Destructor.
- [JeodMemoryTable](#) (const [JeodMemoryTable](#) &)=delete
- [JeodMemoryTable](#) & operator= (const [JeodMemoryTable](#) &)=delete
- unsigned int [find](#) (const std::string &key) const
Find the index number at which key/value pair is stored in the table.
- [const_value_iterator](#) [begin](#) () const
Returns a const iterator that points to the first element of the list.
- [const_value_iterator](#) [end](#) () const
Returns a const iterator that points past the last element of the list.
- unsigned int [add](#) (const std::string &key, const ValueType &val)
Add a key/value pair to the table.
- void [del](#) (const std::string &key)
Delete the key and associated data from the table.
- const ValueType * [get](#) (unsigned int idx) const
Retrieve the value for the specified index from the list.

Protected Member Functions

- virtual const ValueType * [clone](#) (const ValueType &value) const =0
(Somehow) clone the input value.

Private Attributes

- [NameIndex](#) [string_to_index](#)
Maps keys to indices in the value_list.
- [ValueList](#) [value_list](#)
Vector of values.

8.10.1 Detailed Description

```
template<typename ValueType>
class jeod::JeodMemoryTable< ValueType >
```

A [JeodMemoryTable](#) maps strings to values with a coordinated map/vector pair.

Template Parameters

<i>ValueType</i>	The underlying type of the values maintained in the table. The stored values are pointers to this underlying type.
------------------	--

A [JeodMemoryTable](#) contains two data members: a std::map and a std::vector. The map data member maps keys to integers. The integer mapped by a key is the index into the vector where the value associated with the key is stored.

So why not just use a map? The reason is that storing an integer requires less memory than storing a string or a pointer to a string, particularly on 64 bit machines. In the application at hand, keeping track of memory allocations, the number of data types is relatively small compared to the to the number of allocated chunks of data. The extra overhead of maintaining a map and a vector is small compared to the savings that results from storing thousands of integers rather than pointers or strings.

Principal Operations

- [add\(\)](#)
Returns the integer value associated with a key in the table's map. In the case of a new key/value pair, a new key/vector size entry is added to the map and the value is added to the end of the vector. Note well: The value is ignored when the key is already in the map.
- [del\(\)](#)
Deletes the key from the table's map and deletes the cloned value at the corresponding index. The vector itself is modified (truncated) only in the special case of deleting the last-added entry. This ensures that stored indices will remain valid.
- [get\(\)](#)
Returns the value in the table's vector at the specified index.

Assumptions and Limitations

- The value is ignored for duplicate key entries. The underlying assumption is that all of the values for those duplicate entries are somehow equal to one another.
- As-is, the table is not thread-safe. Calls to [add\(\)](#) and [get\(\)](#) made in a multi-threaded environment should be protected by a mutex. This protection is the responsibility of the (programmatic) users.
- JEOD reserves index 0 for internal use. Valid indices are positive.
- The [del\(\)](#) method should be used only if the (programmatic) user *knows* that no other references to the to-be-deleted entry exist.

Definition at line 123 of file `memory_table.hh`.

8.10.2 Member Typedef Documentation

8.10.2.1 `const_value_iterator`

```
template<typename ValueType>
using jeod::JeodMemoryTable< ValueType >::const_value_iterator = typename ValueList::const_↔
iterator
```

Const iterator over values.

Definition at line 140 of file `memory_table.hh`.

8.10.2.2 NameIndex

```
template<typename ValueType>
using jeod::JeodMemoryTable< ValueType >::NameIndex = std::map<const std::string, unsigned
int>
```

Maps strings to an index number.

Definition at line 130 of file memory_table.hh.

8.10.2.3 ValueList

```
template<typename ValueType>
using jeod::JeodMemoryTable< ValueType >::ValueList = std::vector<const ValueType *>
```

Maps index numbers to key values.

Definition at line 135 of file memory_table.hh.

8.10.3 Constructor & Destructor Documentation

8.10.3.1 JeodMemoryTable() [1/2]

```
template<typename ValueType>
jeod::JeodMemoryTable< ValueType >::JeodMemoryTable ( ) [inline]
```

Default constructor.

Note that JEOD reserves table index 0 as meaning nothing.

Definition at line 148 of file memory_table.hh.

8.10.3.2 ~JeodMemoryTable()

```
template<typename ValueType>
virtual jeod::JeodMemoryTable< ValueType >::~~JeodMemoryTable ( ) [inline], [virtual]
```

Destructor.

The contents of the vector are clones created by [add\(\)](#) and hence must be deleted to avoid a leak.

Definition at line 161 of file memory_table.hh.

8.10.3.3 JeodMemoryTable() [2/2]

```
template<typename ValueType>
jeod::JeodMemoryTable< ValueType >::JeodMemoryTable (
    const JeodMemoryTable< ValueType > & ) [delete]
```

8.10.4 Member Function Documentation

8.10.4.1 add()

```
template<typename ValueType>
unsigned int jeod::JeodMemoryTable< ValueType >::add (
    const std::string & key,
    const ValueType & val ) [inline]
```

Add a key/value pair to the table.

Returns

Index number mapped by the key

Parameters

in	<i>key</i>	Key
in	<i>val</i>	Value

Definition at line 231 of file memory_table.hh.

Referenced by jeod::JeodMemoryReflectiveTable::add(), jeod::JeodMemoryManager::get_type_entry_atomic(), and jeod::JeodMemoryManager::get_type_index_nolock().

8.10.4.2 begin()

```
template<typename ValueType>
const_value_iterator jeod::JeodMemoryTable< ValueType >::begin ( ) const [inline]
```

Returns a const iterator that points to the first element of the list.

Definition at line 212 of file memory_table.hh.

Referenced by jeod::JeodMemoryManager::get_type_entry_atomic().

8.10.4.3 clone()

```
template<typename ValueType>
virtual const ValueType* jeod::JeodMemoryTable< ValueType >::clone (
    const ValueType & value ) const [protected], [pure virtual]
```

(Somehow) clone the input value.

Returns

Clone of input value.

Parameters

in	<i>value</i>	Value to be cloned.
----	--------------	---------------------

Implemented in [jeod::JeodMemoryTableCopyable< ValueType >](#), [jeod::JeodMemoryTableCopyable< std::string >](#), [jeod::JeodMemoryTableClonable< ValueType >](#), and [jeod::JeodMemoryTableClonable< JeodMemoryTypeDescriptor >](#).

Referenced by [jeod::JeodMemoryTable< JeodMemoryTypeDescriptor >::add\(\)](#).

8.10.4.4 del()

```
template<typename ValueType>
void jeod::JeodMemoryTable< ValueType >::del (
    const std::string & key ) [inline]
```

Delete the key and associated data from the table.

Use with care.

Parameters

in	<i>key</i>	Key
----	------------	-----

Exceptions

<i>std::invalid_argument</i>	on attempting to delete an element that is not in the table.
------------------------------	--

Definition at line 262 of file `memory_table.hh`.

8.10.4.5 end()

```
template<typename ValueType>
const_value_iterator jeod::JeodMemoryTable< ValueType >::end ( ) const [inline]
```

Returns a const iterator that points past the last element of the list.

Definition at line 220 of file `memory_table.hh`.

Referenced by `jeod::JeodMemoryManager::get_type_entry_atomic()`.

8.10.4.6 `find()`

```
template<typename ValueType>
unsigned int jeod::JeodMemoryTable< ValueType >::find (
    const std::string & key ) const [inline]
```

Find the index number at which key/value pair is stored in the table.

Returns

Index number mapped by the key

Parameters

<code>in</code>	<code>key</code>	Key
-----------------	------------------	-----

Definition at line 186 of file `memory_table.hh`.

Referenced by `jeod::JeodMemoryManager::get_type_descriptor_atomic()`, `jeod::JeodMemoryManager::get_type_entry_atomic()`, and `jeod::JeodMemoryManager::get_type_index_nolock()`.

8.10.4.7 `get()`

```
template<typename ValueType>
const ValueType* jeod::JeodMemoryTable< ValueType >::get (
    unsigned int idx ) const [inline]
```

Retrieve the value for the specified index from the list.

Returns

Value for specified index.

Parameters

<code>in</code>	<code>idx</code>	Table index whose value is to be retrieved.
-----------------	------------------	---

Exceptions

<code>std::out_of_range</code>	for an index of zero or for an index beyond the range of the vector.
--------------------------------	--

Exceptions

<code>std::invalid_argument</code>	when the index is in range but the value is null. This only happens when the item in question has previously been deleted.
------------------------------------	--

Definition at line 301 of file `memory_table.hh`.

Referenced by `jeod::JeodMemoryManager::generate_shutdown_report()`, `jeod::JeodMemoryManager::get_string_atomic()`, `jeod::JeodMemoryManager::get_type_descriptor_atomic()`, and `jeod::JeodMemoryManager::get_type_entry_atomic()`.

8.10.4.8 operator=()

```
template<typename ValueType>
JeodMemoryTable& jeod::JeodMemoryTable< ValueType >::operator= (
    const JeodMemoryTable< ValueType > & ) [delete]
```

8.10.5 Field Documentation

8.10.5.1 string_to_index

```
template<typename ValueType>
NameIndex jeod::JeodMemoryTable< ValueType >::string_to_index [private]
```

Maps keys to indices in the `value_list`.

`trick_io(**)`

Definition at line 334 of file `memory_table.hh`.

Referenced by `jeod::JeodMemoryTable< JeodMemoryTypeDescriptor >::add()`, `jeod::JeodMemoryTable< JeodMemoryTypeDescriptor >::del()`, `jeod::JeodMemoryTable< JeodMemoryTypeDescriptor >::find()`, and `jeod::JeodMemoryTable< JeodMemoryTypeDescriptor >::~~JeodMemoryTable()`.

8.10.5.2 value_list

```
template<typename ValueType>
ValueList jeod::JeodMemoryTable< ValueType >::value_list [private]
```

Vector of values.

`trick_io(**)`

Definition at line 339 of file `memory_table.hh`.

Referenced by `jeod::JeodMemoryTable< JeodMemoryTypeDescriptor >::add()`, `jeod::JeodMemoryTable< JeodMemoryTypeDescriptor >::begin()`, `jeod::JeodMemoryTable< JeodMemoryTypeDescriptor >::del()`, `jeod::JeodMemoryTable< JeodMemoryTypeDescriptor >::end()`, `jeod::JeodMemoryTable< JeodMemoryTypeDescriptor >::get()`, `jeod::JeodMemoryTable< JeodMemoryTypeDescriptor >::JeodMemoryTable()`, and `jeod::JeodMemoryTable< JeodMemoryTypeDescriptor >::~~JeodMemoryTable()`.

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

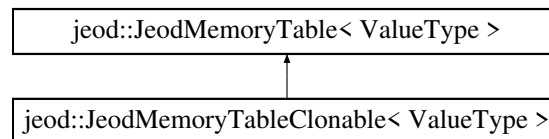
- [memory_table.hh](#)

8.11 jeod::JeodMemoryTableClonable< ValueType > Class Template Reference

A [JeodMemoryTableClonable](#) is a [JeodMemoryTable](#) that implements the required [clone\(\)](#) functionality by invoking the *ValueType*'s [clone\(\)](#) method to create a clone of the input value.

```
#include <memory_table.hh>
```

Inheritance diagram for jeod::JeodMemoryTableClonable< ValueType >:



Public Member Functions

- [JeodMemoryTableClonable](#) ()
Default constructor.
- [JeodMemoryTableClonable](#) (const [JeodMemoryTableClonable](#) &)=delete
- [JeodMemoryTableClonable](#) & operator= (const [JeodMemoryTableClonable](#) &)=delete

Protected Member Functions

- const ValueType * [clone](#) (const ValueType &value) const override
Creates a copy of the input value by invoking its clone method.

Additional Inherited Members

8.11.1 Detailed Description

```
template<typename ValueType>
class jeod::JeodMemoryTableClonable< ValueType >
```

A [JeodMemoryTableClonable](#) is a [JeodMemoryTable](#) that implements the required [clone\(\)](#) functionality by invoking the *ValueType*'s [clone\(\)](#) method to create a clone of the input value.

Definition at line 347 of file `memory_table.hh`.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 JeodMemoryTableClonable() [1/2]

```
template<typename ValueType>
jeod::JeodMemoryTableClonable< ValueType >::JeodMemoryTableClonable ( ) [inline]
```

Default constructor.

Definition at line 354 of file memory_table.hh.

8.11.2.2 JeodMemoryTableClonable() [2/2]

```
template<typename ValueType>
jeod::JeodMemoryTableClonable< ValueType >::JeodMemoryTableClonable (
    const JeodMemoryTableClonable< ValueType > & ) [delete]
```

8.11.3 Member Function Documentation

8.11.3.1 clone()

```
template<typename ValueType>
const ValueType* jeod::JeodMemoryTableClonable< ValueType >::clone (
    const ValueType & value ) const [inline], [override], [protected], [virtual]
```

Creates a copy of the input value by invoking its clone method.

Returns

Duplicate of input value.

Parameters

in	<i>value</i>	Value to be cloned.
----	--------------	---------------------

Implements [jeod::JeodMemoryTable< ValueType >](#).

Definition at line 370 of file memory_table.hh.

8.11.3.2 operator=()

```
template<typename ValueType>
JeodMemoryTableClonable& jeod::JeodMemoryTableClonable< ValueType >::operator= (
    const JeodMemoryTableClonable< ValueType > & ) [delete]
```

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

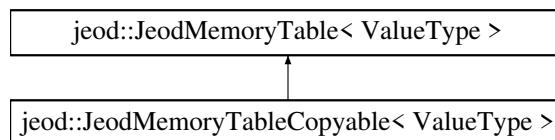
- [memory_table.hh](#)

8.12 jeod::JeodMemoryTableCopyable< ValueType > Class Template Reference

A [JeodMemoryTableCopyable](#) is a [JeodMemoryTable](#) that implements the required [clone\(\)](#) functionality by invoking the *ValueType*'s copy constructor to create a clone of the input value.

```
#include <memory_table.hh>
```

Inheritance diagram for jeod::JeodMemoryTableCopyable< ValueType >:



Public Member Functions

- [JeodMemoryTableCopyable](#) ()
Default constructor.
- [JeodMemoryTableCopyable](#) (const [JeodMemoryTableCopyable](#) &)=delete
- [JeodMemoryTableCopyable](#) & operator= (const [JeodMemoryTableCopyable](#) &)=delete

Protected Member Functions

- const ValueType * [clone](#) (const ValueType &value) const override
Creates a copy of the input value by invoking its copy constructor.

Additional Inherited Members

8.12.1 Detailed Description

```
template<typename ValueType>
class jeod::JeodMemoryTableCopyable< ValueType >
```

A [JeodMemoryTableCopyable](#) is a [JeodMemoryTable](#) that implements the required [clone\(\)](#) functionality by invoking the *ValueType*'s copy constructor to create a clone of the input value.

Definition at line 381 of file [memory_table.hh](#).

8.12.2 Constructor & Destructor Documentation

8.12.2.1 JeodMemoryTableCopyable() [1/2]

```
template<typename ValueType>
jeod::JeodMemoryTableCopyable< ValueType >::JeodMemoryTableCopyable ( ) [inline]
```

Default constructor.

Definition at line 388 of file memory_table.hh.

8.12.2.2 JeodMemoryTableCopyable() [2/2]

```
template<typename ValueType>
jeod::JeodMemoryTableCopyable< ValueType >::JeodMemoryTableCopyable (
    const JeodMemoryTableCopyable< ValueType > & ) [delete]
```

8.12.3 Member Function Documentation

8.12.3.1 clone()

```
template<typename ValueType>
const ValueType* jeod::JeodMemoryTableCopyable< ValueType >::clone (
    const ValueType & value ) const [inline], [override], [protected], [virtual]
```

Creates a copy of the input value by invoking its copy constructor.

Returns

Duplicate of input value.

Parameters

in	<i>value</i>	Value to be cloned.
----	--------------	---------------------

Implements [jeod::JeodMemoryTable< ValueType >](#).

Definition at line 404 of file memory_table.hh.

8.12.3.2 operator=()

```
template<typename ValueType>
JeodMemoryTableCopyable& jeod::JeodMemoryTableCopyable< ValueType >::operator= (
    const JeodMemoryTableCopyable< ValueType > & ) [delete]
```

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

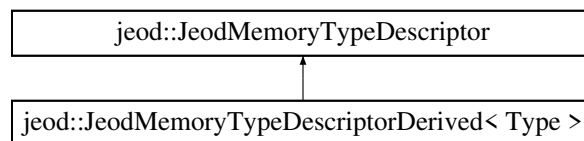
- [memory_table.hh](#)

8.13 jeod::JeodMemoryTypeDescriptor Class Reference

Abstract class for managing data allocated as some specific type.

```
#include <memory_type.hh>
```

Inheritance diagram for jeod::JeodMemoryTypeDescriptor:



Data Structures

- struct [attr](#)

The simulation engine attributes that describe the type. `trick_io(**)`

Public Member Functions

- [JeodMemoryTypeDescriptor](#) (const std::type_info &obj_typeid, const struct ATTRIBUTES_tag &type_attr, std::size_t type_size, bool is_exportable=true)
Non-default constructor.
- virtual [~JeodMemoryTypeDescriptor](#) ()=default
- [JeodMemoryTypeDescriptor](#) (const [JeodMemoryTypeDescriptor](#) &)=default
- [JeodMemoryTypeDescriptor](#) & operator= (const [JeodMemoryTypeDescriptor](#) &)=delete
- const std::type_info & [get_typeid](#) () const
Get the type info for the type.
- const std::string & [get_name](#) () const
Get the name of the type.
- std::size_t [get_size](#) () const
Get the size of the type.
- const struct ATTRIBUTES_tag & [get_attr](#) () const
Get the simulation engine attributes for the type.
- bool [get_register_instances](#) () const
Get the simulation engine attributes for the type.
- std::size_t [dimensionality](#) () const
Determine the dimensionality of the type.
- std::size_t [buffer_size](#) (unsigned int nelems) const
Compute the size of a buffer.
- std::size_t [buffer_size](#) (const [JeodMemoryItem](#) &item) const
Compute the size of a buffer.
- const void * [buffer_end](#) (const void *addr, unsigned int nelems) const
Compute the address of the byte just past the end a buffer.
- const void * [buffer_end](#) (const void *addr, const [JeodMemoryItem](#) &item) const

- *Compute the address of the byte just past the end a buffer.*
const std::string [type_spec](#) (const [JeodMemoryItem](#) &item) const
- *Construct a type specification string.*
void [destroy_memory](#) (bool placement_new, bool is_array, unsigned int nelem, void *addr) const
- *Destroy memory.*
virtual [JeodMemoryTypeDescriptor](#) * [clone](#) () const =0
- *Create a copy of the descriptor.*
virtual bool [is_structured](#) () const =0
- *Indicate whether the type associated with the descriptor is a structured (non-primitive, non-pointer) type.*
virtual void * [construct_array](#) (std::size_t nelem, void *addr) const =0
- *Construct an array of objects of the type.*
virtual const void * [most_derived_pointer](#) (const void *addr) const =0
- *Find the most-derived object corresponding to the input pointer.*
virtual void * [most_derived_pointer](#) (void *addr) const =0
- *Find the most-derived object corresponding to the input pointer.*

Static Public Member Functions

- static void [set_check_for_registration_errors](#) (bool val)
Enable/disable registration error messages.

Protected Member Functions

- virtual void [delete_array](#) (void *addr) const =0
Delete an array of instances of the type associated with the descriptor.
- virtual void [delete_object](#) (void *addr) const =0
Delete a single instance of the type associated with the descriptor.
- virtual void [destruct_array](#) (std::size_t nelem, void *addr) const =0
Destruct (but do not delete) an array of nelem instances of the type associated with the descriptor.

Static Protected Member Functions

- static std::string [initialize_type_name](#) (const std::string &type_name)
The [jeod_alloc.h](#) macros insert a space between the type name and the asterisks.
- static std::size_t [pointer_dimension](#) (const std::string &demangled_name)
Get the pointer dimensionality of the type.
- static const [JeodMemoryTypeDescriptor](#) * [base_type](#) (const std::string &demangled_name)
Get the descriptor for the base (non-pointer) of some pointer type.

Protected Attributes

- const std::type_info & [obj_id](#)
The RTTI descriptor of the type.
- const std::string [name](#)
The name of the type in code.
- const std::size_t [size](#) {}
The size of an instance of the type.
- bool [register_instances](#) {}
Should instances be registered with the simulation engine? If true (default value), instances of the type will be registered with the simulation engine; the simulation engine is responsible for checkpointing and restoring the contents of such instances.

Static Protected Attributes

- static bool `check_for_registration_errors` = false
When set, suspect memory interface results will be reported as a warnings.

8.13.1 Detailed Description

Abstract class for managing data allocated as some specific type.

A `JeodMemoryTypeDescriptor` is a clonable object that contains the name and size of a specific data type. Instantiable subclasses of this class are created by the class templates that derive from this base class.

Definition at line 97 of file `memory_type.hh`.

8.13.2 Constructor & Destructor Documentation

8.13.2.1 `JeodMemoryTypeDescriptor()` [1/2]

```
jeod::JeodMemoryTypeDescriptor::JeodMemoryTypeDescriptor (
    const std::type_info & obj_typeid,
    const struct ATTRIBUTES_tag & type_attr,
    std::size_t type_size,
    bool is_exportable = true )
```

Non-default constructor.

Note that construction is via a `char*` as that is what the C preprocessor creates when it stringifies a token.

Parameters

in	<code>obj_typeid</code>	Type ID for type
in	<code>type_attr</code>	Type attributes
in	<code>type_size</code>	Type size
in	<code>is_exportable</code>	Register instances?

Definition at line 173 of file `memory_type.cc`.

8.13.2.2 `~JeodMemoryTypeDescriptor()`

```
virtual jeod::JeodMemoryTypeDescriptor::~~JeodMemoryTypeDescriptor ( ) [virtual], [default]
```


8.13.2.3 JeodMemoryTypeDescriptor() [2/2]

```
jeod::JeodMemoryTypeDescriptor::JeodMemoryTypeDescriptor (
    const JeodMemoryTypeDescriptor & ) [default]
```

8.13.3 Member Function Documentation

8.13.3.1 base_type()

```
const JeodMemoryTypeDescriptor * jeod::JeodMemoryTypeDescriptor::base_type (
    const std::string & demangled_name ) [static], [protected]
```

Get the descriptor for the base (non-pointer) of some pointer type.

Note

Assumes GNU c++ name mangling, where 'const' is always preceded by a space.

Definition at line 98 of file memory_type.cc.

References `jeod::JeodMemoryManager::Demangled_type_name`, and `jeod::JeodMemoryManager::get_type_descriptor()`.

8.13.3.2 buffer_end() [1/2]

```
const void* jeod::JeodMemoryTypeDescriptor::buffer_end (
    const void * addr,
    unsigned int nelems ) const [inline]
```

Compute the address of the byte just past the end a buffer.

Parameters

in	<i>addr</i>	Start of buffer
in	<i>nelems</i>	Size of the array

Definition at line 217 of file memory_type.hh.

References `buffer_size()`.

Referenced by `jeod::JeodMemoryManager::add_allocation_atomic()`, and `buffer_end()`.

8.13.3.3 `buffer_end()` [2/2]

```
const void* jeod::JeodMemoryTypeDescriptor::buffer_end (
    const void * addr,
    const JeodMemoryItem & item ) const [inline]
```

Compute the address of the byte just past the end a buffer.

Parameters

in	<i>addr</i>	Start of buffer
in	<i>item</i>	Buffer descriptor

Definition at line 227 of file `memory_type.hh`.

References `buffer_end()`, and `jeod::JeodMemoryItem::get_nelems()`.

8.13.3.4 `buffer_size()` [1/2]

```
std::size_t jeod::JeodMemoryTypeDescriptor::buffer_size (
    unsigned int nelems ) const [inline]
```

Compute the size of a buffer.

Parameters

in	<i>nelems</i>	Size of the array
----	---------------	-------------------

Returns

: Buffer size

Definition at line 197 of file `memory_type.hh`.

References `size`.

Referenced by `jeod::JeodMemoryManager::add_allocation_atomic()`, `buffer_end()`, `buffer_size()`, `jeod::JeodMemoryManager::delete_oldest_alloc_entry_atomic()`, `jeod::JeodMemoryManager::destroy_memory_internal()`, `jeod::JeodMemoryManager::find_alloc_entry_atomic()`, `jeod::JeodMemoryManager::register_memory_internal()`, and `jeod::JeodMemoryManager::restart_clear_memory()`.

8.13.3.5 `buffer_size()` [2/2]

```
std::size_t jeod::JeodMemoryTypeDescriptor::buffer_size (
    const JeodMemoryItem & item ) const [inline]
```

Compute the size of a buffer.

Parameters

in	item	Buffer descriptor
----	------	-------------------

Returns

: Buffer size

Definition at line 207 of file `memory_type.hh`.

References `buffer_size()`, and `jeod::JeodMemoryItem::get_nelems()`.

8.13.3.6 clone()

```
virtual JeodMemoryTypeDescriptor* jeod::JeodMemoryTypeDescriptor::clone ( ) const [pure virtual]
```

Create a copy of the descriptor.

Returns

Copy.

Implemented in `jeod::JeodMemoryTypeDescriptorDerived< Type >`.

Referenced by `jeod::JeodMemoryTableClonable< JeodMemoryTypeDescriptor >::clone()`.

8.13.3.7 construct_array()

```
virtual void* jeod::JeodMemoryTypeDescriptor::construct_array (
    std::size_t nelem,
    void * addr ) const [pure virtual]
```

Construct an array of objects of the type.

The default implementation does nothing, which is the right thing to do for primitive types, pointers, and abstract classes.

Implemented in `jeod::JeodMemoryTypeDescriptorDerived< Type >`.

Referenced by `jeod::JeodMemoryManager::restart_reallocate()`.

8.13.3.8 delete_array()

```
virtual void jeod::JeodMemoryTypeDescriptor::delete_array (
    void * addr ) const [protected], [pure virtual]
```

Delete an array of instances of the type associated with the descriptor.

In other words, `delete[] addr`.

Parameters

<i>in, out</i>	<i>addr</i>	Address to be deleted
----------------	-------------	-----------------------

Implemented in [jeod::JeodMemoryTypeDescriptorDerived< Type >](#).

Referenced by `destroy_memory()`.

8.13.3.9 delete_object()

```
virtual void jeod::JeodMemoryTypeDescriptor::delete_object (
    void * addr ) const [protected], [pure virtual]
```

Delete a single instance of the type associated with the descriptor.

In other words, delete *addr*.

Parameters

<i>in, out</i>	<i>addr</i>	Address to be deleted
----------------	-------------	-----------------------

Implemented in [jeod::JeodMemoryTypeDescriptorDerived< Type >](#).

Referenced by `destroy_memory()`.

8.13.3.10 destroy_memory()

```
void jeod::JeodMemoryTypeDescriptor::destroy_memory (
    bool placement_new,
    bool is_array,
    unsigned int nelem,
    void * addr ) const [inline]
```

Destroy memory.

Parameters

<i>in</i>	<i>placement_new</i>	Constructed with placement new?
<i>in</i>	<i>is_array</i>	Allocated as an array?
<i>in</i>	<i>nelem</i>	Number of elements
<i>in, out</i>	<i>addr</i>	Address to destroy

Definition at line 242 of file `memory_type.hh`.

References `delete_array()`, `delete_object()`, and `destruct_array()`.

Referenced by `jeod::JeodMemoryManager::destroy_memory_internal()`, and `jeod::JeodMemoryManager::restart↵_clear_memory()`.

8.13.3.11 destruct_array()

```
virtual void jeod::JeodMemoryTypeDescriptor::destruct_array (
    std::size_t nelem,
    void * addr ) const    [protected], [pure virtual]
```

Destruct (but do not delete) an array of *nelem* instances of the type associated with the descriptor.

Parameters

in	<i>nelem</i>	Number of elements in <i>addr</i>
in, out	<i>addr</i>	Address to be destructed

Implemented in [jeod::JeodMemoryTypeDescriptorDerived< Type >](#).

Referenced by `destroy_memory()`.

8.13.3.12 dimensionality()

```
std::size_t jeod::JeodMemoryTypeDescriptor::dimensionality ( ) const    [inline]
```

Determine the dimensionality of the type.

Returns

: Type dimensionality

Definition at line 187 of file `memory_type.hh`.

References `name`, and `pointer_dimension()`.

8.13.3.13 get_attr()

```
const struct ATTRIBUTES_tag& jeod::JeodMemoryTypeDescriptor::get_attr ( ) const    [inline]
```

Get the simulation engine attributes for the type.

Returns

Type attributes

Definition at line 167 of file `memory_type.hh`.

8.13.3.14 get_name()

```
const std::string& jeod::JeodMemoryTypeDescriptor::get_name ( ) const [inline]
```

Get the name of the type.

Returns

Type name

Definition at line 149 of file memory_type.hh.

References name.

Referenced by jeod::JeodMemoryManager::find_alloc_entry_atomic(), and jeod::JeodMemoryManager::get_type↵_entry_atomic().

8.13.3.15 get_register_instances()

```
bool jeod::JeodMemoryTypeDescriptor::get_register_instances ( ) const [inline]
```

Get the simulation engine attributes for the type.

Returns

Type attributes

Definition at line 176 of file memory_type.hh.

References register_instances.

Referenced by jeod::JeodMemoryManager::register_memory_internal().

8.13.3.16 get_size()

```
std::size_t jeod::JeodMemoryTypeDescriptor::get_size ( ) const [inline]
```

Get the size of the type.

Returns

Type size

Definition at line 158 of file memory_type.hh.

References size.

Referenced by jeod::JeodMemoryManager::create_memory_internal(), and jeod::JeodMemoryManager::restart_↵reallocate().

8.13.3.17 get_typeid()

```
const std::type_info& jeod::JeodMemoryTypeDescriptor::get_typeid ( ) const [inline]
```

Get the type info for the type.

Returns

Type info

Definition at line 140 of file memory_type.hh.

References `obj_id`.

Referenced by `jeod::JeodMemoryManager::get_type_index_nolock()`.

8.13.3.18 initialize_type_name()

```
std::string jeod::JeodMemoryTypeDescriptor::initialize_type_name (
    const std::string & type_name ) [static], [protected]
```

The [jeod_alloc.hh](#) macros insert a space between the type name and the asterisks.

Delete that space.

Returns

Name, as c++ string

Parameters

in	<i>type_name</i>	Name, as C string
----	------------------	-------------------

Definition at line 57 of file memory_type.cc.

8.13.3.19 is_structured()

```
virtual bool jeod::JeodMemoryTypeDescriptor::is_structured ( ) const [pure virtual]
```

Indicate whether the type associated with the descriptor is a structured (non-primitive, non-pointer) type.

Implemented in [jeod::JeodMemoryTypeDescriptorDerived< Type >](#).

Referenced by `jeod::JeodMemoryManager::register_memory_internal()`.

8.13.3.20 most_derived_pointer() [1/2]

```
virtual const void* jeod::JeodMemoryTypeDescriptor::most_derived_pointer (
    const void * addr ) const [pure virtual]
```

Find the most-derived object corresponding to the input pointer.

Implemented in [jeod::JeodMemoryTypeDescriptorDerived< Type >](#).

8.13.3.21 most_derived_pointer() [2/2]

```
virtual void* jeod::JeodMemoryTypeDescriptor::most_derived_pointer (
    void * addr ) const [pure virtual]
```

Find the most-derived object corresponding to the input pointer.

Implemented in [jeod::JeodMemoryTypeDescriptorDerived< Type >](#).

8.13.3.22 operator=()

```
JeodMemoryTypeDescriptor& jeod::JeodMemoryTypeDescriptor::operator= (
    const JeodMemoryTypeDescriptor & ) [delete]
```

8.13.3.23 pointer_dimension()

```
size_t jeod::JeodMemoryTypeDescriptor::pointer_dimension (
    const std::string & demangled_name ) [static], [protected]
```

Get the pointer dimensionality of the type.

Definition at line 74 of file `memory_type.cc`.

Referenced by `dimensionality()`.

8.13.3.24 set_check_for_registration_errors()

```
static void jeod::JeodMemoryTypeDescriptor::set_check_for_registration_errors (
    bool val ) [inline], [static]
```

Enable/disable registration error messages.

Parameters

in	val	New value for check_for_registration_errors
----	-----	---

Definition at line 106 of file memory_type.hh.

References check_for_registration_errors.

8.13.3.25 type_spec()

```
const std::string jeod::JeodMemoryTypeDescriptor::type_spec (
    const JeodMemoryItem & item ) const
```

Construct a type specification string.

Returns

Type string

Parameters

in	item	Item descriptor
----	------	-----------------

Definition at line 190 of file memory_type.cc.

References jeod::JeodMemoryItem::get_is_array(), jeod::JeodMemoryItem::get_nelems(), and obj_id.

Referenced by jeod::JeodMemoryManager::destroy_memory_internal(), jeod::JeodMemoryManager::generate_shutdown_report(), and jeod::JeodMemoryManager::register_memory_internal().

8.13.4 Field Documentation

8.13.4.1 check_for_registration_errors

```
bool jeod::JeodMemoryTypeDescriptor::check_for_registration_errors = false [static], [protected]
```

When set, suspect memory interface results will be reported as a warnings.

No messages are issued when this flag is clear.trick_units(-)

Definition at line 330 of file memory_type.hh.

Referenced by set_check_for_registration_errors().

8.13.4.2 name

```
const std::string jeod::JeodMemoryTypeDescriptor::name [protected]
```

The name of the type in code.

trick_io(**)

Definition at line 342 of file memory_type.hh.

Referenced by dimensionality(), and get_name().

8.13.4.3 obj_id

```
const std::type_info& jeod::JeodMemoryTypeDescriptor::obj_id [protected]
```

The RTTI descriptor of the type.

trick_io(**)

Definition at line 337 of file memory_type.hh.

Referenced by get_typeid(), and type_spec().

8.13.4.4 register_instances

```
bool jeod::JeodMemoryTypeDescriptor::register_instances {} [protected]
```

Should instances be registered with the simulation engine? If true (default value), instances of the type will be registered with the simulation engine; the simulation engine is responsible for checkpointing and restoring the contents of such instances.

If false, instances will not be registered with the simulation engine; the simulation engine is not responsible for checkpointing/restarting such instances.trick_io(**)

Definition at line 365 of file memory_type.hh.

Referenced by get_register_instances().

8.13.4.5 size

```
const std::size_t jeod::JeodMemoryTypeDescriptor::size {} [protected]
```

The size of an instance of the type.

trick_io(**)

Definition at line 353 of file memory_type.hh.

Referenced by buffer_size(), and get_size().

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

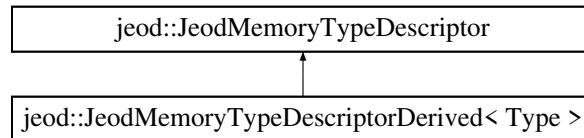
- [memory_type.hh](#)
- [memory_type.cc](#)

8.14 jeod::JeodMemoryTypeDescriptorDerived< Type > Class Template Reference

Extends [JeodMemoryTypeDescriptor](#) to describe a specific type.

```
#include <memory_type.hh>
```

Inheritance diagram for jeod::JeodMemoryTypeDescriptorDerived< Type >:



Public Types

- using [TypeDescriptor](#) = [JeodMemoryTypeDescriptorDerived](#)< Type >
This class.
- using [Attributes](#) = [JeodSimEngineAttributes](#)< Type, std::is_class< Type >::value >
Attributes for the Type.

Public Member Functions

- [JeodMemoryTypeDescriptorDerived](#) (bool is_exportable=true)
Default constructor.
- [JeodMemoryTypeDescriptorDerived](#) (const [JeodMemoryTypeDescriptorDerived](#) &src)
Copy constructor; pass-through to the parent class equivalent.
- [~JeodMemoryTypeDescriptorDerived](#) () override=default
- [JeodMemoryTypeDescriptorDerived](#) & operator= (const [JeodMemoryTypeDescriptorDerived](#) &)=delete
- [JeodMemoryTypeDescriptor](#) * [clone](#) () const override
Create a copy of the descriptor.
- bool [is_structured](#) () const override
Indicate whether the type associated with the descriptor is a structured (non-primitive, non-pointer) type.
- void * [construct_array](#) (std::size_t nelem, void *addr) const override
Construct an array of objects of the type.
- const void * [most_derived_pointer](#) (const void *addr) const override
Find the most-derived object corresponding to the input pointer.
- void * [most_derived_pointer](#) (void *addr) const override
Find the most-derived object corresponding to the input pointer.

Protected Member Functions

- void [delete_array](#) (void *addr) const override
Delete an array of instances of type Type.
- void [delete_object](#) (void *addr) const override
Delete a single instance of type Type.
- void [destruct_array](#) (std::size_t nelem, void *addr) const override
Destroy an array of nelem instances of type Type.

Additional Inherited Members

8.14.1 Detailed Description

```
template<typename Type>
class jeod::JeodMemoryTypeDescriptorDerived< Type >
```

Extends [JeodMemoryTypeDescriptor](#) to describe a specific type.

tparam Type The type to be described.

Definition at line 372 of file memory_type.hh.

8.14.2 Member Typedef Documentation

8.14.2.1 Attributes

```
template<typename Type >
using jeod::JeodMemoryTypeDescriptorDerived< Type >::Attributes = JeodSimEngineAttributes<Type,
std::is_class<Type>::value>
```

Attributes for the Type.

Definition at line 385 of file memory_type.hh.

8.14.2.2 TypeDescriptor

```
template<typename Type >
using jeod::JeodMemoryTypeDescriptorDerived< Type >::TypeDescriptor = JeodMemoryTypeDescriptorDerived<Type>
```

This class.

Definition at line 380 of file memory_type.hh.

8.14.3 Constructor & Destructor Documentation

8.14.3.1 JeodMemoryTypeDescriptorDerived() [1/2]

```
template<typename Type >
jeod::JeodMemoryTypeDescriptorDerived< Type >::JeodMemoryTypeDescriptorDerived (
    bool is_exportable = true ) [inline]
```

Default constructor.

Invoke the parent class non-default constructor with type, attributes, and size information.

Definition at line 394 of file memory_type.hh.

Referenced by jeod::JeodMemoryTypeDescriptorDerived< Type >::clone().

8.14.3.2 JeodMemoryTypeDescriptorDerived() [2/2]

```
template<typename Type >
jeod::JeodMemoryTypeDescriptorDerived< Type >::JeodMemoryTypeDescriptorDerived (
    const JeodMemoryTypeDescriptorDerived< Type > & src ) [inline]
```

Copy constructor; pass-through to the parent class equivalent.

Parameters

in	src	Item to be copied
----	-----	-------------------

Definition at line 403 of file memory_type.hh.

8.14.3.3 ~JeodMemoryTypeDescriptorDerived()

```
template<typename Type >
jeod::JeodMemoryTypeDescriptorDerived< Type >::~~JeodMemoryTypeDescriptorDerived ( ) [override],
[default]
```

8.14.4 Member Function Documentation

8.14.4.1 clone()

```
template<typename Type >
JeodMemoryTypeDescriptor* jeod::JeodMemoryTypeDescriptorDerived< Type >::clone ( ) const [inline],
[override], [virtual]
```

Create a copy of the descriptor.

Returns

Copy.

Implements [jeod::JeodMemoryTypeDescriptor](#).

Definition at line 418 of file `memory_type.hh`.

References `jeod::JeodMemoryTypeDescriptorDerived< Type >::JeodMemoryTypeDescriptorDerived()`.

8.14.4.2 `construct_array()`

```
template<typename Type >
void* jeod::JeodMemoryTypeDescriptorDerived< Type >::construct_array (
    std::size_t nelem,
    void * addr ) const [inline], [override], [virtual]
```

Construct an array of objects of the type.

Implements [jeod::JeodMemoryTypeDescriptor](#).

Definition at line 436 of file `memory_type.hh`.

8.14.4.3 `delete_array()`

```
template<typename Type >
void jeod::JeodMemoryTypeDescriptorDerived< Type >::delete_array (
    void * addr ) const [inline], [override], [protected], [virtual]
```

Delete an array of instances of type *Type*.

In other words, `delete[] addr`.

Parameters

<i>in, out</i>	<i>addr</i>	Address to be deleted
----------------	-------------	-----------------------

Implements [jeod::JeodMemoryTypeDescriptor](#).

Definition at line 468 of file `memory_type.hh`.

8.14.4.4 `delete_object()`

```
template<typename Type >
void jeod::JeodMemoryTypeDescriptorDerived< Type >::delete_object (
    void * addr ) const [inline], [override], [protected], [virtual]
```

Delete a single instance of type *Type*.

In other words, delete *addr*.

Parameters

<i>in, out</i>	<i>addr</i>	Address to be deleted
----------------	-------------	-----------------------

Implements [jeod::JeodMemoryTypeDescriptor](#).

Definition at line 479 of file `memory_type.hh`.

8.14.4.5 destruct_array()

```
template<typename Type >
void jeod::JeodMemoryTypeDescriptorDerived< Type >::destruct_array (
    std::size_t nelem,
    void * addr ) const [inline], [override], [protected], [virtual]
```

Destroy an array of *nelem* instances of type *Type*.

Implements [jeod::JeodMemoryTypeDescriptor](#).

Definition at line 488 of file `memory_type.hh`.

8.14.4.6 is_structured()

```
template<typename Type >
bool jeod::JeodMemoryTypeDescriptorDerived< Type >::is_structured ( ) const [inline], [override],
[virtual]
```

Indicate whether the type associated with the descriptor is a structured (non-primitive, non-pointer) type.

Implements [jeod::JeodMemoryTypeDescriptor](#).

Definition at line 428 of file `memory_type.hh`.

8.14.4.7 most_derived_pointer() [1/2]

```
template<typename Type >
const void* jeod::JeodMemoryTypeDescriptorDerived< Type >::most_derived_pointer (
    const void * addr ) const [inline], [override], [virtual]
```

Find the most-derived object corresponding to the input pointer.

Parameters

in	<i>addr</i>	Pointer to be examined
----	-------------	------------------------

Returns

Pointer to most-derived object.

Implements [jeod::JeodMemoryTypeDescriptor](#).

Definition at line 446 of file `memory_type.hh`.

8.14.4.8 most_derived_pointer() [2/2]

```
template<typename Type >
void* jeod::JeodMemoryTypeDescriptorDerived< Type >::most_derived_pointer (
    void * addr ) const [inline], [override], [virtual]
```

Find the most-derived object corresponding to the input pointer.

Parameters

in	<i>addr</i>	Pointer to be examined
----	-------------	------------------------

Returns

Pointer to most-derived object.

Implements [jeod::JeodMemoryTypeDescriptor](#).

Definition at line 456 of file `memory_type.hh`.

References `jeod::jeod_alloc_get_allocated_pointer()`.

8.14.4.9 operator=()

```
template<typename Type >
JeodMemoryTypeDescriptorDerived& jeod::JeodMemoryTypeDescriptorDerived< Type >::operator= (
    const JeodMemoryTypeDescriptorDerived< Type > & ) [delete]
```

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

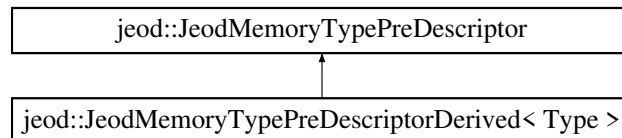
- [memory_type.hh](#)

8.15 jeod::JeodMemoryTypePreDescriptor Class Reference

Abstract class for describing a type without necessarily needing to create a [JeodMemoryTypeDescriptor](#) of that type.

```
#include <memory_type.hh>
```

Inheritance diagram for jeod::JeodMemoryTypePreDescriptor:



Public Member Functions

- virtual [~JeodMemoryTypePreDescriptor](#) ()=default
- virtual const std::type_info & [get_typeid](#) () const =0
Get the type info for the type.
- virtual const [JeodMemoryTypeDescriptor](#) & [get_descriptor](#) ()=0
Get a type descriptor for the type.

8.15.1 Detailed Description

Abstract class for describing a type without necessarily needing to create a [JeodMemoryTypeDescriptor](#) of that type.

The intent is to avoid creating a type descriptor for a type if the type is already represented in the type table.

Usage of a [JeodMemoryTypePreDescriptor](#) is highly constrained. There are two simple rules:

- Never cache a pointer or reference to a [JeodMemoryTypeDescriptor](#) in long-term memory.
- Never cache a pointer or reference to a [JeodMemoryTypeDescriptor](#) obtained by calling the [JeodMemoryTypeDescriptor](#)'s [get_descriptor](#) method.

Definition at line 507 of file `memory_type.hh`.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 ~JeodMemoryTypePreDescriptor()

```
virtual jeod::JeodMemoryTypePreDescriptor::~~JeodMemoryTypePreDescriptor ( ) [virtual], [default]
```

8.15.3 Member Function Documentation

8.15.3.1 `get_descriptor()`

```
virtual const JeodMemoryTypeDescriptor& jeod::JeodMemoryTypePreDescriptor::get_descriptor ( )
[pure virtual]
```

Get a type descriptor for the type.

The returned value should not be cached in a permanent store. The reference has a lifespan limited to that of the [JeodMemoryTypePreDescriptor](#) object.

Returns

Type descriptor.

Implemented in [jeod::JeodMemoryTypePreDescriptorDerived< Type >](#).

Referenced by [jeod::JeodMemoryManager::get_type_entry_atomic\(\)](#).

8.15.3.2 `get_typeid()`

```
virtual const std::type_info& jeod::JeodMemoryTypePreDescriptor::get_typeid ( ) const [pure
virtual]
```

Get the type info for the type.

Returns

Type info

Implemented in [jeod::JeodMemoryTypePreDescriptorDerived< Type >](#).

Referenced by [jeod::JeodMemoryManager::get_type_entry_atomic\(\)](#).

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

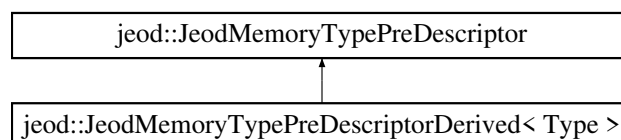
- [memory_type.hh](#)

8.16 `jeod::JeodMemoryTypePreDescriptorDerived< Type >` Class Template Reference

A [JeodMemoryTypePreDescriptorDerived](#) describes a *Type*.

```
#include <memory_type.hh>
```

Inheritance diagram for `jeod::JeodMemoryTypePreDescriptorDerived< Type >`:



Public Types

- using [TypeDescriptor](#) = [JeodMemoryTypeDescriptorDerived](#)< Type >
The type descriptor this class describes.

Public Member Functions

- [JeodMemoryTypePreDescriptorDerived](#) (bool exportable=true)
Default constructor.
- [JeodMemoryTypePreDescriptorDerived](#) (const [JeodMemoryTypePreDescriptorDerived](#) &src)
Copy constructor.
- [~JeodMemoryTypePreDescriptorDerived](#) () override
Destructor.
- [JeodMemoryTypePreDescriptor](#) & [get_ref](#) ()
Get a reference to this object.
- const std::type_info & [get_typeid](#) () const override
Get the type info for the type.
- const [JeodMemoryTypeDescriptor](#) & [get_descriptor](#) () override
Get a type descriptor for the type.

Private Attributes

- [TypeDescriptor](#) * [descriptor](#) {}
- bool [is_exportable](#) {true}

8.16.1 Detailed Description

```
template<typename Type>
class jeod::JeodMemoryTypePreDescriptorDerived< Type >
```

A [JeodMemoryTypePreDescriptorDerived](#) describes a *Type*.

Definition at line 531 of file `memory_type.hh`.

8.16.2 Member Typedef Documentation

8.16.2.1 TypeDescriptor

```
template<typename Type >
using jeod::JeodMemoryTypePreDescriptorDerived< Type >::TypeDescriptor = JeodMemoryTypeDescriptorDerived<Type>
```

The type descriptor this class describes.

Definition at line 539 of file `memory_type.hh`.

8.16.3 Constructor & Destructor Documentation

8.16.3.1 JeodMemoryTypePreDescriptorDerived() [1/2]

```
template<typename Type >
jeod::JeodMemoryTypePreDescriptorDerived< Type >::JeodMemoryTypePreDescriptorDerived (
    bool exportable = true ) [inline], [explicit]
```

Default constructor.

Definition at line 544 of file memory_type.hh.

8.16.3.2 JeodMemoryTypePreDescriptorDerived() [2/2]

```
template<typename Type >
jeod::JeodMemoryTypePreDescriptorDerived< Type >::JeodMemoryTypePreDescriptorDerived (
    const JeodMemoryTypePreDescriptorDerived< Type > & src ) [inline]
```

Copy constructor.

Definition at line 552 of file memory_type.hh.

References jeod::JeodMemoryTypePreDescriptorDerived< Type >::descriptor.

8.16.3.3 ~JeodMemoryTypePreDescriptorDerived()

```
template<typename Type >
jeod::JeodMemoryTypePreDescriptorDerived< Type >::~~JeodMemoryTypePreDescriptorDerived ( )
[inline], [override]
```

Destructor.

Definition at line 564 of file memory_type.hh.

References jeod::JeodMemoryTypePreDescriptorDerived< Type >::descriptor.

8.16.4 Member Function Documentation

8.16.4.1 get_descriptor()

```
template<typename Type >
const JeodMemoryTypeDescriptor& jeod::JeodMemoryTypePreDescriptorDerived< Type >::get_descriptor
( ) [inline], [override], [virtual]
```

Get a type descriptor for the type.

Note well: The referenced value has a lifespan limited to that of this object. The returned value must not be cached in a permanent store. Use new in conjunction with the copy constructor instead.

Returns

Type descriptor.

Implements [jeod::JeodMemoryTypePreDescriptor](#).

Definition at line 606 of file memory_type.hh.

References [jeod::JeodMemoryTypePreDescriptorDerived< Type >::descriptor](#), and [jeod::JeodMemoryTypePreDescriptorDerived< Type >::is_exportable](#).

8.16.4.2 get_ref()

```
template<typename Type >
JeodMemoryTypePreDescriptor& jeod::JeodMemoryTypePreDescriptorDerived< Type >::get_ref ( )
[inline]
```

Get a reference to this object.

This is an utter hack. Because the descriptor is created after the fact, a function that receives a [JeodMemoryTypePreDescriptor](#) must either take a copy or a non-const reference as input. A reference is preferred. The problem: Non-const references cannot be bound to rvalues. They can however be bound to other references, and hence this method.

Note well: The returned reference has a lifespan limited to that of this object. Use with great care. This is not intended for general consumption.

Returns

Reference to this object.

Definition at line 583 of file memory_type.hh.

8.16.4.3 get_typeid()

```
template<typename Type >
const std::type_info& jeod::JeodMemoryTypePreDescriptorDerived< Type >::get_typeid ( ) const
[inline], [override], [virtual]
```

Get the type info for the type.

Returns

Type info

Implements [jeod::JeodMemoryTypePreDescriptor](#).

Definition at line 592 of file `memory_type.hh`.

8.16.5 Field Documentation

8.16.5.1 descriptor

```
template<typename Type >
TypeDescriptor* jeod::JeodMemoryTypePreDescriptorDerived< Type >::descriptor {} [private]
```

Definition at line 616 of file `memory_type.hh`.

Referenced by `jeod::JeodMemoryTypePreDescriptorDerived< Type >::get_descriptor()`, `jeod::JeodMemoryTypePreDescriptorDerived< Type >::JeodMemoryTypePreDescriptorDerived()`, and `jeod::JeodMemoryTypePreDescriptorDerived< Type >::~~JeodMemoryTypePreDescriptorDerived()`.

8.16.5.2 is_exportable

```
template<typename Type >
bool jeod::JeodMemoryTypePreDescriptorDerived< Type >::is_exportable {true} [private]
```

Definition at line 617 of file `memory_type.hh`.

Referenced by `jeod::JeodMemoryTypePreDescriptorDerived< Type >::get_descriptor()`.

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

- [memory_type.hh](#)

8.17 jeod::JeodSimEngineAttributes< Type, is_class > Class Template Reference

Class template to construct a simulation engine attributes object that represents some type.

```
#include <memory_attributes_templates.hh>
```

Static Public Member Functions

- static struct ATTRIBUTES_tag [attributes](#) (bool)
Construct a JEOD_ATTRIBUTES_TYPE that represents a primitive type.

8.17.1 Detailed Description

```
template<typename Type, bool is_class>
class jeod::JeodSimEngineAttributes< Type, is_class >
```

Class template to construct a simulation engine attributes object that represents some type.

All partial template instantiations of this template define a class with a single static function named attributes. This default implementation is for a primitive type. Subsequent partial instantiations will address other types.

Template Parameters

<i>Type</i>	The type for which an attributes is to be constructed.
<i>is_class</i>	True if the type is a class, false otherwise.

Definition at line 91 of file memory_attributes_templates.hh.

8.17.2 Member Function Documentation

8.17.2.1 attributes()

```
template<typename Type , bool is_class>
static struct ATTRIBUTES_tag jeod::JeodSimEngineAttributes< Type, is_class >::attributes (
    bool ) [inline], [static]
```

Construct a JEOD_ATTRIBUTES_TYPE that represents a primitive type.

Returns

Constructed attributes object.

Definition at line 98 of file memory_attributes_templates.hh.

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

- [memory_attributes_templates.hh](#)

8.18 jeod::JeodSimEngineAttributes< Type *, false > Class Template Reference

Partial template instantiation of [JeodSimEngineAttributes](#) for a pointer type.

```
#include <memory_attributes_templates.hh>
```

Static Public Member Functions

- static struct ATTRIBUTES_tag [attributes](#) (bool is_exportable=true)
Construct a JEOD_ATTRIBUTES_TYPE that represents a pointer type.

8.18.1 Detailed Description

```
template<typename Type>
class jeod::JeodSimEngineAttributes< Type *, false >
```

Partial template instantiation of [JeodSimEngineAttributes](#) for a pointer type.

Template Parameters

<i>Type</i>	The pointed-to type.
-------------	----------------------

Definition at line 110 of file memory_attributes_templates.hh.

8.18.2 Member Function Documentation

8.18.2.1 attributes()

```
template<typename Type >
static struct ATTRIBUTES_tag jeod::JeodSimEngineAttributes< Type *, false >::attributes (
    bool is_exportable = true ) [inline], [static]
```

Construct a JEOD_ATTRIBUTES_TYPE that represents a pointer type.

Parameters

<i>is_exportable</i>	True => type is exportable.
----------------------	-----------------------------

Returns

Constructed attributes object.

Definition at line 118 of file memory_attributes_templates.hh.

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

- [memory_attributes_templates.hh](#)

8.19 jeod::JeodSimEngineAttributes< Type, true > Class Template Reference

Partial template instantiation of [JeodSimEngineAttributes](#) for a class.

```
#include <memory_attributes_templates.hh>
```


Static Public Member Functions

- static struct ATTRIBUTES_tag [attributes](#) (bool is_exportable=true)
Construct a JEOD_ATTRIBUTES_TYPE that represents a structured type.

8.19.1 Detailed Description

```
template<typename Type>
class jeod::JeodSimEngineAttributes< Type, true >
```

Partial template instantiation of [JeodSimEngineAttributes](#) for a class.

Template Parameters

<i>Type</i>	The class.
-------------	------------

Definition at line 148 of file memory_attributes_templates.hh.

8.19.2 Member Function Documentation

8.19.2.1 attributes()

```
template<typename Type >
static struct ATTRIBUTES_tag jeod::JeodSimEngineAttributes< Type, true >::attributes (
    bool is_exportable = true ) [inline], [static]
```

Construct a JEOD_ATTRIBUTES_TYPE that represents a structured type.

Parameters

<i>is_exportable</i>	True => type is exportable.
----------------------	-----------------------------

Returns

Constructed attributes object.

Definition at line 156 of file memory_attributes_templates.hh.

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

- [memory_attributes_templates.hh](#)

8.20 jeod::JeodSimEngineAttributes< void *, false > Class Template Reference

Template specialization of [JeodSimEngineAttributes](#) for void*.

```
#include <memory_attributes_templates.hh>
```

Static Public Member Functions

- static struct ATTRIBUTES_tag [attributes](#) (bool)
Construct a JEOD_ATTRIBUTES_TYPE that represents a void pointer.

8.20.1 Detailed Description

```
template<>
class jeod::JeodSimEngineAttributes< void *, false >
```

Template specialization of [JeodSimEngineAttributes](#) for void*.

Definition at line 130 of file memory_attributes_templates.hh.

8.20.2 Member Function Documentation

8.20.2.1 attributes()

```
static struct ATTRIBUTES_tag jeod::JeodSimEngineAttributes< void *, false >::attributes (
    bool ) [inline], [static]
```

Construct a JEOD_ATTRIBUTES_TYPE that represents a void pointer.

Returns

Constructed attributes object.

Definition at line 137 of file memory_attributes_templates.hh.

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

- [memory_attributes_templates.hh](#)

8.21 jeod::MemoryMessages Class Reference

Declares messages associated with the integration test model.

```
#include <memory_messages.hh>
```

Public Member Functions

- [MemoryMessages](#) ()=delete
- [MemoryMessages](#) (const [MemoryMessages](#) &)=delete
- [MemoryMessages](#) & operator= (const [MemoryMessages](#) &)=delete

Static Public Attributes

- static const char * [singleton_error](#) = "utils/memory/" "singleton_error"
Error issued when multiple instance of a class that should be a singleton are created or when no such instance exists (but should).
- static const char * [out_of_memory](#) = "utils/memory/" "out_of_memory"
Issued when malloc returns NULL.
- static const char * [lock_error](#) = "utils/memory/" "lock_error"
Issued when problems arise with in protection for atomic operations.
- static const char * [null_pointer](#) = "utils/memory/" "null_pointer"
Issued when the caller attempts to do something with a null pointer such as registering or freeing.
- static const char * [suspect_pointer](#) = "utils/memory/" "suspect_pointer"
Issued when the caller attempts to register memory that overlaps with previously recording allocations or attempts to destroy memory that was not previously registered.
- static const char * [invalid_size](#) = "utils/memory/" "invalid_size"
Issued when the caller attempts to allocate zero bytes.
- static const char * [corrupted_memory](#) = "utils/memory/" "corrupted_memory"
Issued when guard bytes have been overwritten.
- static const char * [registration_error](#) = "utils/memory/" "registration_error"
Issued when a model programmer messed up.
- static const char * [internal_error](#) = "utils/memory/" "internal_error"
Issued when the memory model programmer messed up.
- static const char * [debug](#) = "utils/memory/" "debug"
Used to identify debug output.

Friends

- class [InputProcessor](#)
- void [init_attrjeod__MemoryMessages](#) ()

8.21.1 Detailed Description

Declares messages associated with the integration test model.

Definition at line 85 of file `memory_messages.hh`.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 MemoryMessages() [1/2]

```
jeod::MemoryMessages::MemoryMessages ( ) [delete]
```

8.21.2.2 MemoryMessages() [2/2]

```
jeod::MemoryMessages::MemoryMessages (
    const MemoryMessages & ) [delete]
```

8.21.3 Member Function Documentation

8.21.3.1 operator=()

```
MemoryMessages& jeod::MemoryMessages::operator= (
    const MemoryMessages & ) [delete]
```

8.21.4 Friends And Related Function Documentation

8.21.4.1 init_attrjeod__MemoryMessages

```
void init_attrjeod__MemoryMessages ( ) [friend]
```

8.21.4.2 InputProcessor

```
friend class InputProcessor [friend]
```

Definition at line 87 of file memory_messages.hh.

8.21.5 Field Documentation

8.21.5.1 corrupted_memory

```
char const * jeod::MemoryMessages::corrupted_memory = "utils/memory/" "corrupted_memory" [static]
```

Issued when guard bytes have been overwritten.

```
trick_units(-)
```

Definition at line 126 of file memory_messages.hh.

Referenced by jeod::JeodMemoryManager::add_allocation_atomic(), jeod::JeodMemoryManager::free_memory(), jeod::JeodMemoryManager::generate_shutdown_report(), and jeod::JeodMemoryManager::get_alloc_id_atomic().

8.21.5.2 debug

```
char const * jeod::MemoryMessages::debug = "utils/memory/" "debug" [static]
```

Used to identify debug output.

trick_units(-)

Definition at line 141 of file memory_messages.hh.

Referenced by jeod::JeodMemoryManager::destroy_memory_internal(), jeod::JeodMemoryManager::generate_shutdown_report(), jeod::JeodMemoryManager::get_type_entry_atomic(), and jeod::JeodMemoryManager::register_memory_internal().

8.21.5.3 internal_error

```
char const * jeod::MemoryMessages::internal_error = "utils/memory/" "internal_error" [static]
```

Issued when the memory model programmer messed up.

trick_units(-)

Definition at line 136 of file memory_messages.hh.

Referenced by jeod::JeodMemoryManager::get_string_atomic(), jeod::JeodMemoryManager::get_type_descriptor_atomic(), and jeod::JeodMemoryItem::set_unique_id().

8.21.5.4 invalid_size

```
char const * jeod::MemoryMessages::invalid_size = "utils/memory/" "invalid_size" [static]
```

Issued when the caller attempts to allocate zero bytes.

trick_units(-)

Definition at line 121 of file memory_messages.hh.

Referenced by jeod::JeodMemoryManager::register_memory_internal().

8.21.5.5 lock_error

```
char const * jeod::MemoryMessages::lock_error = "utils/memory/" "lock_error" [static]
```

Issued when problems arise with in protection for atomic operations.

trick_units(-)

Definition at line 103 of file memory_messages.hh.

Referenced by jeod::JeodMemoryManager::begin_atomic_block(), and jeod::JeodMemoryManager::end_atomic_block().

8.21.5.6 null_pointer

```
char const * jeod::MemoryMessages::null_pointer = "utils/memory/" "null_pointer" [static]
```

Issued when the caller attempts to do something with a null pointer such as registering or freeing.

trick_units(-)

Definition at line 109 of file memory_messages.hh.

Referenced by `jeod::JeodMemoryManager::deregister_container()`, `jeod::JeodMemoryManager::destroy_↔memory_internal()`, and `jeod::JeodMemoryManager::register_container()`.

8.21.5.7 out_of_memory

```
char const * jeod::MemoryMessages::out_of_memory = "utils/memory/" "out_of_memory" [static]
```

Issued when malloc returns NULL.

trick_units(-)

Definition at line 98 of file memory_messages.hh.

Referenced by `jeod::JeodMemoryManager::allocate_memory()`.

8.21.5.8 registration_error

```
char const * jeod::MemoryMessages::registration_error = "utils/memory/" "registration_error" [static]
```

Issued when a model programmer messed up.

trick_units(-)

Definition at line 131 of file memory_messages.hh.

8.21.5.9 singleton_error

```
char const * jeod::MemoryMessages::singleton_error = "utils/memory/" "singleton_error" [static]
```

Error issued when multiple instance of a class that should be a singleton are created or when no such instance exists (but should).

trick_units(-)

Definition at line 93 of file memory_messages.hh.

Referenced by `jeod::JeodMemoryManager::check_master()`, and `jeod::JeodMemoryManager::JeodMemory↔Manager()`.

8.21.5.10 suspect_pointer

```
char const * jeod::MemoryMessages::suspect_pointer = "utils/memory/" "suspect_pointer" [static]
```

Issued when the caller attempts to register memory that overlaps with previously recording allocations or attempts to destroy memory that was not previously registered.

trick_units(-)

Definition at line 116 of file memory_messages.hh.

Referenced by `jeod::JeodMemoryManager::destroy_memory_internal()`, `jeod::JeodMemoryManager::find_alloc_entry_atomic()`, and `jeod::JeodMemoryManager::restart_reallocate()`.

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

- [memory_messages.hh](#)
- [memory_messages.cc](#)

8.22 jeod::JeodMemoryManager::TypeEntry Struct Reference

The type table is indexed by an integer and contains type descriptors.

```
#include <memory_manager.hh>
```

Public Member Functions

- [TypeEntry](#) (uint32_t num, const [JeodMemoryTypeDescriptor](#) *desc)
Pair constructor.

Data Fields

- uint32_t [index](#)
Type table index number.
- const [JeodMemoryTypeDescriptor](#) * [tdesc](#)
Type descriptor.

8.22.1 Detailed Description

The type table is indexed by an integer and contains type descriptors.

This class bundles the two together.

Definition at line 248 of file memory_manager.hh.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 TypeEntry()

```
jeod::JeodMemoryManager::TypeEntry::TypeEntry (
    uint32_t num,
    const JeodMemoryTypeDescriptor * desc ) [inline]
```

Pair constructor.

Definition at line 263 of file memory_manager.hh.

8.22.3 Field Documentation

8.22.3.1 index

```
uint32_t jeod::JeodMemoryManager::TypeEntry::index
```

Type table index number.

trick_io(**)

Definition at line 253 of file memory_manager.hh.

Referenced by jeod::JeodMemoryManager::register_memory_internal().

8.22.3.2 tdesc

```
const JeodMemoryTypeDescriptor* jeod::JeodMemoryManager::TypeEntry::tdesc
```

Type descriptor.

trick_io(**)

Definition at line 258 of file memory_manager.hh.

Referenced by jeod::JeodMemoryManager::create_memory_internal(), jeod::JeodMemoryManager::get_type_descriptor(), jeod::JeodMemoryManager::register_memory_internal(), and jeod::JeodMemoryManager::restart_reallocate().

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

- [memory_manager.hh](#)

Chapter 9

File Documentation

9.1 class_declarations.hh File Reference

Forward declarations of classes defined in the utils/memory model.

Namespaces

- [jeod](#)

Namespace jeod.

9.1.1 Detailed Description

Forward declarations of classes defined in the utils/memory model.

9.2 jeod_alloc.hh File Reference

Define JEOD memory allocation macros.

```
#include <cstdint>
#include <new>
#include "utils/sim_interface/include/memory_attributes.hh"
#include "jeod_alloc_get_allocated_pointer.hh"
#include "memory_manager.hh"
```

Macros

- `#define JEOD_MEMORY_DEBUG 2`
Specifies the level of checking performed by the JEOD memory model.
- `#define JEOD_ALLOC_OBJECT_FILL 0xdf`
Fill pattern for non-primitive types.
- `#define JEOD_ALLOC_PRIMITIVE_FILL 0`
Fill pattern for primitive types.
- `#define JEOD_ALLOC_POINTER_FILL 0`
Fill pattern for pointer types.
- `#define JEOD_CREATE_MEMORY(is_array, nelem, fill, tentry) jeod::JeodMemoryManager::create_memory(is_array, nelem, fill, tentry, __FILE__, __LINE__)`
Allocate and register memory to be populated via placement new.
- `#define JEOD_ALLOC_ARRAY_INTERNAL(type, nelem, fill, tentry) new(JEOD_CREATE_MEMORY(true, nelem, fill, tentry)) type[nelem]`
Allocate nelem elements of pointers to the specified structured type.
- `#define JEOD_ALLOC_OBJECT_INTERNAL(type, fill, constr, tentry) new(JEOD_CREATE_MEMORY(false, 1, fill, tentry)) type constr`
Allocate an instance of the specified class using the specified constructor arguments.
- `#define JEOD_DELETE_INTERNAL(ptr, is_array)`
Free memory allocated with some JEOD_ALLOC macro.
- `#define JEOD_REGISTER_CLASS(type) jeod::JeodMemoryManager::register_class(jeod::JeodMemoryTypePreDescriptorDer_ref())`
Register the type type with the memory manager.
- `#define JEOD_REGISTER_INCOMPLETE_CLASS(type) JEOD_REGISTER_CLASS(type)`
Register the incomplete class type with the memory manager.
- `#define JEOD_REGISTER_NONEXPORTED_CLASS(type) jeod::JeodMemoryManager::register_class(jeod::JeodMemoryTypePreDescriptorDer_ref())`
Register the type type with the memory manager, but with the class marked as not exportable to the simulation engine.
- `#define JEOD_REGISTER_CHECKPOINTABLE(owner, elem_name)`
Register the data member elem_name of the owner as a Checkpointable object.
- `#define JEOD_DEREGISTER_CHECKPOINTABLE(owner, elem_name)`
Register the data member elem_name of the owner as a Checkpointable object.
- `#define JEOD_ALLOC_CLASS_MULTI_POINTER_ARRAY(nelem, type, asters) JEOD_ALLOC_ARRAY_INTERNAL(type, asters, nelem, JEOD_ALLOC_POINTER_FILL, JEOD_REGISTER_CLASS(type, asters))`
Allocate an array of nelem multi-level pointers to the specified type.
- `#define JEOD_ALLOC_CLASS_POINTER_ARRAY(nelem, type) JEOD_ALLOC_CLASS_MULTI_POINTER_ARRAY(nelem, type, *)`
Allocate an array of nelem pointers to the specified type.
- `#define JEOD_ALLOC_CLASS_ARRAY(nelem, type) JEOD_ALLOC_ARRAY_INTERNAL(type, nelem, JEOD_ALLOC_OBJECT_FILL, JEOD_REGISTER_CLASS(type))`
Allocate an array of nelem instances of the specified structured type.
- `#define JEOD_ALLOC_PRIM_ARRAY(nelem, type) JEOD_ALLOC_ARRAY_INTERNAL(type, nelem, JEOD_ALLOC_PRIMITIVE_FILL, JEOD_REGISTER_CLASS(type))`
Allocate nelem elements of the specified primitive type.
- `#define JEOD_ALLOC_CLASS_OBJECT(type, constr) JEOD_ALLOC_OBJECT_INTERNAL(type, JEOD_ALLOC_OBJECT_FILL, constr, JEOD_REGISTER_CLASS(type))`
Allocate one instance of the specified class.
- `#define JEOD_ALLOC_PRIM_OBJECT(type, initial) JEOD_ALLOC_OBJECT_INTERNAL(type, JEOD_ALLOC_PRIMITIVE_FILL, (initial), JEOD_REGISTER_CLASS(type))`
Allocate one instance of the specified type.
- `#define JEOD_IS_ALLOCATED(ptr) jeod::JeodMemoryManager::is_allocated(jeod::jeod_alloc_get_allocated_pointer(ptr), __FILE__, __LINE__)`

- Determine if ptr was allocated by some JEOD_ALLOC_XXX_ARRAY macro.*

 - #define `JEOD_DELETE_ARRAY(ptr) JEOD_DELETE_INTERNAL(ptr, true)`

Free memory at ptr that was earlier allocated with some JEOD_ALLOC_XXX_ARRAY macro.
- #define `JEOD_DELETE_OBJECT(ptr) JEOD_DELETE_INTERNAL(ptr, false)`

Free memory at ptr that was earlier allocated with some JEOD_ALLOC_XXX_OBJECT macro.
- #define `JEOD_DELETE_2D(ptr, size, is_array)`

9.2.1 Detailed Description

Define JEOD memory allocation macros.

The [jeod_alloc.hh](#) memory macros can be viewed as

- Being externally-usable or for internal use only.
The supported use of the JEOD memory model is via those macros advertised as externally-usable. These externally-usable macros expand into invocations of internal macros, which in turn expand into calls to methods of classes defined in the memory model. Those macros marked as internal are for internal use only by this file.
- Supporting allocation versus deletion.
Some of the [jeod_alloc.hh](#) memory macros allocate memory while others delete it. With one exception, the allocation/delete nature of a macro is explicit in the macro name. Allocation macros start with JEOD_ALLOC. Macros that address deleting memory start with JEOD_DELETE.
- Operating on objects versus arrays.
The memory management macros come in two basic forms: ARRAY and OBJECT. Memory allocated with an ARRAY allocator macro must be freed with JEOD_DELETE_ARRAY. Memory allocated with an OBJECT allocator macro must be freed with JEOD_DELETE_OBJECT. This corresponds to the C++ distinction between operator new[], delete[], new, and delete.
- Operating on structured versus non-structured data.
The JEOD memory model registers allocated memory with the underlying simulation engine (e.g., Trick). To make the data in a structured type visible to the engine, the user must declare an external reference to the engine's description of the type. For example, to allocate an instance of some class Foo using the default constructor use

```
JEOD_DECLARE_ATTRIBUTES (Foo)
...
Foo * foo_obj = JEOD_ALLOC_CLASS (Foo, ());
```

See JEOD_DECLARE_ATTRIBUTES.

Two compile -D options affect the behavior of these macros. These are

- JEOD_MEMORY_DEBUG - The memory model debugging level. The debugging level ranges from 0 (off) to 3 (all transactions). If this is not set in the compile flags the value is set to 0 (off).
- JEOD_MEMORY_GUARD - Guards will be added around allocated memory if this option is defined and has a non-zero value.

9.3 jeod_alloc_construct_destruct.hh File Reference

Define templates for use by [jeod_alloc.hh](#).

```
#include "utils/sim_interface/include/jeod_class.hh"
#include <cstdint>
#include <cstring>
#include <type_traits>
```

Data Structures

- class `jeod::JeodAllocHelperConstructDestruct< T, is_class, is_abstract >`
Class template that provides static functions construct and destruct that construct an array of objects.
- class `jeod::JeodAllocHelperConstructDestruct< T, false, is_abstract >`
Partial instantiation for non-classes.
- class `jeod::JeodAllocHelperConstructDestruct< T, true, false >`
Partial instantiation for non-abstract classes.

Namespaces

- `jeod`
Namespace jeod.

Functions

- template<typename T >
void * `jeod::jeod_alloc_construct_array` (std::size_t nelem, void *addr)
Construct an array of objects of type T.
- template<typename T >
void `jeod::jeod_alloc_destruct_array` (std::size_t nelem, void *addr)
Destruct an array of objects of type T.

9.3.1 Detailed Description

Define templates for use by `jeod_alloc.hh`.

These are isolated from `jeod_alloc.hh` because

- They are templates; everything in `jeod_alloc.hh` is a macro.
- Some of the templates might have wider interest than JEOD.
- Some of this stuff can go away with C++11.

The externally-usable items defined in this file are

- Function template `jeod_alloc_construct_array`, and
- Function template `jeod_alloc_destruct_array`.

9.4 jeod_alloc_get_allocated_pointer.hh File Reference

Define function template `jeod_alloc_get_allocated_pointer`.

```
#include <cstddef>
#include <cstring>
#include <type_traits>
```

Data Structures

- class [jeod::JeodAllocHelperAllocatedPointer< T, is_poly >](#)
Class template that provides a static function cast that casts a pointer to an object of type T to a void pointer.*
- class [jeod::JeodAllocHelperAllocatedPointer< T, true >](#)
Partial instantiation of [JeodAllocHelperAllocatedPointer](#) for polymorphic classes.

Namespaces

- [jeod](#)
Namespace jeod.

Functions

- `template<typename T >`
`void * jeod::jeod_alloc_get_allocated_pointer (T *pointer)`
Cast a pointer to some object to a pointer to void such that a pointer to a polymorphic object, downcast to a base class pointer, becomes a pointer to the original object, but also such that a pointer to an instance of a non-polymorphic class or a pointer to a non-class type is handled correctly.*

9.4.1 Detailed Description

Define function template `jeod_alloc_get_allocated_pointer`.

9.5 memory_attributes_templates.hh File Reference

Define the class template `JeodSimEngineAttributes`.

```
#include "utils/sim_interface/include/memory_attributes.hh"
#include "utils/sim_interface/include/memory_interface.hh"
#include "utils/sim_interface/include/simulation_interface.hh"
#include <type_traits>
#include <typeinfo>
```

Data Structures

- class [jeod::JeodSimEngineAttributes< Type, is_class >](#)
Class template to construct a simulation engine attributes object that represents some type.
- class [jeod::JeodSimEngineAttributes< Type *, false >](#)
Partial template instantiation of [JeodSimEngineAttributes](#) for a pointer type.
- class [jeod::JeodSimEngineAttributes< void *, false >](#)
Template specialization of [JeodSimEngineAttributes](#) for void.*
- class [jeod::JeodSimEngineAttributes< Type, true >](#)
Partial template instantiation of [JeodSimEngineAttributes](#) for a class.

Namespaces

- [jeod](#)

Namespace jeod.

9.5.1 Detailed Description

Define the class template JeodSimEngineAttributes.

9.6 memory_item.cc File Reference

Implement the JeodMemoryItem class.

```
#include "utils/message/include/message_handler.hh"
#include "../include/memory_item.hh"
#include "../include/memory_messages.hh"
```

Namespaces

- [jeod](#)

Namespace jeod.

9.6.1 Detailed Description

Implement the JeodMemoryItem class.

9.7 memory_item.hh File Reference

Define the class JeodMemoryItem.

```
#include <cstdint>
#include "utils/sim_interface/include/jeod_class.hh"
```

Data Structures

- class [jeod::JeodMemoryItem](#)

A [JeodMemoryItem](#) contains metadata about some chunk of allocated memory.

Namespaces

- [jeod](#)

Namespace jeod.

9.7.1 Detailed Description

Define the class JeodMemoryItem.

9.8 memory_manager.cc File Reference

Implement the JeodMemoryManager class.

```
#include <cstdint>
#include <cstdint>
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <map>
#include <pthread.h>
#include <sstream>
#include <typeinfo>
#include "utils/message/include/message_handler.hh"
#include "../include/memory_item.hh"
#include "../include/memory_manager.hh"
#include "../include/memory_messages.hh"
```

Namespaces

- [jeod](#)
Namespace jeod.

Macros

- `#define MAKE_DESCRIPTOR(type)`

Variables

- static constexpr uint32_t [jeod::MAGIC0](#) = 0x2203992c
- static constexpr uint32_t [jeod::MAGIC1](#) = 0x6c052d84

9.8.1 Detailed Description

Implement the JeodMemoryManager class.

9.8.2 Macro Definition Documentation

9.8.2.1 MAKE_DESCRIPTOR

```
#define MAKE_DESCRIPTOR(  
    type )
```

Value:

```
do  
    {  
        JeodMemoryTypeDescriptorDerived<type> tdesc;  
        type_table.add(tdesc.get_typeid().name(), tdesc);  
    } while (0)
```

Referenced by `jeod::JeodMemoryManager::JeodMemoryManager()`.

9.9 memory_manager.hh File Reference

Define the `JeodMemoryManager` class, the central agent of the memory model.

```
#include <cstdlib>  
#include <list>  
#include <map>  
#include <ostream>  
#include <pthread.h>  
#include <string>  
#include <typeinfo>  
#include "utils/container/include/checkpointable.hh"  
#include "utils/sim_interface/include/config.hh"  
#include "utils/sim_interface/include/jeod_class.hh"  
#include "utils/sim_interface/include/memory_interface.hh"  
#include "utils/sim_interface/include/simulation_interface.hh"  
#include "memory_item.hh"  
#include "memory_table.hh"  
#include "memory_type.hh"
```

Data Structures

- class [jeod::JeodMemoryManager](#)
This class provides the interface between the macros in [jeod_alloc.hh](#) and the rest of the JEOD memory model.
- struct [jeod::JeodMemoryManager::TypeEntry](#)
The type table is indexed by an integer and contains type descriptors.

Namespaces

- [jeod](#)
Namespace `jeod`.

9.9.1 Detailed Description

Define the JeodMemoryManager class, the central agent of the memory model.

9.10 memory_manager_hide_from_trick.hh File Reference

Trick doesn't understand these.

```
#include "memory_item.hh"
#include "memory_table.hh"
#include "memory_type.hh"
#include <map>
```

Namespaces

- [jeod](#)
Namespace jeod.

Typedefs

- using [jeod::AllocTable](#) = std::map< const void *, JeodMemoryItem >
An AllocTable maps memory addresses to memory descriptions.
- using [jeod::TypeTable](#) = JeodMemoryTableClonable< JeodMemoryTypeDescriptor >
The type type itself is a memory table with copy implemented by clone().

9.10.1 Detailed Description

Trick doesn't understand these.

This file is included from the private part of [memory_manager.hh](#). The types are private and the corresponding members hidden from Trick. These will be folded into [memory_manager.hh](#) when Trick ICG, both Trick 7 and Trick 10, understands these or provides a common mechanism for telling ICG to ignore content.

9.11 memory_manager_protected.cc File Reference

Implement those JeodMemoryManager member functions that access data members that need to be treated with care to make the memory manager thread safe.

```
#include <cstddef>
#include <stdint>
#include <stdlib>
#include <iomanip>
#include <iostream>
#include <map>
#include <pthread.h>
#include <sstream>
#include <typeinfo>
#include "utils/message/include/message_handler.hh"
#include "../include/memory_item.hh"
#include "../include/memory_manager.hh"
#include "../include/memory_messages.hh"
```

Namespaces

- [jeod](#)

Namespace jeod.

Macros

- `#define __STDC_LIMIT_MACROS`

9.11.1 Detailed Description

Implement those JeodMemoryManager member functions that access data members that need to be treated with care to make the memory manager thread safe.

9.12 memory_manager_static.cc File Reference

Implement the static methods of the JeodMemoryManager class.

```
#include <string>
#include "utils/message/include/message_handler.hh"
#include "utils/named_item/include/named_item.hh"
#include "../include/memory_manager.hh"
#include "../include/memory_messages.hh"
```

Namespaces

- [jeod](#)

Namespace jeod.

9.12.1 Detailed Description

Implement the static methods of the JeodMemoryManager class.

9.13 memory_messages.cc File Reference

Implement the class MemoryMessages.

```
#include "utils/message/include/make_message_code.hh"
#include "../include/memory_messages.hh"
```

Namespaces

- [jeod](#)

Namespace jeod.

Macros

- `#define MAKE_MEMORY_MESSAGE_CODE(id) JEOD_MAKE_MESSAGE_CODE(MemoryMessages, "utils/memory/", id)`

9.13.1 Detailed Description

Implement the class MemoryMessages.

9.13.2 Macro Definition Documentation

9.13.2.1 MAKE_MEMORY_MESSAGE_CODE

```
#define MAKE_MEMORY_MESSAGE_CODE(  
    id ) JEOD_MAKE_MESSAGE_CODE(MemoryMessages, "utils/memory/", id)
```

Definition at line 43 of file memory_messages.cc.

9.14 memory_messages.hh File Reference

Define the class MemoryMessages, the class that specifies the message IDs used in the memory model.

```
#include "utils/sim_interface/include/jeod_class.hh"
```

Data Structures

- class [jeod::MemoryMessages](#)
Declares messages associated with the integration test model.

Namespaces

- [jeod](#)
Namespace jeod.

9.14.1 Detailed Description

Define the class MemoryMessages, the class that specifies the message IDs used in the memory model.

9.15 memory_table.hh File Reference

Define classes for representing data types.

```
#include "utils/sim_interface/include/jeod_class.hh"
#include <cstdint>
#include <map>
#include <stdexcept>
#include <string>
#include <vector>
```

Data Structures

- class [jeod::JeodMemoryTable< ValueType >](#)
A *JeodMemoryTable* maps strings to values with a coordinated map/vector pair.
- class [jeod::JeodMemoryTableClonable< ValueType >](#)
A *JeodMemoryTableClonable* is a *JeodMemoryTable* that implements the required [clone\(\)](#) functionality by invoking the *ValueType*'s [clone\(\)](#) method to create a clone of the input value.
- class [jeod::JeodMemoryTableCopyable< ValueType >](#)
A *JeodMemoryTableCopyable* is a *JeodMemoryTable* that implements the required [clone\(\)](#) functionality by invoking the *ValueType*'s copy constructor to create a clone of the input value.
- class [jeod::JeodMemoryReflectiveTable](#)
A *JeodMemoryReflectiveTable* maps strings to themselves.

Namespaces

- [jeod](#)
Namespace *jeod*.

9.15.1 Detailed Description

Define classes for representing data types.

9.16 memory_type.cc File Reference

Implement destructors for the classes for representing data types.

```
#include <cstdint>
#include <sstream>
#include <string>
#include "utils/named_item/include/named_item.hh"
#include "utils/sim_interface/include/simulation_interface.hh"
#include "../include/memory_item.hh"
#include "../include/memory_manager.hh"
#include "../include/memory_type.hh"
```

Namespaces

- [jeod](#)
Namespace jeod.

9.16.1 Detailed Description

Implement destructors for the classes for representing data types.

9.17 memory_type.hh File Reference

Define the abstract class JeodMemoryTypeDescriptor and templates that create instantiable classes that derive from JeodMemoryTypeDescriptor.

```
#include "jeod_alloc_construct_destruct.hh"
#include "jeod_alloc_get_allocated_pointer.hh"
#include "memory_attributes_templates.hh"
#include "memory_item.hh"
#include "memory_messages.hh"
#include "utils/message/include/message_handler.hh"
#include "utils/sim_interface/include/jeod_class.hh"
#include "utils/sim_interface/include/memory_attributes.hh"
#include <cstdint>
#include <cstring>
#include <new>
#include <string>
#include <type_traits>
#include <typeinfo>
```

Data Structures

- class [jeod::JeodMemoryTypeDescriptor](#)
Abstract class for managing data allocated as some specific type.
- struct [jeod::JeodMemoryTypeDescriptor::attr](#)
*The simulation engine attributes that describe the type.trick_io(**)*
- class [jeod::JeodMemoryTypeDescriptorDerived< Type >](#)
Extends JeodMemoryTypeDescriptor to describe a specific type.
- class [jeod::JeodMemoryTypePreDescriptor](#)
Abstract class for describing a type without necessarily needing to create a JeodMemoryTypeDescriptor of that type.
- class [jeod::JeodMemoryTypePreDescriptorDerived< Type >](#)
A JeodMemoryTypePreDescriptorDerived describes a Type.

Namespaces

- [jeod](#)
Namespace jeod.

9.17.1 Detailed Description

Define the abstract class JeodMemoryTypeDescriptor and templates that create instantiable classes that derive from JeodMemoryTypeDescriptor.

Index

- `__STDC_LIMIT_MACROS`
 - Support classes, [25](#)
- `~JeodMemoryItem`
 - `jeod::JeodMemoryItem`, [47](#)
- `~JeodMemoryManager`
 - `jeod::JeodMemoryManager`, [61](#)
- `~JeodMemoryTable`
 - `jeod::JeodMemoryTable`, [93](#)
- `~JeodMemoryTypeDescriptor`
 - `jeod::JeodMemoryTypeDescriptor`, [104](#)
- `~JeodMemoryTypeDescriptorDerived`
 - `jeod::JeodMemoryTypeDescriptorDerived`, [117](#)
- `~JeodMemoryTypePreDescriptor`
 - `jeod::JeodMemoryTypePreDescriptor`, [121](#)
- `~JeodMemoryTypePreDescriptorDerived`
 - `jeod::JeodMemoryTypePreDescriptorDerived`, [124](#)
- `add`
 - `jeod::JeodMemoryReflectiveTable`, [89](#)
 - `jeod::JeodMemoryTable`, [94](#)
- `add_allocation_atomic`
 - `jeod::JeodMemoryManager`, [61](#)
- `add_string_atomic`
 - `jeod::JeodMemoryManager`, [62](#)
- `alloc_info_index`
 - `jeod::JeodMemoryItem`, [52](#)
- `alloc_table`
 - `jeod::JeodMemoryManager`, [84](#)
- `AllocTable`
 - `jeod`, [32](#)
 - `jeod::JeodMemoryManager`, [59](#)
- `allocate_memory`
 - `jeod::JeodMemoryManager`, [63](#)
- `allocation_number`
 - `jeod::JeodMemoryManager`, [84](#)
- `Attributes`
 - `jeod::JeodMemoryTypeDescriptorDerived`, [116](#)
- `attributes`
 - `jeod::JeodSimEngineAttributes`, [127](#)
 - `jeod::JeodSimEngineAttributes< Type *, false >`, [128](#)
 - `jeod::JeodSimEngineAttributes< Type, true >`, [129](#)
 - `jeod::JeodSimEngineAttributes< void *, false >`, [130](#)
- `base_type`
 - `jeod::JeodMemoryTypeDescriptor`, [105](#)
- `begin`
 - `jeod::JeodMemoryTable`, [94](#)
- `begin_atomic_block`
 - `jeod::JeodMemoryManager`, [63](#)
- `buffer_end`
 - `jeod::JeodMemoryTypeDescriptor`, [105](#)
- `buffer_size`
 - `jeod::JeodMemoryTypeDescriptor`, [106](#)
- `cast`
 - `jeod::JeodAllocHelperAllocatedPointer`, [38](#)
 - `jeod::JeodAllocHelperAllocatedPointer< T, true >`, [39](#)
- `check_for_registration_errors`
 - `jeod::JeodMemoryTypeDescriptor`, [113](#)
- `check_master`
 - `jeod::JeodMemoryManager`, [64](#)
- `class_declarations.hh`, [137](#)
- `clone`
 - `jeod::JeodMemoryTable`, [94](#)
 - `jeod::JeodMemoryTableClonable`, [99](#)
 - `jeod::JeodMemoryTableCopyable`, [101](#)
 - `jeod::JeodMemoryTypeDescriptor`, [107](#)
 - `jeod::JeodMemoryTypeDescriptorDerived`, [117](#)
- `const_value_iterator`
 - `jeod::JeodMemoryTable`, [92](#)
- `construct`
 - `jeod::JeodAllocHelperConstructDestruct`, [40](#)
 - `jeod::JeodAllocHelperConstructDestruct< T, false, is_abstract >`, [42](#)
 - `jeod::JeodAllocHelperConstructDestruct< T, true, false >`, [43](#)
- `construct_array`
 - `jeod::JeodMemoryTypeDescriptor`, [107](#)
 - `jeod::JeodMemoryTypeDescriptorDerived`, [118](#)
- `construct_flags`
 - `jeod::JeodMemoryItem`, [47](#)
- `corrupted_memory`
 - `jeod::MemoryMessages`, [132](#)
- `create_memory`
 - `jeod::JeodMemoryManager`, [65](#)
- `create_memory_internal`
 - `jeod::JeodMemoryManager`, [66](#)
- `cur_data_size`
 - `jeod::JeodMemoryManager`, [85](#)
- `debug`
 - `jeod::MemoryMessages`, [132](#)
- `debug_level`
 - `jeod::JeodMemoryManager`, [85](#)
- `DebugLevel`
 - `jeod::JeodMemoryManager`, [60](#)
- `del`

- jeod::JeodMemoryTable, 95
- delete_array
 - jeod::JeodMemoryTypeDescriptor, 107
 - jeod::JeodMemoryTypeDescriptorDerived, 118
- delete_object
 - jeod::JeodMemoryTypeDescriptor, 108
 - jeod::JeodMemoryTypeDescriptorDerived, 118
- delete_oldest_alloc_entry_atomic
 - jeod::JeodMemoryManager, 66
- deregister_container
 - jeod::JeodMemoryManager, 67
- descriptor
 - jeod::JeodMemoryTypePreDescriptorDerived, 126
- descriptor_index_hi
 - jeod::JeodMemoryItem, 52
- descriptor_index_lo
 - jeod::JeodMemoryItem, 52
- destroy_memory
 - jeod::JeodMemoryManager, 68
 - jeod::JeodMemoryTypeDescriptor, 108
- destroy_memory_internal
 - jeod::JeodMemoryManager, 68
- destruct
 - jeod::JeodAllocHelperConstructDestruct, 41
 - jeod::JeodAllocHelperConstructDestruct< T, false, is_abstract >, 42
 - jeod::JeodAllocHelperConstructDestruct< T, true, false >, 43
- destruct_array
 - jeod::JeodMemoryTypeDescriptor, 109
 - jeod::JeodMemoryTypeDescriptorDerived, 119
- dimensionality
 - jeod::JeodMemoryTypeDescriptor, 109
- end
 - jeod::JeodMemoryTable, 95
- end_atomic_block
 - jeod::JeodMemoryManager, 69
- Externally-usable macros, 11
 - JEOD_ALLOC_CLASS_ARRAY, 12
 - JEOD_ALLOC_CLASS_MULTI_POINTER_ARRAY, 13
 - JEOD_ALLOC_CLASS_OBJECT, 13
 - JEOD_ALLOC_CLASS_POINTER_ARRAY, 14
 - JEOD_ALLOC_PRIM_ARRAY, 14
 - JEOD_ALLOC_PRIM_OBJECT, 15
 - JEOD_DELETE_2D, 15
 - JEOD_DELETE_ARRAY, 16
 - JEOD_DELETE_OBJECT, 17
 - JEOD_DEREGISTER_CHECKPOINTABLE, 17
 - JEOD_IS_ALLOCATED, 18
 - JEOD_MEMORY_DEBUG, 18
 - JEOD_REGISTER_CHECKPOINTABLE, 18
 - JEOD_REGISTER_CLASS, 19
 - JEOD_REGISTER_INCOMPLETE_CLASS, 19
 - JEOD_REGISTER_NONEXPORTED_CLASS, 20
- find
 - jeod::JeodMemoryTable, 96
- find_alloc_entry_atomic
 - jeod::JeodMemoryManager, 69
- Flags
 - jeod::JeodMemoryItem, 45
- flags
 - jeod::JeodMemoryItem, 52
- free_memory
 - jeod::JeodMemoryManager, 70
- generate_shutdown_report
 - jeod::JeodMemoryManager, 71
- get
 - jeod::JeodMemoryTable, 96
- get_alloc_id_atomic
 - jeod::JeodMemoryManager, 71
- get_alloc_index
 - jeod::JeodMemoryItem, 47
- get_attr
 - jeod::JeodMemoryTypeDescriptor, 109
- get_checkpointed
 - jeod::JeodMemoryItem, 48
- get_descriptor
 - jeod::JeodMemoryTypePreDescriptor, 121
 - jeod::JeodMemoryTypePreDescriptorDerived, 124
- get_descriptor_index
 - jeod::JeodMemoryItem, 48
- get_is_array
 - jeod::JeodMemoryItem, 48
- get_is_guarded
 - jeod::JeodMemoryItem, 49
- get_is_registered
 - jeod::JeodMemoryItem, 49
- get_name
 - jeod::JeodMemoryTypeDescriptor, 109
- get_nelems
 - jeod::JeodMemoryItem, 49
- get_placement_new
 - jeod::JeodMemoryItem, 50
- get_ref
 - jeod::JeodMemoryTypePreDescriptorDerived, 125
- get_register_instances
 - jeod::JeodMemoryTypeDescriptor, 110
- get_size
 - jeod::JeodMemoryTypeDescriptor, 110
- get_string_atomic
 - jeod::JeodMemoryManager, 72
- get_type_descriptor
 - jeod::JeodMemoryManager, 72, 73
- get_type_descriptor_atomic
 - jeod::JeodMemoryManager, 73, 74
- get_type_descriptor_nolock
 - jeod::JeodMemoryManager, 75
- get_type_entry_atomic
 - jeod::JeodMemoryManager, 75, 76
- get_type_index_nolock
 - jeod::JeodMemoryManager, 76
- get_typeid
 - jeod::JeodMemoryTypeDescriptor, 110
 - jeod::JeodMemoryTypePreDescriptor, 122

- jeod::JeodMemoryTypePreDescriptorDerived, [125](#)
- get_unique_id
 - jeod::JeodMemoryItem, [50](#)
- guard_enabled
 - jeod::JeodMemoryManager, [85](#)
- index
 - jeod::JeodMemoryManager::TypeEntry, [136](#)
- init_attrjeod__JeodMemoryManager
 - jeod::JeodMemoryManager, [84](#)
- init_attrjeod__MemoryMessages
 - jeod::MemoryMessages, [132](#)
- initialize_type_name
 - jeod::JeodMemoryTypeDescriptor, [111](#)
- InputProcessor
 - jeod::JeodMemoryManager, [84](#)
 - jeod::MemoryMessages, [132](#)
- Internal macros, [21](#)
 - JEOD_ALLOC_ARRAY_INTERNAL, [21](#)
 - JEOD_ALLOC_OBJECT_FILL, [22](#)
 - JEOD_ALLOC_OBJECT_INTERNAL, [22](#)
 - JEOD_ALLOC_POINTER_FILL, [22](#)
 - JEOD_ALLOC_PRIMITIVE_FILL, [22](#)
 - JEOD_CREATE_MEMORY, [23](#)
 - JEOD_DELETE_INTERNAL, [23](#)
- internal_error
 - jeod::MemoryMessages, [133](#)
- invalid_size
 - jeod::MemoryMessages, [133](#)
- is_allocated
 - jeod::JeodMemoryManager, [77](#)
- is_allocated_internal
 - jeod::JeodMemoryManager, [77](#)
- is_exportable
 - jeod::JeodMemoryTypePreDescriptorDerived, [126](#)
- is_structured
 - jeod::JeodMemoryTypeDescriptor, [111](#)
 - jeod::JeodMemoryTypeDescriptorDerived, [119](#)
- is_structured_data
 - jeod::JeodMemoryItem, [50](#)
- is_table_empty
 - jeod::JeodMemoryManager, [78](#)
- JEOD_ALLOC_ARRAY_INTERNAL
 - Internal macros, [21](#)
- JEOD_ALLOC_CLASS_ARRAY
 - Externally-usable macros, [12](#)
- JEOD_ALLOC_CLASS_MULTI_POINTER_ARRAY
 - Externally-usable macros, [13](#)
- JEOD_ALLOC_CLASS_OBJECT
 - Externally-usable macros, [13](#)
- JEOD_ALLOC_CLASS_POINTER_ARRAY
 - Externally-usable macros, [14](#)
- JEOD_ALLOC_OBJECT_FILL
 - Internal macros, [22](#)
- JEOD_ALLOC_OBJECT_INTERNAL
 - Internal macros, [22](#)
- JEOD_ALLOC_POINTER_FILL
 - Internal macros, [22](#)
- JEOD_ALLOC_PRIM_ARRAY
 - Externally-usable macros, [14](#)
- JEOD_ALLOC_PRIM_OBJECT
 - Externally-usable macros, [15](#)
- JEOD_ALLOC_PRIMITIVE_FILL
 - Internal macros, [22](#)
- JEOD_CREATE_MEMORY
 - Internal macros, [23](#)
- JEOD_DELETE_2D
 - Externally-usable macros, [15](#)
- JEOD_DELETE_ARRAY
 - Externally-usable macros, [16](#)
- JEOD_DELETE_INTERNAL
 - Internal macros, [23](#)
- JEOD_DELETE_OBJECT
 - Externally-usable macros, [17](#)
- JEOD_DEREGISTER_CHECKPOINTABLE
 - Externally-usable macros, [17](#)
- JEOD_IS_ALLOCATED
 - Externally-usable macros, [18](#)
- JEOD_MEMORY_DEBUG
 - Externally-usable macros, [18](#)
- JEOD_REGISTER_CHECKPOINTABLE
 - Externally-usable macros, [18](#)
- JEOD_REGISTER_CLASS
 - Externally-usable macros, [19](#)
- JEOD_REGISTER_INCOMPLETE_CLASS
 - Externally-usable macros, [19](#)
- JEOD_REGISTER_NONEXPORTED_CLASS
 - Externally-usable macros, [20](#)
- jeod, [31](#)
 - AllocTable, [32](#)
 - jeod_alloc_construct_array, [33](#)
 - jeod_alloc_destruct_array, [33](#)
 - jeod_alloc_get_allocated_pointer, [34](#)
 - MAGIC0, [35](#)
 - MAGIC1, [35](#)
 - TypeTable, [33](#)
- jeod::JeodAllocHelperAllocatedPointer
 - cast, [38](#)
- jeod::JeodAllocHelperAllocatedPointer< T, is_poly >, [37](#)
- jeod::JeodAllocHelperAllocatedPointer< T, true >, [38](#)
 - cast, [39](#)
- jeod::JeodAllocHelperConstructDestruct
 - construct, [40](#)
 - destruct, [41](#)
- jeod::JeodAllocHelperConstructDestruct< T, false, is_↔ >, [41](#)
 - abstract >, [41](#)
 - construct, [42](#)
 - destruct, [42](#)
- jeod::JeodAllocHelperConstructDestruct< T, is_class, is_abstract >, [39](#)
- jeod::JeodAllocHelperConstructDestruct< T, true, false >, [43](#)
 - construct, [43](#)
 - destruct, [43](#)
- jeod::JeodMemoryItem, [44](#)

- ~JeodMemoryItem, 47
- alloc_info_index, 52
- construct_flags, 47
- descriptor_index_hi, 52
- descriptor_index_lo, 52
- Flags, 45
- flags, 52
- get_alloc_index, 47
- get_checkpointed, 48
- get_descriptor_index, 48
- get_is_array, 48
- get_is_guarded, 49
- get_is_registered, 49
- get_nelems, 49
- get_placement_new, 50
- get_unique_id, 50
- is_structured_data, 50
- JeodMemoryItem, 46
- nelems, 53
- set_is_registered, 51
- set_unique_id, 51
- unique_id, 53
- jeod::JeodMemoryManager, 54
 - ~JeodMemoryManager, 61
 - add_allocation_atomic, 61
 - add_string_atomic, 62
 - alloc_table, 84
 - AllocTable, 59
 - allocate_memory, 63
 - allocation_number, 84
 - begin_atomic_block, 63
 - check_master, 64
 - create_memory, 65
 - create_memory_internal, 66
 - cur_data_size, 85
 - debug_level, 85
 - DebugLevel, 60
 - delete_oldest_alloc_entry_atomic, 66
 - deregister_container, 67
 - destroy_memory, 68
 - destroy_memory_internal, 68
 - end_atomic_block, 69
 - find_alloc_entry_atomic, 69
 - free_memory, 70
 - generate_shutdown_report, 71
 - get_alloc_id_atomic, 71
 - get_string_atomic, 72
 - get_type_descriptor, 72, 73
 - get_type_descriptor_atomic, 73, 74
 - get_type_descriptor_nolock, 75
 - get_type_entry_atomic, 75, 76
 - get_type_index_nolock, 76
 - guard_enabled, 85
 - init_attrjeod__JeodMemoryManager, 84
 - InputProcessor, 84
 - is_allocated, 77
 - is_allocated_internal, 77
 - is_table_empty, 78
 - JeodMemoryManager, 60, 61
 - Master, 86
 - max_data_size, 86
 - max_table_size, 86
 - mode, 86
 - mutex, 87
 - NameType, 60
 - operator=, 78
 - register_class, 78
 - register_container, 79
 - register_memory_internal, 80
 - reset_alloc_id_atomic, 80
 - restart_clear_memory, 81
 - restart_reallocate, 81
 - set_debug_level, 82
 - set_guard_enabled, 82
 - set_mode, 83
 - set_mode_internal, 83
 - sim_interface, 87
 - string_table, 87
 - type_table, 87
 - TypeTable, 59
- jeod::JeodMemoryManager::TypeEntry, 135
 - index, 136
 - tdesc, 136
 - TypeEntry, 135
- jeod::JeodMemoryReflectiveTable, 88
 - add, 89
 - JeodMemoryReflectiveTable, 89
 - operator=, 90
- jeod::JeodMemoryTable
 - ~JeodMemoryTable, 93
 - add, 94
 - begin, 94
 - clone, 94
 - const_value_iterator, 92
 - del, 95
 - end, 95
 - find, 96
 - get, 96
 - JeodMemoryTable, 93
 - NameIndex, 92
 - operator=, 97
 - string_to_index, 97
 - value_list, 97
 - ValueList, 93
- jeod::JeodMemoryTable< ValueType >, 90
- jeod::JeodMemoryTableClonable
 - clone, 99
 - JeodMemoryTableClonable, 98, 99
 - operator=, 99
- jeod::JeodMemoryTableClonable< ValueType >, 98
- jeod::JeodMemoryTableCopyable
 - clone, 101
 - JeodMemoryTableCopyable, 100, 101
 - operator=, 101
- jeod::JeodMemoryTableCopyable< ValueType >, 100
- jeod::JeodMemoryTypeDescriptor, 102

- ~JeodMemoryTypeDescriptor, 104
- base_type, 105
- buffer_end, 105
- buffer_size, 106
- check_for_registration_errors, 113
- clone, 107
- construct_array, 107
- delete_array, 107
- delete_object, 108
- destroy_memory, 108
- destruct_array, 109
- dimensionality, 109
- get_attr, 109
- get_name, 109
- get_register_instances, 110
- get_size, 110
- get_typeid, 110
- initialize_type_name, 111
- is_structured, 111
- JeodMemoryTypeDescriptor, 104
- most_derived_pointer, 111, 112
- name, 113
- obj_id, 114
- operator=, 112
- pointer_dimension, 112
- register_instances, 114
- set_check_for_registration_errors, 112
- size, 114
- type_spec, 113
- jeod::JeodMemoryTypeDescriptor::attr, 37
- jeod::JeodMemoryTypeDescriptorDerived
 - ~JeodMemoryTypeDescriptorDerived, 117
 - Attributes, 116
 - clone, 117
 - construct_array, 118
 - delete_array, 118
 - delete_object, 118
 - destruct_array, 119
 - is_structured, 119
 - JeodMemoryTypeDescriptorDerived, 116, 117
 - most_derived_pointer, 119, 120
 - operator=, 120
 - TypeDescriptor, 116
- jeod::JeodMemoryTypeDescriptorDerived< Type >, 115
- jeod::JeodMemoryTypePreDescriptor, 121
 - ~JeodMemoryTypePreDescriptor, 121
 - get_descriptor, 121
 - get_typeid, 122
- jeod::JeodMemoryTypePreDescriptorDerived
 - ~JeodMemoryTypePreDescriptorDerived, 124
 - descriptor, 126
 - get_descriptor, 124
 - get_ref, 125
 - get_typeid, 125
 - is_exportable, 126
 - JeodMemoryTypePreDescriptorDerived, 124
 - TypeDescriptor, 123
- jeod::JeodMemoryTypePreDescriptorDerived< Type >, 122
- jeod::JeodSimEngineAttributes
 - attributes, 127
- jeod::JeodSimEngineAttributes< Type *, false >, 127
 - attributes, 128
- jeod::JeodSimEngineAttributes< Type, is_class >, 126
- jeod::JeodSimEngineAttributes< Type, true >, 128
 - attributes, 129
- jeod::JeodSimEngineAttributes< void *, false >, 129
 - attributes, 130
- jeod::MemoryMessages, 130
 - corrupted_memory, 132
 - debug, 132
 - init_attrjeod__MemoryMessages, 132
 - InputProcessor, 132
 - internal_error, 133
 - invalid_size, 133
 - lock_error, 133
 - MemoryMessages, 131
 - null_pointer, 133
 - operator=, 132
 - out_of_memory, 134
 - registration_error, 134
 - singleton_error, 134
 - suspect_pointer, 134
- jeod_alloc.hh, 137
- jeod_alloc_construct_array
 - jeod, 33
- jeod_alloc_construct_destruct.hh, 139
- jeod_alloc_destruct_array
 - jeod, 33
- jeod_alloc_get_allocated_pointer
 - jeod, 34
- jeod_alloc_get_allocated_pointer.hh, 140
- JeodMemoryItem
 - jeod::JeodMemoryItem, 46
- JeodMemoryManager
 - jeod::JeodMemoryManager, 60, 61
- JeodMemoryReflectiveTable
 - jeod::JeodMemoryReflectiveTable, 89
- JeodMemoryTable
 - jeod::JeodMemoryTable, 93
- JeodMemoryTableClonable
 - jeod::JeodMemoryTableClonable, 98, 99
- JeodMemoryTableCopyable
 - jeod::JeodMemoryTableCopyable, 100, 101
- JeodMemoryTypeDescriptor
 - jeod::JeodMemoryTypeDescriptor, 104
- JeodMemoryTypeDescriptorDerived
 - jeod::JeodMemoryTypeDescriptorDerived, 116, 117
- JeodMemoryTypePreDescriptorDerived
 - jeod::JeodMemoryTypePreDescriptorDerived, 124
- lock_error
 - jeod::MemoryMessages, 133
- MAGIC0

- jeod, 35
- MAGIC1
 - jeod, 35
- MAKE_DESCRIPTOR
 - memory_manager.cc, 143
- MAKE_MEMORY_MESSAGE_CODE
 - memory_messages.cc, 147
- Master
 - jeod::JeodMemoryManager, 86
- max_data_size
 - jeod::JeodMemoryManager, 86
- max_table_size
 - jeod::JeodMemoryManager, 86
- Memory, 28
- memory_attributes_templates.hh, 141
- memory_item.cc, 142
- memory_item.hh, 142
- memory_manager.cc, 143
 - MAKE_DESCRIPTOR, 143
- memory_manager.hh, 144
- memory_manager_hide_from_trick.hh, 145
- memory_manager_protected.cc, 145
- memory_manager_static.cc, 146
- memory_messages.cc, 146
 - MAKE_MEMORY_MESSAGE_CODE, 147
- memory_messages.hh, 147
- memory_table.hh, 148
- memory_type.cc, 148
- memory_type.hh, 149
- MemoryMessages
 - jeod::MemoryMessages, 131
- mode
 - jeod::JeodMemoryManager, 86
- Models, 26
- most_derived_pointer
 - jeod::JeodMemoryTypeDescriptor, 111, 112
 - jeod::JeodMemoryTypeDescriptorDerived, 119, 120
- mutex
 - jeod::JeodMemoryManager, 87
- name
 - jeod::JeodMemoryTypeDescriptor, 113
- NameIndex
 - jeod::JeodMemoryTable, 92
- NameType
 - jeod::JeodMemoryManager, 60
- nelems
 - jeod::JeodMemoryItem, 53
- null_pointer
 - jeod::MemoryMessages, 133
- obj_id
 - jeod::JeodMemoryTypeDescriptor, 114
- operator=
 - jeod::JeodMemoryManager, 78
 - jeod::JeodMemoryReflectiveTable, 90
 - jeod::JeodMemoryTable, 97
 - jeod::JeodMemoryTableClonable, 99
 - jeod::JeodMemoryTableCopyable, 101
 - jeod::JeodMemoryTypeDescriptor, 112
 - jeod::JeodMemoryTypeDescriptorDerived, 120
 - jeod::MemoryMessages, 132
- out_of_memory
 - jeod::MemoryMessages, 134
- pointer_dimension
 - jeod::JeodMemoryTypeDescriptor, 112
- register_class
 - jeod::JeodMemoryManager, 78
- register_container
 - jeod::JeodMemoryManager, 79
- register_instances
 - jeod::JeodMemoryTypeDescriptor, 114
- register_memory_internal
 - jeod::JeodMemoryManager, 80
- registration_error
 - jeod::MemoryMessages, 134
- reset_alloc_id_atomic
 - jeod::JeodMemoryManager, 80
- restart_clear_memory
 - jeod::JeodMemoryManager, 81
- restart_reallocate
 - jeod::JeodMemoryManager, 81
- set_check_for_registration_errors
 - jeod::JeodMemoryTypeDescriptor, 112
- set_debug_level
 - jeod::JeodMemoryManager, 82
- set_guard_enabled
 - jeod::JeodMemoryManager, 82
- set_is_registered
 - jeod::JeodMemoryItem, 51
- set_mode
 - jeod::JeodMemoryManager, 83
- set_mode_internal
 - jeod::JeodMemoryManager, 83
- set_unique_id
 - jeod::JeodMemoryItem, 51
- sim_interface
 - jeod::JeodMemoryManager, 87
- singleton_error
 - jeod::MemoryMessages, 134
- size
 - jeod::JeodMemoryTypeDescriptor, 114
- string_table
 - jeod::JeodMemoryManager, 87
- string_to_index
 - jeod::JeodMemoryTable, 97
- Support classes, 25
 - __STDC_LIMIT_MACROS, 25
- suspect_pointer
 - jeod::MemoryMessages, 134
- tdesc
 - jeod::JeodMemoryManager::TypeEntry, 136
- type_spec

- jeod::JeodMemoryTypeDescriptor, [113](#)
- type_table
 - jeod::JeodMemoryManager, [87](#)
- TypeDescriptor
 - jeod::JeodMemoryTypeDescriptorDerived, [116](#)
 - jeod::JeodMemoryTypePreDescriptorDerived, [123](#)
- TypeEntry
 - jeod::JeodMemoryManager::TypeEntry, [135](#)
- TypeTable
 - jeod, [33](#)
 - jeod::JeodMemoryManager, [59](#)
- unique_id
 - jeod::JeodMemoryItem, [53](#)
- Utils, [27](#)
- value_list
 - jeod::JeodMemoryTable, [97](#)
- ValueList
 - jeod::JeodMemoryTable, [93](#)