

---

# Freescal<sup>e</sup> MQX™ RTOS Reference Manual

MQXRM  
Rev. 19  
04/2015



**How to Reach Us:****Home Page:**

freescale.com

**Web Support:**

freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: [freescale.com/SalesTermsandConditions](http://freescale.com/SalesTermsandConditions).

Freescale, the Freescale logo, and Kinetis, are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Vybrid is a trademark of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. ARM, ARM Powered logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.  
© 2009-2015 Freescale Semiconductor, Inc.



## Revision History

To provide the most up-to-date information, the revision of our documents on the World Wide Web is the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to [freescale.com](http://freescale.com) and navigate to Design Resources>Software and Tools>All Software and Tools>Freescale MQX Software Solutions.

The following revision history table summarizes changes contained in this document.

Revision Number	Revision Date	Description of Changes
Rev. 0	01/2009	Initial Release coming with MQX RTOS version 3.0
Rev. 1	05/2009	Minor formatting updates.
Rev. 2	09/2009	Formatting significantly updated for MQX RTOS version 3.4 Release. Autoclear feature of <code>lwevent</code> described ( <code>_lwevent_set_auto_clear</code> ).
Rev. 3	01/2010	Update coming with MQX RTOS version 3.5. <code>_lwevent_wait_description</code> updated.
Rev. 4	08/2010	Update of <code>_lwevent_xxx</code> & <code>_time_diff_xxx</code> sections.
Rev. 5	11/2010	Update of the following sections: <code>_time_diff_ticks</code> <code>_task_create_xxx</code> <code>_task_get_template_ptr</code> <code>_mem_alloc</code> <code>_lwmem_alloc</code> <code>_lwevent_get_signalled</code> (chapter added)
Rev. 6	04/2011	Update of <code>_time_get_ticks_per_sec</code> , <code>_lwmsgq_init</code> and <code>_sem_open</code> and <code>_time_delay</code> sections.
Rev. 7	12/2011	<code>_ipc_task</code> function and <code>IPC_PROTOCOL_INIT_STRUCT</code> description updated. <code>IPC_INIT_STRUCT</code> description added. <code>_lwevent_wait...</code> and <code>_mqx_exit</code> chapters updated. "Function Listing Format" section updated by the User Mode specific function parameter categories. <code>_mem_set_pool_access</code> , <code>_usr_lwevent_*</code> , <code>_usr_lwmem_*</code> , <code>_usr_lwsem_*</code> , <code>_usr_task_*</code> , <code>_usr_time_*</code> sections added.
Rev. 8	06/2012	<code>_lwtimer_add_timer_to_queue()</code> , <code>_msgq_send_queue</code> and <code>_task_stop_preemption()</code> function descriptions updated.
Rev. 9	12/2012	Update reflecting changes in the MQX RTOS code and the source tree structure (paths, prototypes, file names, etc.)

Revision Number	Revision Date	Description of Changes
Rev. 10	04/2013	<p>_lwmsgq_send and _lwmsgq_receive function descriptions updated.</p> <p>Cautions added to all _DCACHE_xxx sections.</p> <p>Task Error Codes of _msgq_send_broadcast, _lwmem_alloc, _lwmem_alloc_*_from and _mem_alloc sections updated.</p> <p>Error codes of _mutex_destroy, _lwmem_free and _mem_test functions updated.</p> <p>Return values of the _mutatr_init, _mem_extend_pool and _mem_extend functions updated.</p> <p>_lwmem_alloc and _lwmem_alloc_*_from sections updated.</p>
Rev. 11	05/2013	<p>Added description of following functions:</p> <p>_mem_alloc_system_align</p> <p>_mem_alloc_system_align_from</p> <p>_lwmem_alloc_system_align</p> <p>_lwmem_alloc_system_align_from</p>
Rev. 12	06/2013	Made corrections to various _mmu_ functions.
Rev. 13	08/2013	Added _psp_push_fp_context and _psp_pop_fp_context.
Rev. 14	10/2013	Updated _lwmem_alloc_*_from, _mem_alloc ..., and _time_get_resolution, _time_set_resolution
Rev. 15	01/2014	Added C99 types.
Rev.16	02/2014	4.1.0 updates.
Rev. 17	07/2014	Kinetis SDK -related updates
Rev. 18	12/2014	Updates for KSDK 1.1.0
Rev. 19	04/2015	Updated for KSDK 1.2.0

© Freescale Semiconductor, Inc., 2008-2014. All rights reserved.

## Chapter 1

### Before You Begin

1.1	About MQX RTOS	15
1.2	About This Book	15
1.3	Function Listing Format	15
1.4	Conventions	17
1.4.1	Tips	17
1.4.2	Notes	17
1.4.3	Cautions	17
1.4.4	MQX RTOS directory <MQX_KSDK_DIR>	17

## Chapter 2

### MQX RTOS Functions and Macros

2.1	MQX RTOS Function Overview	19
2.1.1	_DCACHE_DISABLE	21
2.1.2	_DCACHE_ENABLE	22
2.1.3	_DCACHE_FLUSH	23
2.1.4	_DCACHE_FLUSH_LINE	24
2.1.5	_DCACHE_FLUSH_MLINES	25
2.1.6	_DCACHE_INVALIDATE	26
2.1.7	_DCACHE_INVALIDATE_LINE	27
2.1.8	_DCACHE_INVALIDATE_MLINES	28
2.1.9	_event_clear	29
2.1.10	_event_close	31
2.1.11	_event_create, _event_create_auto_clear	32
2.1.12	_event_create_component	34
2.1.13	_event_create_fast, _event_create_fast_auto_clear	36
2.1.14	_event_destroy	38
2.1.15	_event_destroy_fast	40
2.1.16	_event_get_value	41
2.1.17	_event_get_wait_count	43
2.1.18	_event_open	44
2.1.19	_event_open_fast	46
2.1.20	_event_set	47
2.1.21	_event_test	49
2.1.22	_event_wait_all ...	50
2.1.23	_event_wait_any ...	52
2.1.24	_ICACHE_DISABLE	54
2.1.25	_ICACHE_ENABLE	55
2.1.26	_ICACHE_INVALIDATE	56
2.1.27	_ICACHE_INVALIDATE_LINE	57
2.1.28	_ICACHE_INVALIDATE_MLINES	58
2.1.29	_int_default_isr	59
2.1.30	_int_disable, _int_enable	60

2.1.31 _int_exception_isr	61
2.1.32 _int_get_default_isr	62
2.1.33 _int_get_exception_handler	63
2.1.34 _int_get_isr	64
2.1.35 _int_get_isr_data	65
2.1.36 _int_get_isr_depth	66
2.1.37 _int_get_kernel_isr	67
2.1.38 _int_get_previous_vector_table	68
2.1.39 _int_get_vector_table	69
2.1.40 _int_install_default_isr	70
2.1.41 _int_install_exception_isr	71
2.1.42 _int_install_isr	72
2.1.43 _int_install_kernel_isr	74
2.1.44 _int_install_unexpected_isr	75
2.1.45 _int_kernel_isr	76
2.1.46 _int_set_exception_handler	77
2.1.47 _int_set_isr_data	78
2.1.48 _int_set_vector_table	79
2.1.49 _int_unexpected_isr	80
2.1.50 _psp_push_fp_context	81
2.1.51 _psp_pop_fp_context	82
2.1.52 _ipc_add_io_ipc_handler	83
2.1.53 _ipc_add_ipc_handler	84
2.1.54 _ipc_msg_processor_route_exists	85
2.1.55 _ipc_msg_route_add	86
2.1.56 _ipc_msg_route_remove	87
2.1.57 _ipc_pcb_init	88
2.1.58 _ipc_task	90
2.1.59 _klog_control	91
2.1.60 _klog_create, _klog_create_at	93
2.1.61 _klog_disable_logging_task, _klog_enable_logging_task	95
2.1.62 _klog_display	96
2.1.63 _klog_get_interrupt_stack_usage	97
2.1.64 _klog_get_task_stack_usage	99
2.1.65 _klog_show_stack_usage	100
2.1.66 _log_create	101
2.1.67 _log_create_component	103
2.1.68 _log_destroy	104
2.1.69 _log_disable, _log_enable	105
2.1.70 _log_read	106
2.1.71 _log_reset	108
2.1.72 _log_test	109
2.1.73 _log_write	110
2.1.74 _lwevent_clear	111
2.1.75 _lwevent_create	112

2.1.76	<code>_lwevent_destroy</code>	113
2.1.77	<code>_lwevent_get_signalled</code>	114
2.1.78	<code>_lwevent_set</code>	116
2.1.79	<code>_lwevent_set_auto_clear</code>	117
2.1.80	<code>_lwevent_test</code>	118
2.1.81	<code>_lwevent_wait</code> ...	119
2.1.82	<code>_lwlog_calculate_size</code>	121
2.1.83	<code>_lwlog_create</code> , <code>_lwlog_create_at</code>	122
2.1.84	<code>_lwlog_create_component</code>	124
2.1.85	<code>_lwlog_destroy</code>	125
2.1.86	<code>_lwlog_disable</code> , <code>_lwlog_enable</code>	126
2.1.87	<code>_lwlog_read</code>	127
2.1.88	<code>_lwlog_reset</code>	128
2.1.89	<code>_lwlog_test</code>	129
2.1.90	<code>_lwlog_write</code>	130
2.1.91	<code>_lwmem_alloc</code> ...	131
2.1.92	<code>_lwmem_alloc_*_from</code>	134
2.1.93	<code>_lwmem_create_pool</code>	136
2.1.94	<code>_lwmem_free</code>	137
2.1.95	<code>_lwmem_get_size</code>	139
2.1.96	<code>_lwmem_set_default_pool</code>	140
2.1.97	<code>_lwmem_test</code>	141
2.1.98	<code>_lwmem_transfer</code>	142
2.1.99	<code>_lwmsgq_deinit</code>	143
2.1.100	<code>_lwmsgq_init</code>	144
2.1.101	<code>_lwmsgq_receive</code>	145
2.1.102	<code>_lwmsgq_send</code>	147
2.1.103	<code>_lwsem_create</code>	148
2.1.104	<code>_lwsem_destroy</code>	150
2.1.105	<code>_lwsem_poll</code>	151
2.1.106	<code>_lwsem_post</code>	152
2.1.107	<code>_lwsem_test</code>	153
2.1.108	<code>_lwsem_wait</code> ...	154
2.1.109	<code>_lwtimer_add_timer_to_queue</code>	156
2.1.110	<code>_lwtimer_cancel_period</code>	158
2.1.111	<code>_lwtimer_cancel_timer</code>	159
2.1.112	<code>_lwtimer_create_periodic_queue</code>	160
2.1.113	<code>_lwtimer_test</code>	161
2.1.114	<code>_mem_alloc</code> ...	162
2.1.115	<code>_mem_copy</code>	166
2.1.116	<code>_mem_create_pool</code>	167
2.1.117	<code>_mem_extend</code>	168
2.1.118	<code>_mem_extend_pool</code>	169
2.1.119	<code>_mem_free</code>	170
2.1.120	<code>_mem_free_part</code>	172

2.1.121_mem_get_error	173
2.1.122_mem_get_error_pool	175
2.1.123_mem_get_highwater	176
2.1.124_mem_get_highwater_pool	177
2.1.125_mem_get_size	178
2.1.126_mem_set_pool_access	180
2.1.127_mem_sum_ip	181
2.1.128_mem_swap_endian	182
2.1.129_mem_test	183
2.1.130_mem_test_all	184
2.1.131_mem_test_and_set	185
2.1.132_mem_test_pool	186
2.1.133_mem_transfer	187
2.1.134_mem_zero	188
2.1.135_mmu_add_vregion	189
2.1.136_mmu_vdisable	190
2.1.137_mmu_venable	191
2.1.138_mmu_vinit	192
2.1.139_mmu_vtop	193
2.1.140_mqx	194
2.1.141_mqx_bsp_revision	196
2.1.142_mqx_copyright	197
2.1.143_mqx_date	198
2.1.144_mqx_exit	199
2.1.145_mqx_fatal_error	200
2.1.146_mqx_generic_revision	201
2.1.147_mqx_get_counter	202
2.1.148_mqx_get_exit_handler	203
2.1.149_mqx_get_initialization	204
2.1.150_mqx_get_kernel_data	205
2.1.151_mqx_get_system_task_id	206
2.1.152_mqx_get_tad_data, _mqx_set_tad_data	207
2.1.153_mqx_idle_task	208
2.1.154_mqx_monitor_type	209
2.1.155_mqx_psp_revision	210
2.1.156_mqx_set_context_switch_handler	211
2.1.157_mqx_set_exit_handler	212
2.1.158_mqx_version	213
2.1.159_mqx_zero_tick_struct	214
2.1.160_msg_alloc	215
2.1.161_msg_alloc_system	217
2.1.162_msg_available	218
2.1.163_msg_create_component	219
2.1.163.1Example	220
2.1.164_msg_free	221



2.1.165_msg_swap_endian_data .....	222
2.1.166_msg_swap_endian_header .....	224
2.1.167_msgpool_create .....	225
2.1.168_msgpool_create_system .....	227
2.1.169_msgpool_destroy .....	228
2.1.170_msgpool_test .....	229
2.1.171_msgq_close .....	230
2.1.172_msgq_get_count .....	232
2.1.173_msgq_get_id .....	233
2.1.174_msgq_get_notification_function .....	234
2.1.175_msgq_get_owner .....	235
2.1.176_msgq_open .....	236
2.1.177_msgq_open_system .....	238
2.1.178_msgq_peek .....	240
2.1.179_msgq_poll .....	241
2.1.180_msgq_receive .....	243
2.1.181_msgq_send .....	246
2.1.182_msgq_send_broadcast .....	249
2.1.183_msgq_send_priority .....	251
2.1.184_msgq_send_queue .....	253
2.1.185_msgq_send_urgent .....	255
2.1.186_msgq_set_notification_function .....	256
2.1.187_msgq_test .....	258
2.1.188_mutatr_destroy .....	259
2.1.189_mutatr_get_priority_ceiling, _mutatr_set_priority_ceiling .....	260
2.1.190_mutatr_get_sched_protocol, _mutatr_set_sched_protocol .....	262
2.1.191_mutatr_get_spin_limit, _mutatr_set_spin_limit .....	264
2.1.192_mutatr_get_wait_protocol, _mutatr_set_wait_protocol .....	266
2.1.192.1Waiting protocols .....	267
2.1.193_mutatr_init .....	268
2.1.194_mutex_create_component .....	269
2.1.195_mutex_destroy .....	270
2.1.196_mutex_get_priority_ceiling, _mutex_set_priority_ceiling .....	271
2.1.197_mutex_get_wait_count .....	273
2.1.198_mutex_init .....	274
2.1.199_mutex_lock .....	275
2.1.200_mutex_test .....	277
2.1.201_mutex_try_lock .....	279
2.1.202_mutex_unlock .....	280
2.1.203_name_add .....	281
2.1.204_name_create_component .....	282
2.1.205_name_delete .....	284
2.1.206_name_find .....	285
2.1.207_name_find_by_number .....	286
2.1.208_name_test .....	287

2.1.209_partition_alloc, _partition_alloc_zero	289
2.1.210_partition_alloc_system, _partition_alloc_system_zero	291
2.1.211_partition_calculate_blocks	292
2.1.212_partition_calculate_size	293
2.1.213_partition_create	294
2.1.214_partition_create_at	296
2.1.215_partition_create_component	298
2.1.216_partition_destroy	299
2.1.217_partition_extend	300
2.1.218_partition_free	301
2.1.219_partition_get_block_size	302
2.1.220_partition_get_free_blocks	304
2.1.221_partition_get_max_used_blocks	305
2.1.222_partition_get_total_blocks	306
2.1.223_partition_get_total_size	307
2.1.224_partition_test	308
2.1.225_partition_transfer	309
2.1.226_queue_dequeue	310
2.1.227_queue_enqueue	312
2.1.228_queue_get_size	313
2.1.229_queue_head	314
2.1.230_queue_init	315
2.1.231_queue_insert	316
2.1.232_queue_is_empty	318
2.1.233_queue_next	319
2.1.234_queue_test	320
2.1.235_queue_unlink	321
2.1.236_sched_get_max_priority	322
2.1.237_sched_get_min_priority	323
2.1.238_sched_get_policy	324
2.1.239_sched_get_rr_interval, _sched_get_rr_interval_ticks	325
2.1.240_sched_set_policy	326
2.1.241_sched_set_rr_interval, _sched_set_rr_interval_ticks	327
2.1.242_sched_yield	328
2.1.243_sem_close	329
2.1.244_sem_create	330
2.1.245_sem_create_component	332
2.1.246_sem_create_fast	334
2.1.247_sem_destroy, _sem_destroy_fast	335
2.1.248_sem_get_value	337
2.1.249_sem_get_wait_count	338
2.1.250_sem_open, _sem_open_fast	339
2.1.251_sem_post	341
2.1.252_sem_test	343
2.1.253_sem_wait ...	344

2.1.254_str_mqx_uint_to_hex_string .....	346
2.1.255_strnlen .....	347
2.1.256_task_abort .....	348
2.1.257_task_block .....	350
2.1.258_task_check_stack .....	351
2.1.259_task_create, _task_create_blocked, _task_create_at, create_task .....	352
2.1.260_task_destroy .....	354
2.1.261_task_disable_fp, _task_enable_fp .....	356
2.1.262_task_errno .....	357
2.1.263_task_get_creator .....	358
2.1.264_task_get_environment, _task_set_environment .....	359
2.1.265_task_get_error, _task_get_error_ptr .....	360
2.1.266_task_get_exception_handler, _task_set_exception_handler .....	361
2.1.267_task_get_exit_handler, _task_set_exit_handler .....	362
2.1.268_task_get_id .....	363
2.1.269_task_get_id_from_name .....	364
2.1.270_task_get_index_from_id .....	365
2.1.271_task_get_parameter ..., _task_set_parameter ... ..	366
2.1.272_task_get_priority, _task_set_priority .....	367
2.1.273_task_get_processor .....	369
2.1.274_task_get_td .....	370
2.1.275_task_get_template_index .....	371
2.1.276_task_get_template_ptr .....	372
2.1.277_task_ready .....	373
2.1.278_task_restart .....	375
2.1.279_task_set_error .....	376
2.1.280_task_start_preemption, _task_stop_preemption .....	377
2.1.281_taskq_create .....	378
2.1.282_taskq_destroy .....	380
2.1.283_taskq_get_value .....	381
2.1.284_taskq_resume .....	382
2.1.285_taskq_suspend .....	384
2.1.286_taskq_suspend_task .....	385
2.1.287_taskq_test .....	387
2.1.288_ticks_to_time .....	388
2.1.289_time_add ... ..	389
2.1.290_time_delay ... ..	391
2.1.291_time_dequeue .....	392
2.1.292_time_dequeue_td .....	394
2.1.293_time_diff, _time_diff_ticks .....	395
2.1.294_time_diff_ ... ..	397
2.1.295_time_from_date .....	400
2.1.296_time_get, _time_get_ticks .....	402
2.1.297_time_get_elapsed, _time_get_elapsed_ticks .....	403
2.1.298_time_get_hwticks .....	404

2.1.299_time_get_hwticks_per_tick, _time_set_hwticks_per_tick	405
2.1.300_time_get_microseconds	406
2.1.301_time_get_nanoseconds	407
2.1.302_time_get_resolution, _time_set_resolution	408
2.1.303_time_get_ticks_per_sec, _time_set_ticks_per_sec	409
2.1.304_time_init_ticks	410
2.1.305_time_notify_kernel	411
2.1.306_time_set, _time_set_ticks	412
2.1.307_time_set_timer_vector	413
2.1.308_time_to_date	414
2.1.309_time_to_ticks	415
2.1.310mktime	416
2.1.311gmtime_r	417
2.1.312timegm	418
2.1.313localtime_r	419
2.1.314_timer_cancel	420
2.1.315_timer_create_component	421
2.1.316_timer_start_oneshot_after	424
2.1.317_timer_start_oneshot_at	426
2.1.318_timer_start_periodic_at	428
2.1.319_timer_start_periodic_every	430
2.1.320_timer_test	432
2.1.321_usr_lwevent_clear	433
2.1.322_usr_lwevent_create	434
2.1.323_usr_lwevent_destroy	435
2.1.324_usr_lwevent_get_signalled	436
2.1.325_usr_lwevent_set	437
2.1.326_usr_lwevent_set_auto_clear	438
2.1.327_usr_lwevent_wait	439
2.1.328_usr_lwmem_alloc	441
2.1.329_usr_lwmem_alloc_from	442
2.1.330_usr_lwmem_create_pool	443
2.1.331_usr_lwmem_free	444
2.1.332_usr_lwmsgq_init	445
2.1.333_usr_lwmsgq_receive	446
2.1.334_usr_lwmsgq_send	447
2.1.335_usr_lwsem_create	448
2.1.336_usr_lwsem_destroy	449
2.1.337_usr_lwsem_poll	450
2.1.338_usr_lwsem_post	451
2.1.339_usr_lwsem_wait	452
2.1.340_usr_task_abort	454
2.1.341_usr_task_create	455
2.1.342_usr_task_destroy	456
2.1.343_usr_task_get_td	457

2.1.344_usr_task_ready .....	458
2.1.345_usr_task_set_error .....	459
2.1.346_usr_time_delay .....	460
2.1.347_usr_time_get_elapsed_ticks .....	461
2.1.348_watchdog_create_component .....	462
2.1.349_watchdog_start, _watchdog_start_ticks .....	464
2.1.350_watchdog_stop .....	465
2.1.351_watchdog_test .....	466
2.1.352MSG_MUST_CONVERT_DATA_ENDIAN .....	467
2.1.353MSG_MUST_CONVERT_HDR_ENDIAN .....	468

## Chapter 3

### MQX Data Types

3.1 Data Types Overview .....	469
3.2 MQX RTOS Complex Data Types in Alphabetical Order .....	471
3.2.1 DATE_STRUCT .....	471
3.2.2 IPC_PCB_INIT_STRUCT .....	473
3.2.3 IPC_PROTOCOL_INIT_STRUCT .....	474
3.2.4 IPC_ROUTING_STRUCT .....	475
3.2.5 IPC_INIT_STRUCT .....	476
3.2.6 LOG_ENTRY_STRUCT .....	477
3.2.7 LWEVENT_STRUCT .....	478
3.2.8 LWLOG_ENTRY_STRUCT .....	479
3.2.9 LWSEM_STRUCT .....	480
3.2.10 LWTIMER_PERIOD_STRUCT .....	481
3.2.11 LWTIMER_STRUCT .....	482
3.2.12 MESSAGE_HEADER_STRUCT .....	483
3.2.13 MQX_INITIALIZATION_STRUCT .....	485
3.2.14 MQX_TICK_STRUCT .....	487
3.2.15 MUTEX_ATTR_STRUCT .....	488
3.2.16 MUTEX_STRUCT .....	489
3.2.17 QUEUE_ELEMENT_STRUCT .....	491
3.2.18 QUEUE_STRUCT .....	492
3.2.19 TASK_TEMPLATE_STRUCT .....	493
3.2.20 TIME_STRUCT .....	496
3.2.21 time_t .....	497
3.2.22 TM_STRUCT .....	498



# Chapter 1 Before You Begin

## 1.1 About MQX RTOS

The MQX™ Real-Time Operating System has been designed for uni-processor, multi-processor, and distributed-processor embedded real-time systems.

MQX RTOS is a runtime library of functions that programs use to become real-time multi-tasking applications. The main features are its scalable size, component-oriented architecture, and ease of use.

MQX RTOS supports multi-processor applications and can be used with flexible embedded input/output products for networking, data communications, and file management.

## 1.2 About This Book

This book contains alphabetical listings of MQX RTOS function prototypes and alphabetical listings of data type definitions.

Use this book in conjunction with the appropriate MQX RTOS User's Guide which covers the following general topics:

- MQX RTOS at a glance
- Using MQX RTOS
- Rebuilding MQX RTOS
- Developing a new BSP
- Frequently asked questions

## 1.3 Function Listing Format

This is the general format for listing a function or a data type.

**function\_name()**

A short description of what function **function\_name()** does.

### Prototype

Provides a prototype for the function **function\_name()**.

```
<return_type> function_name(  
    <type_1>  parameter_1,  
    <type_2>  parameter_2,  
    ...  
    <type_n>  parameter_n)
```

### Parameters

*parameter\_1 [in]* — Pointer to x  
*parameter\_2 [out]* — Handle for y  
*parameter\_n [in/out]* — Pointer to z

Parameter passing is categorized as follows:

- *In* — It means the function uses one or more values in the parameter you give it, without storing any changes.
- *Out* — It means the function saves one or more values in the parameter you give it. You can examine the saved values to find out useful information about your application.
- *In/out* — It means the function changes one or more values in the parameter you give it, and saves the result. You can examine the saved values to find out useful information about your application.

When User-mode and Memory Protection (new in MQX RTOS 3.8) is enabled in the MQX RTOS PSP, there are some additional restrictions on the parameters being passed by a pointer reference to MQX RTOS API functions. See the functions prefixed with the **\_usr** prefix. The following parameter categories should be taken into a consideration:

- *RO* — means the function parameter must be located in the "Read Only" memory for a User task or other code executed in the User mode.
- *RW* — means the function parameter must be located in the "Read Write" memory for a User task or other code executed in the User mode.

## Returns

Specifies any value or values returned by the function.

## Traits

Specifies any of the following that might apply for the function:

- it blocks, or conditions under which it might block
- it must be started as a task
- it creates a task
- it disables and enables interrupts
- pre-conditions that might not be obvious
- any other restrictions or special behavior

## See Also

Lists other functions or data types related to the function **function\_name()**.

## Example

Provides an example (or a reference to an example) that illustrates the use of function **function\_name()**.

## Description

Describes the function **function\_name()**. This section also describes any special characteristics or restrictions that might apply:



- Function blocks, or might block under certain conditions.
- Function must be started as a task.
- Function creates a task.
- Function has pre-conditions that might not be obvious.
- Function has restrictions or special behavior.

## 1.4 Conventions

### 1.4.1 Tips

Tips point out useful information.

#### **TIP**

The most efficient way to allocate a message from an ISR is to use `_msg_alloc()`.

### 1.4.2 Notes

Notes point out important information.

#### **NOTE**

Non-strict semaphores do not have priority inheritance.

### 1.4.3 Cautions

Cautions tell you about commands or procedures that could have unexpected or undesirable side effects or could be dangerous to your files or your hardware.

#### **CAUTION**

If you modify MQX RTOS data types, some MQX RTOS Host Tools from MQX RTOS Embedded might not operate properly.

### 1.4.4 MQX RTOS directory <MQX\_KSDK\_DIR>

In this document, the identifier <MQX\_KSDK\_DIR> designates the directory for MQX RTOS within the KSDK installation.



## Chapter 2 MQX RTOS Functions and Macros

### 2.1 MQX RTOS Function Overview

Table 2-1. MQX RTOS Functions

Component	Prefix
Cache-control macros for data cache	_DCACHE_
Cache-control macros for instruction cache	_ICACHE_
Endian conversion macros	MSG_
Events	_event_
Inter-processor communication	_ipc_
Interrupt handling	_int_
Kernel log	_klog_
Lightweight events	_lwevent_
Lightweight logs	_lwlog_
Lightweight memory with variable-size blocks	_lwmem_
Lightweight semaphores	_lwsem_
Logs (user logs)	_log_
Memory with fixed-size blocks (partitions)	_partition_
Memory with variable-size blocks	_mem_
Messages	_msg_ _msgpool_ _msgq_
Miscellaneous	_mqx_
MMU and virtual memory control	_mmu_
Mutexes	_mutatr_ _mutex_
Names	_name_
Partitions	_partition_
Queues	_queue_
Scheduling	_sched_
Semaphores	_sem_
String functions	_str
Task management	_task_
Task queues	_taskq_
Timers	_timer_
Timing	_time_

**Table 2-1. MQX RTOS Functions**

Virtual memory control	_mmu_
Watchdogs	_watchdog

## 2.1.1 **`_DCACHE_DISABLE`**

If the PSP supports disabling the data cache, the macro calls a PSP-specific function to do so.

### **Prototype**

```
source\psp\cpu_family\cpu.h  
#include <psp.h>  
_DCACHE_DISABLE(void)
```

### **Parameters**

None

### **Returns**

None

### **See Also**

[\*\*`\_DCACHE\_ENABLE`\*\*](#)

## 2.1.2 **`_DCACHE_ENABLE`**

If the PSP supports enabling the data cache, the macro calls a PSP-specific function to do so.

### Prototype

```
source\psp\cpu_family\cpu.h  
#include <psp.h>  
_DCACHE_ENABLE(  
    uint32_t flags)
```

### Parameters

*flags* [IN] — CPU-type-specific flags that the processor needs to enable its data cache

### Returns

None

### See Also

[\*\*`\_DCACHE\_DISABLE`\*\*](#)

### 2.1.3 **\_DCACHE\_FLUSH**

If the PSP supports flushing the data cache, the macro calls a PSP-specific function to do so.

#### Prototype

```
source\psp\cpu_family\cpu.h
#include <psp.h>
_DCACHE_FLUSH(void)
```

#### Parameters

None

#### Returns

None

#### See Also

[\\_DCACHE\\_FLUSH\\_LINE](#)

[\\_DCACHE\\_FLUSH\\_MLINES](#)

#### Description

The macro flushes the entire data cache. Unwritten data that is in the cache is written to physical memory.

#### CAUTION

The flushing and invalidating functions always operate with whole cache lines. In case the data entity is not aligned to the cache line size these operations affects data that precedes and follows data area currently being flushed/invalidated.

The MQX RTOS memory allocators align data entity to the cache line size by default. Once an entity is declared statically the alignment to the cache line size is not guaranteed (unless align pragma is used).

On some CPUs, flushing the data cache also invalidates the data cache entries.

## 2.1.4 `_DCACHE_FLUSH_LINE`

If the PSP supports flushing one data cache line, the macro calls a PSP-specific function to flush the line.

### Prototype

```
source\psp\cpu_family\cpu.h
#include <psp.h>
_DCACHE_FLUSH_LINE(
    void *addr)
```

### Parameters

*addr* [IN] — Address to be flushed

### Returns

None

### See Also

[\\_DCACHE\\_FLUSH](#)

[\\_DCACHE\\_FLUSH\\_MLINES](#)

### Description

The line that is flushed is the one that contains *addr*.

The macro is used when a device (such as a DMA) needs to access memory and the CPU does not provide bus snooping.

### CAUTION

The flushing and invalidating functions always operate with whole cache lines. In case the data entity is not aligned to the cache line size these operations affects data that precedes and follows data area currently being flushed/invalidated.

The MQX RTOS memory allocators align data entity to the cache line size by default. Once an entity is declared statically the alignment to the cache line size is not guaranteed (unless align pragma is used).

On some CPUs, flushing the data cache also invalidates the data cache entries.

### Example

Flush a data cache line.

```
...
uint32_t data;
...
data = 55;
_DCACHE_FLUSH_LINE(&data);
```



## 2.1.5 `_DCACHE_FLUSH_MLINES`

If the PSP supports flushing a memory region from the data cache, the macro calls a PSP-support function to flush the region.

### Prototype

```
source\psp\cpu_family\cpu.h
#include <psp.h>
_DCACHE_FLUSH_MLINES(
    void      *addr,
    _mem_size  length)
```

### Parameters

*addr* [IN] — Address from which to start flushing the data cache

*length* [IN] — Number of single-addressable units to flush

### Returns

None

### See Also

[`\_DCACHE\_FLUSH`](#)

[`\_DCACHE\_FLUSH\_LINE`](#)

### Description

The macro is used when a device (such as a DMA) needs to access memory and the CPU does not provide bus snooping.

### CAUTION

The flushing and invalidating functions always operate with whole cache lines. In case the data entity is not aligned to the cache line size these operations affects data that precedes and follows data area currently being flushed/invalidated.

The MQX RTOS memory allocators align data entity to the cache line size by default. Once an entity is declared statically the alignment to the cache line size is not guaranteed (unless align pragma is used).

On some CPUs, flushing the data cache also invalidates the data cache entries.

### Example

Flush an array of data from the data cache.

```
...
uint32_t data[10];
...
data[5] = 55;
_DCACHE_FLUSH_MLINES(data, sizeof(data));
```

## 2.1.6 **\_DCACHE\_INVALIDATE**

If the PSP supports invalidating all the data cache entries, the macro calls a PSP-specific function to do so.

### Prototype

```
source\psp\cpu_family\cpu.h  
#include <psp.h>  
_DCACHE_INVALIDATE(void)
```

### Parameters

None

### Returns

None

### See Also

[\\_DCACHE\\_INVALIDATE\\_LINE](#)

[\\_DCACHE\\_INVALIDATE\\_MLINES](#)

### Description

Data that is in the data cache and has not been written to memory is lost. A subsequent data access reloads the cache with data from physical memory.

### CAUTION

The flushing and invalidating functions always operate with whole cache lines. In case the data entity is not aligned to the cache line size these operations affects data that precedes and follows data area currently being flushed/invalidated.

The MQX RTOS memory allocators align data entity to the cache line size by default. Once an entity is declared statically the alignment to the cache line size is not guaranteed (unless align pragma is used).

## 2.1.7 `_DCACHE_INVALIDATE_LINE`

If the PSP supports invalidating one data cache line, the macro calls a PSP-specific function to invalidate the line.

### Prototype

```
source\psp\cpu_family\cpu.h
#include <psp.h>
_DCACHE_INVALIDATE_LINE(
    void *addr)
```

### Parameters

*addr* [IN] — Address to be invalidated

### Returns

None

### See Also

[`\_DCACHE\_INVALIDATE`](#)

[`\_DCACHE\_INVALIDATE\_MLINES`](#)

### Description

The line that is invalidated is the one that contains *addr*.

The macro is used when a device (such as a DMA) needs to access memory and the CPU does not provide bus snooping.

### CAUTION

The flushing and invalidating functions always operate with whole cache lines. In case the data entity is not aligned to the cache line size these operations affects data that precedes and follows data area currently being flushed/invalidated.

The MQX RTOS memory allocators align data entity to the cache line size by default. Once an entity is declared statically the alignment to the cache line size is not guaranteed (unless align pragma is used).

### Example

Invalidate a data cache line.

```
...
uint32_t data;
...
_DCACHE_INVALIDATE_LINE(&data);
if (data == 55) {
    ...
}
```

## 2.1.8 `_DCACHE_INVALIDATE_MLINES`

If the PSP supports invalidating a memory region in the data cache, the macro calls a PSP-specific function to invalidate the region.

### Prototype

```
source\psp\cpu_family\cpu.h
#include <psp.h>
_DCACHE_INVALIDATE_MLINES(
    void      *addr,
    _mem_size  length)
```

### Parameters

*addr* [IN] — Address from which to start invalidating the data cache

*length* [IN] — Number of single-addressable units to invalidate

### Returns

None

### See Also

[`\_DCACHE\_INVALIDATE`](#)

[`\_DCACHE\_INVALIDATE\_LINE`](#)

### Description

The macro is used when a device (such as a DMA) needs to access memory and the CPU does not provide bus snooping.

### CAUTION

The flushing and invalidating functions always operate with whole cache lines. In case the data entity is not aligned to the cache line size these operations affects data that precedes and follows data area currently being flushed/invalidated.

The MQX RTOS memory allocators align data entity to the cache line size by default. Once an entity is declared statically the alignment to the cache line size is not guaranteed (unless align pragma is used).

### Example

Invalidate an array of data in the data cache.

```
...
uint32_t data[10];
...
_DCACHE_INVALIDATE_MLINES(data, sizeof(data));
if (data[5] == 55) {
}
```

## 2.1.9 `_event_clear`

Clears the specified event bits in the event group.

### Prototype

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_clear(
    void *event_group_ptr,
    _mx_uint bit_mask)
```

### Parameters

*event\_group\_ptr* [IN] — Event group handle returned by `_event_open()` or `_event_open_fast()`

*bit\_mask* [IN] — Each set bit represents an event bit to clear

### Returns

- MQX\_OK
- Errors

Error	Description
EVENT_INVALID_EVENT	Event group is not valid.
EVENT_INVALID_EVENT_HANDLE	One of the following: <ul style="list-style-type: none"> <li>• <code>_event_open()</code> or <code>_event_open_fast()</code> did not get the event group handle</li> <li>• <code>_event_create()</code> did not create the event group</li> </ul>

### Traits

### See Also

[\\_event\\_create](#), [\\_event\\_create\\_auto\\_clear](#)

[\\_event\\_open](#)

[\\_event\\_open\\_fast](#)

[\\_event\\_set](#)

[\\_event\\_get\\_value](#)

[\\_event\\_wait\\_all ...](#)

[\\_event\\_wait\\_any ...](#)

### Example

Task 1 waits for an event condition so that it can do some processing. When Task 2 sets the event bit, Task 1 does the processing. When Task 1 finishes the processing, it clears the event bit so that another task can set the bit the next time the event condition occurs.

```
void *event_ptr;
```

```
result = _event_open("global", &event_ptr);
if (result == MQX_OK) {
    while (TRUE) {
        result = _event_wait_all(event_ptr, 0x01, 0);
        /* Do some processing. */
        . . . .
        result = _event_clear(event_ptr, 0x01);
    }
    result = _event_close(event_ptr);
}
```

## 2.1.10 `_event_close`

Closes the connection to the event group.

### Prototype

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_close(
    void *event_group_ptr)
```

### Parameters

*event\_group\_ptr [IN]* — Event group handle returned by `_event_open()` or `_event_open_fast()`

### Returns

- `MQX_OK`
- Errors

### Errors

Task error code from `_mem_free()`

MQX RTOS could not free the event group handle.

Error	Description
<code>EVENT_INVALID_EVENT_HANDLE</code>	Event group connection is not valid.
<code>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</code>	Function cannot be called from an ISR.

### Traits

Cannot be called from an ISR

### See Also

[\\_event\\_destroy](#)

[\\_event\\_open](#)

[\\_event\\_open\\_fast](#)

### Description

The function closes the connection to the event group and frees the event group handle.

A task that opened an event group on a remote processor can also close the event group.

### Example

See [\\_event\\_clear\(\)](#).

## 2.1.11 `_event_create`, `_event_create_auto_clear`

<code>_event_create()</code>	Creates the named event group.
<code>_event_create_auto_clear()</code>	Creates the named event group with autoclearing event bits.

### Prototype

`_event_create()`

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_create(
    char *name)
```

`_event_create_auto_clear()`

```
source\event\ev_creaa.c
#include <event.h>
_mqx_uint _event_create_auto_clear(
    char *name)
```

### Parameters

*name* [IN] — Name of the event group

### Returns

- MQX\_OK
- Errors

Error	Description
EVENT_EXISTS	Event group was already created.
EVENT_TABLE_FULL	Name table is full and cannot be expanded.
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_INVALID_COMPONENT_BASE	Event component data is not valid.
MQX_OUT_OF_MEMORY	MQX RTOS could not allocate memory for the event group.

### Traits

- Creates the event component with default values if it was not previously created
- Cannot be called from an ISR

### See Also

[\\_event\\_close](#)

[\\_event\\_create\\_component](#)



[\*\*\\_event\\_destroy\*\*](#)[\*\*\\_event\\_open\*\*](#)**Description**

After a task creates a named event group, any task that wants to use it must open a connection to it with **\_event\_open()**. When a task no longer needs a named event group, it can destroy the event group with **\_event\_destroy()**.

If a task creates an event group with autoclearing event bits, MQX RTOS clears the event bits as soon as they are set. Task that are waiting for the event bits are made ready, but need not clear the bits.

**Example**

See [\*\*\\_event\\_create\\_component\(\)\*\*](#).

## 2.1.12 `_event_create_component`

Creates the event component.

### Prototype

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_create_component(
    _mx_uint initial_number,
    _mx_uint grow_number,
    _mx_uint maximum_number)
```

### Parameters

*initial\_number* [IN] — Initial number of event groups that the application can create  
*grow\_number* [IN] — Number of event groups to add if the application creates all the event groups  
*maximum\_number* [IN] — If *grow\_number* is non-zero, maximum number of event groups (0 means an unlimited number)

### Returns

- MQX\_OK (success)
- MQX\_OUT\_OF\_MEMORY (MQX RTOS could not allocate memory for the event group)
- MQX\_CANNOT\_CALL\_FUNCTION\_FROM\_ISR (Function cannot be called from an ISR.)

### See Also

[\\_event\\_create](#), [\\_event\\_create\\_auto\\_clear](#)

[\\_event\\_create\\_fast](#), [\\_event\\_create\\_fast\\_auto\\_clear](#)

[\\_event\\_open](#)

[\\_event\\_open\\_fast](#)

### Description

If an application previously called the function and *maximum\_number* is now greater than what was previously specified, MQX RTOS changes the maximum number of event groups to *maximum\_number*.

If an application does not explicitly create the event component, MQX RTOS does so with the following default values the first time that a task calls a function in the **`_event_create`** family of functions.

Parameter	Default
<i>initial_number</i>	8
<i>grow_number</i>	8
<i>maximum_number</i>	0 (unlimited)

## Example

Create the event component with two event groups, the ability to grow by one, and up to a maximum of four. Create an event group, do some processing, and then destroy the event group.

```
result = _event_create_component(2, 1, 4);
if (result != MQX_OK)
{
    printf("\nCould not create the event component");
    _mqx_exit();
}
result = _event_create("global");
...
result = _event_destroy("global");
```

## 2.1.13 `_event_create_fast`, `_event_create_fast_auto_clear`

<code>_event_create_fast()</code>	Creates the fast event group.
<code>_event_create_fast_auto_clear()</code>	Creates the fast event group with autoclearing event bits.

### Prototype

`_event_create_fast()`

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_create_fast(
    _mx_uint index)
```

`_event_create_fast_auto_clear()`

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_create_fast_auto_clear(
    _mx_uint index)
```

### Parameters

*index* [IN] — Number of the event group

### Returns

- MQX\_OK (success)
- Error: See [\\_event\\_create](#), [\\_event\\_create\\_auto\\_clear](#)

### Traits

- Creates the event component with default values if they were not previously created
- Cannot be called from an ISR

### See Also

[\\_event\\_close](#)

[\\_event\\_create](#), [\\_event\\_create\\_auto\\_clear](#)

[\\_event\\_create\\_component](#)

[\\_event\\_destroy\\_fast](#)

[\\_event\\_open\\_fast](#)

### Description

See [\\_event\\_create](#), [\\_event\\_create\\_auto\\_clear](#).

### Example

```
#define MY_EVENT_GROUP 123
```

```
void *event_ptr;
...
result = _event_create_fast(MY_EVENT_GROUP);
if (result != MQX_OK) {
    _mqx_exit();
}
result = _event_open_fast(MY_EVENT_GROUP, &event_ptr);
if (result != MQX_OK) {
    _mqx_exit();
}
...
result = _event_close(event_ptr);
result = _event_destroy_fast(MY_EVENT_GROUP);
...
```

## 2.1.14 `_event_destroy`

Destroys the named event group.

### Prototype

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_destroy(
    char *name)
```

### Parameters

*name* [IN] — Name of the event group

### Returns

- MQX\_OK
- Errors

Error	Description
EVENT_INVALID_EVENT	Event group is no longer valid.
EVENT_NOT_FOUND	Event group is not in the table.
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_COMPONENT_DOES_NOT_EXIST	Event component was not created.
MQX_INVALID_COMPONENT_BASE	Event component data is not valid.

### Traits

Cannot be called from an ISR

### See Also

[\\_event\\_create](#), [\\_event\\_create\\_auto\\_clear](#)

[\\_event\\_create\\_component](#)

[\\_event\\_wait\\_all ...](#)

[\\_event\\_wait\\_any ...](#)

### Description

The event group must have been created with `_event_create()` or `_event_create_auto_clear()`.

If tasks are blocked waiting for an event bit in the event group, MQX RTOS does the following:

- moves them to their ready queues
- sets their task error code to **EVENT\_DELETED**

- returns **EVENT\_DELETED** for `_event_wait_all()` and `_event_wait_any()`

### Example

See [\\_event\\_create\\_component\(\)](#).

## 2.1.15 `_event_destroy_fast`

Destroys the fast event group.

### Prototype

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_destroy_fast(
    _mx_uint index)
```

### Parameters

*index [IN]* — Number of the event group

### Returns

- MQX\_OK
- Error: See [\\_event\\_destroy](#)

### Traits

Cannot be called from an ISR

### See Also

[\\_event\\_create\\_component](#)

[\\_event\\_create\\_fast](#), [\\_event\\_create\\_fast\\_auto\\_clear](#)

### Description

The event group must have been created with [\\_event\\_create\\_fast\(\)](#) or [\\_event\\_create\\_fast\\_auto\\_clear\(\)](#).

See [\\_event\\_destroy](#).

### Example

See [\\_event\\_create\\_fast](#), [\\_event\\_create\\_fast\\_auto\\_clear](#).



## 2.1.16 `_event_get_value`

Gets the event bits for the event group.

### Prototype

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_get_value(
    void      *event_group_ptr,
    _mx_uint  *event_group_value_ptr)
```

### Parameters

*event\_group\_ptr [IN]* — Event group handle returned by `_event_open()` or `_event_open_fast()`  
*event\_group\_value\_ptr [OUT]* — Where to write the value of the event bits (on error, 0 is written)

### Returns

- `MQX_OK`
- Errors

Error	Description
<code>EVENT_INVALID_EVENT</code>	Event group is no longer valid.
<code>EVENT_INVALID_EVENT_HANDLE</code>	Event group handle is not valid.

### See Also

[\\_event\\_clear](#)  
[\\_event\\_set](#)  
[\\_event\\_wait\\_all ...](#)  
[\\_event\\_wait\\_any ...](#)

### Example

If another task has set event bit 0, this task sets event bit 1.

```
void *event_ptr;
_mqx_uint event_bits;
...
if (_event_open("global", &event_ptr) == MQX_OK) {
    for (;;) {
        if (_event_get_value(event_ptr, &event_bits) == MQX_OK) {
            if (event_bits & 0x01) {
                _event_set(event_ptr, 0x02);
            }
        }
    }
}
```

```
    ...  
}  
}
```

## 2.1.17 `_event_get_wait_count`

Gets the number of tasks that are waiting for event bits in the event group.

### Prototype

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_get_wait_count(
    void *event_group_ptr)
```

### Parameters

*event\_group\_ptr [IN]* — Event group handle returned by `_event_open()` or `_event_open_fast()`

### Returns

- Number of waiting tasks (success)
- `MAX_MQX_UINT` (failure)

### Traits

On failure, calls `_task_set_error()` to set the task error code to `EVENT_INVALID_EVENT_HANDLE`.

### See also

[\\_event\\_open](#)

[\\_event\\_open\\_fast](#)

[\\_event\\_wait\\_all ...](#)

[\\_event\\_wait\\_any ...](#)

[\\_task\\_set\\_error](#)

### Description

Tasks can be waiting for different combinations of event bits.

### Example

```
void *event_ptr;
_mqx_uint task_wait_count;
...
if (_event_open("global", &event_ptr) == MQX_OK) {
    ...
    task_wait_count = _event_get_wait_count(event_ptr);
    ...
}
```

## 2.1.18 `_event_open`

Opens a connection to the named event group.

### Prototype

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_open(
    char *name_ptr,
    void **event_ptr)
```

### Parameters

*name\_ptr* [IN] — Pointer to the name of the event group (see description)

*event\_ptr* [OUT] — Where to write the event group handle (*NULL* is written if an error occurred)

### Returns

- MQX\_OK
- Errors

Error	Description
EVENT_INVALID_EVENT	Event group data is no longer valid.
EVENT_NOT_FOUND	Named event group is not in the name table.
MQX_COMPONENT_DOES_NOT_EXIST	Event component is not created.
MQX_INVALID_COMPONENT_BASE	Event component data is not valid.
MQX_OUT_OF_MEMORY	MQX RTOS could not allocate memory for the event connection data.

### See Also

[\\_event\\_close](#)

[\\_event\\_create](#), [\\_event\\_create\\_auto\\_clear](#),

[\\_event\\_set](#)

[\\_event\\_get\\_wait\\_count](#)

[\\_event\\_get\\_value](#)

[\\_event\\_wait\\_all ...](#)

[\\_event\\_wait\\_any ...](#)

## Description

The named event group must have been created with `_event_create()` or `_event_create_auto_clear()`. Each task that needs access to the named event group must first open a connection to it.

To open an event group on a remote processor, prepend the event-group name with the remote processor number as follows.

This string:	Opens this named event group:	On this processor:
"2:Fred"	"Fred"	2
"0:Sue"	"Sue"	Local processor

The other allowed event operations on remote processors are:

- `_event_set()`
- `_event_close()`

The task closes the connection with `_event_close()`.

## Example

See [\\_event\\_clear\(\)](#).

## 2.1.19 `_event_open_fast`

Opens a connection to the fast event group.

### Prototype

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_open_fast(
    _mx_uint      index,
    void **event_group_ptr)
```

### Parameters

*index* [IN] — Index of the event group

*event\_group\_ptr* [OUT] — Where to write the event group handle (*NULL* is written if an error occurred)

### Returns

- **MQX\_OK**
- Error: See [\\_event\\_open](#)

### See Also

[\\_event\\_close](#)

[\\_event\\_create\\_fast](#), [\\_event\\_create\\_fast\\_auto\\_clear](#)

[\\_event\\_set](#)

[\\_event\\_get\\_wait\\_count](#)

[\\_event\\_get\\_value](#)

[\\_event\\_wait\\_all ...](#)

[\\_event\\_wait\\_any ...](#)

### Description

See [\\_event\\_open](#).

### Example

See [\\_event\\_create\\_fast](#), [\\_event\\_create\\_fast\\_auto\\_clear](#).

## 2.1.20 `_event_set`

Sets the specified event bits in the event group.

### Prototype

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_set(
    void      *event_group_ptr,
    _mx_uint  bit_mask)
```

### Parameters

*event\_group\_ptr* [IN] — Event group handle returned by `_event_open()` or `_event_open_fast()`  
*bit\_mask* [IN] — Each set bit represents an event bit to be set

### Returns

- MQX\_OK
- Errors

Error	Description
EVENT_INVALID_EVENT	Event group is no longer valid.
EVENT_INVALID_EVENT_HANDLE	Event group handle is not a valid event connection.
MQX_COMPONENT_DOES_NOT_EXIST	Event component is not created.
MQX_INVALID_COMPONENT_BASE	Event component data is no longer valid.

### Traits

Tasks waiting for the event bits might be dispatched.

### See Also

[\\_event\\_get\\_wait\\_count](#)

[\\_event\\_get\\_value](#)

[\\_event\\_wait\\_all ...](#)

[\\_event\\_wait\\_any ...](#)

### Description

Before a task can set an event bit in an event group, the event group must be created and the task must open an connection to the event group.

A task can set or clear one event bit or any combination of event bits in the event group.

A task that opened an event group on a remote processor can set bits in the event group.

## Example

The task is responsible for setting event bits 0 and 1 in the named event.

```
void    *event_ptr;
_mqx_uint result;
...
if (_event_create("global") == MQX_OK) {
    if (_event_open("global", &event_ptr) == MQX_OK) {
        for (;;) {
            /*If some condition is true, */
            _event_set(event_ptr, 0x03);
            ...
        }
    }
}
```



## 2.1.21 `_event_test`

Tests the event component.

### Prototype

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_test(
    void **event_error_ptr)
```

### Parameters

*event\_error\_ptr* [OUT] — Handle for the event group that has an error if MQX RTOS found an error in the event component (*NULL* if no error is found)

### Returns

- MQX\_OK
- Errors

Error	Description
EVENT_INVALID_EVENT	Data for an event group is not valid.
MQX_INVALID_COMPONENT_BASE	Event component data is not valid.
Return code from <code>_queue_test()</code>	Waiting queue for an event group has an error.

### See Also

[\\_event\\_close](#)

[\\_event\\_open](#)

[\\_event\\_set](#)

[\\_event\\_get\\_wait\\_count](#)

[\\_event\\_get\\_value](#)

[\\_event\\_wait\\_all ...](#)

[\\_event\\_wait\\_any ...](#)

### Example

```
void *event_ptr;
...
if (_event_test(&event_ptr) != MQX_OK) {
    printf("Event component test failed - Event group in error: 0x%lx",
        event_ptr);
    ...
}
```

## 2.1.22 `_event_wait_all ...`

	Wait for all the specified event bits to be set in the event group:
<code>_event_wait_all()</code>	For the number of milliseconds
<code>_event_wait_all_for()</code>	For the number of ticks (in tick time)
<code>_event_wait_all_ticks()</code>	For the number of ticks
<code>_event_wait_all_until()</code>	Until the specified time (in tick time)

### Prototype

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_wait_all(
    void      *event_group_ptr,
    _mx_uint  bit_mask,
    uint32_t   ms_timeout)

source\kernel\event.c
#include <event.h>
_mqx_uint _event_wait_all_for(
    void      *event_group_ptr,
    _mx_uint  bit_mask,
    MQX_TICK_STRUCT_PTR tick_time_timeout_ptr)

source\kernel\event.c
#include <event.h>
_mqx_uint _event_wait_all_ticks(
    void      *event_group_ptr,
    _mx_uint  bit_mask,
    uint32_t   tick_timeout)

source\kernel\event.c
#include <event.h>
_mqx_uint _event_wait_all_until(
    void      *event_group_ptr,
    _mx_uint  bit_mask,
    MQX_TICK_STRUCT_PTR tick_time_ptr)
```

### Parameters

*event\_group\_ptr* [IN] — Event group handle returned by [\\_event\\_open](#) or [\\_event\\_open\\_fast](#)

*bit\_mask* [IN] — Each set bit represents an event bit to wait for

*ms\_timeout* [IN] — One of the following:

maximum number of milliseconds to wait for the events to be set. After the timeout elapses without the event signalled, the function returns.

0 (unlimited wait)

*tick\_time\_timeout\_ptr [IN]* — One of the following:

pointer to the maximum number of ticks to wait  
 NULL (unlimited wait)

*tick\_timeout [IN]* — One of the following:

maximum number of ticks to wait  
 0 (unlimited wait)

*tick\_time\_ptr [IN]* — One of the following:

pointer to the time (in tick time) until which to wait  
 NULL (unlimited wait)

## Returns

- MQX\_OK
- Errors

Error	Description
EVENT_DELETED	Event group was destroyed while the task waited.
EVENT_INVALID_EVENT	Event group is no longer valid.
EVENT_INVALID_EVENT_HANDLE	Handle is not a valid event group handle.
EVENT_WAIT_TIMEOUT	Timeout expired before the event bits were set.
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.

## Traits

- Blocks until the event combination is set or until the timeout expires
- Cannot be called from an ISR

## See Also

[\\_event\\_clear](#)

[\\_event\\_open](#)

[\\_event\\_open\\_fast](#)

[\\_event\\_set](#)

[\\_event\\_get\\_wait\\_count](#)

[\\_event\\_get\\_value](#)

[\\_event\\_wait\\_any ...](#)

## Example

See [\\_event\\_clear](#).

## 2.1.23 `_event_wait_any ...`

	Wait for any of the specified event bits to be set in the event group:
<code>_event_wait_any()</code>	For the number of milliseconds
<code>_event_wait_any_for()</code>	For the number of ticks (in tick time)
<code>_event_wait_any_ticks()</code>	For the number of ticks
<code>_event_wait_any_until()</code>	Until the specified time (in tick time)

### Prototype

```
source\kernel\event.c
#include <event.h>
_mqx_uint _event_wait_any(
    void      *event_group_ptr,
    _mx_uint  bit_mask,
    uint32_t   ms_timeout)

source\kernel\event.c
#include <event.h>
_mqx_uint _event_wait_any_for(
    void      *event_group_ptr,
    _mx_uint  bit_mask,
    MQX_TICK_STRUCT_PTR tick_time_timeout_ptr)

source\kernel\event.c
#include <event.h>
_mqx_uint _event_wait_any_ticks(
    void      *event_group_ptr,
    _mx_uint  bit_mask,
    _mx_uint  tick_timeout)

source\kernel\event.c
#include <event.h>
_mqx_uint _event_wait_any_until(
    void      *event_group_ptr,
    _mx_uint  bit_mask,
    MQX_TICK_STRUCT_PTR tick_time_ptr)
```

### Parameters

*event\_group\_ptr* [IN] — Event group handle returned by `_event_open()` or `_event_open_fast()`

*bit\_mask* [IN] — Each set bit represents an event bit to wait for

*ms\_timeout* [IN] — One of the following:

- maximum number of milliseconds to wait
- 0 (unlimited wait)

*tick\_time\_timeout\_ptr* [IN] — One of the following:

pointer to the maximum number of ticks to wait  
*NULL* (unlimited wait)

*tick\_timeout* [IN] — One of the following:

maximum number of ticks to wait  
 0 (unlimited wait)

*tick\_time\_ptr* [IN] — One of the following:

pointer to the time (in tick time) until which to wait  
*NULL* (unlimited wait)

### Returns

- MQX\_OK
- See `_event_wait_all` family

### Traits

- Blocks until the event combination is set or until the timeout expires
- Cannot be called from an ISR

### See also

[\\_event\\_clear](#)

[\\_event\\_open](#)

[\\_event\\_open\\_fast](#)

[\\_event\\_set](#)

[\\_event\\_get\\_wait\\_count](#)

[\\_event\\_get\\_value](#)

[\\_event\\_wait\\_all ...](#)

### Example

See [\\_event\\_clear](#).

## 2.1.24 **\_ICACHE\_DISABLE**

If the PSP supports disabling the instruction cache, the macro calls a PSP-specific function to do so.

### Prototype

```
source\psp\cpu_family\cpu.h  
#include <psp.h>  
_ICACHE_DISABLE(void)
```

### Parameters

None

### Returns

None

### See Also

[\\_ICACHE\\_ENABLE](#)

## 2.1.25 `_ICACHE_ENABLE`

If the PSP supports enabling the instruction cache, the macro calls a PSP-specific function to do so.

### Prototype

```
source\psp\cpu_family\cpu.h  
#include <psp.h>  
_ICACHE_ENABLE(  
    uint32_t flags)
```

### Parameters

*flags* [IN] — CPU-type-specific flags that the processor needs to enable its instruction cache

### Returns

None

### See Also

[\\_ICACHE\\_DISABLE](#)

## 2.1.26 **\_ICACHE\_INVALIDATE**

If the PSP supports invalidating all the entries in the instruction cache, the macro calls a PSP-specific function to do so.

### Prototype

```
source\psp\cpu_family\cpu.h  
#include <psp.h>  
_ICACHE_INVALIDATE(void)
```

### Parameters

None

### Returns

None

### See Also

- [\\_ICACHE\\_INVALIDATE\\_LINE](#)
- [\\_ICACHE\\_INVALIDATE\\_MLINES](#)

### Description

Instructions that are in the cache and have not been written to memory are lost. A subsequent instruction access reloads the cache with instructions from physical memory.



## 2.1.27 `_ICACHE_INVALIDATE_LINE`

If the PSP supports invalidating one instruction cache line, the macro calls a PSP-specific function to invalidate the line.

### Prototype

```
source\psp\cpu_family\cpu.h
#include <psp.h>
_ICACHE_INVALIDATE_LINE(
    void *addr)
```

### Parameters

*addr* [IN] — Address to be invalidated

### Returns

None

### See Also

- [\\_ICACHE\\_INVALIDATE](#)
- [\\_ICACHE\\_INVALIDATE\\_MLINES](#)

### Description

The line that is invalidated is the one that contains *addr*.

If an application writes to code space (such as when it patches or loads code), the instruction cache for write operations is incorrect. In this case, the application calls `_ICACHE_INVALIDATE_LINE` to invalidate the appropriate line in the cache.

### NOTE

The amount of memory that is invalidated depends on the size of the CPU's instruction cache line.

### Example

Invalidate an instruction cache line.

```
...
extern int some_function();
...
_ICACHE_INVALIDATE_LINE(&some_function);
```

## 2.1.28 `_ICACHE_INVALIDATE_MLINES`

If the PSP supports invalidating a memory region in the instruction cache, the macro calls a PSP-specific function to invalidate the region.

### Prototype

```
source\psp\cpu_family\cpu.h
#include <psp.h>
_ICACHE_INVALIDATE_MLINES(
    void      *addr,
    _mem_size  length)
```

### Parameters

*addr* [IN] — Address from which to start invalidating the instruction cache

*length* [IN] — Number of single-addressable units to invalidate

### Returns

None

### See Also

- [\\_ICACHE\\_INVALIDATE](#)
- [\\_ICACHE\\_INVALIDATE\\_LINE](#)

### Description

If an application writes to code space (such as when it patches or loads code), the instruction cache for write operations is incorrect. In this case, the application calls `_ICACHE_INVALIDATE_MLINES` to invalidate the appropriate lines in the cache.

### Example

Invalidate an entire function in the instruction cache.

```
...
extern int some_function();
extern int end_some_function();
...
_ICACHE_INVALIDATE_MLINES(some_function, end_some_function -
    some_function);
```

## 2.1.29 `_int_default_isr`

Default ISR that MQX RTOS calls if an unhandled interrupt or exception occurs.

### Prototype

```
source\kernel\int.c  
void _int_default_isr(  
    void *vector_number)
```

### Parameters

*vector\_number* [IN] — Parameter that MQX RTOS passes to the ISR

### Returns

None

### Traits

Blocks the active task

### See Also

[`\_int\_install\_default\_isr`](#)

[`\_int\_install\_unexpected\_isr`](#)

[`\_int\_install\_exception\_isr`](#)

### Description

An application can replace the function with `_int_install_unexpected_isr()` or `_int_install_exception_isr()`, both of which install MQX RTOS-provided default ISRs.

An application can install an application-provided default ISR with `_int_install_default_isr()`.

MQX RTOS changes the state of the active task to **UNHANDLED\_INT\_BLOCKED** and blocks it.

## 2.1.30 `_int_disable`, `_int_enable`

<code>_int_disable()</code>	Disable hardware interrupts.
<code>_int_enable()</code>	Enable hardware interrupts.

### Prototype

```
source\kernel\int.c
void _int_disable(void)

void _int_enable(void)
```

### Parameters

None

### Returns

None

### Description

The function `_int_enable()` resets the processor priority to the hardware priority that corresponds to the active task's software priority.

The function `_int_disable()` disables all hardware interrupts at priorities up to and including the MQX RTOS disable-interrupt level. As a result, no task can interrupt the active task while the active task is running until interrupts are re-enabled with `_int_enable()`. If the active task blocks while interrupts are disabled, the state of the interrupts (disabled or enabled) depends on the interrupt-disabled state of the next task that MQX RTOS makes ready.

Keep to a minimum code between calls to `_int_disable()` and its matching `_int_enable()`.

If `_int_disable()` or `_int_enable()` are nested, MQX RTOS re-enables interrupts only after the number of calls to `_int_enable()` equals the number of calls to `_int_disable()`.

### Example

See `_task_ready()`.

### 2.1.31 `_int_exception_isr`

To provide support for exception handlers, applications can use this ISR to replace the default ISR. The ISR is specific to the PSP.

#### Prototype

```
source\psp\cpu_family\int_xcpt.c
void _int_exception_isr(
    void    *parameter)
```

#### Parameters

*parameter [IN]* — Parameter passed to the default ISR (the vector number)

#### Returns

None

#### Traits

See description

#### See Also

[`\_int\_install\_exception\_isr`](#)

[`\_mqx\_fatal\_error`](#)

[`\_task\_abort`](#)

#### Description

An application calls `_int_install_exception_isr()` to install `_int_exception_isr()`.

The function `_int_exception_isr()` does the following:

- If an exception occurs when a task is running and a task exception ISR exists, MQX RTOS runs the ISR; if a task exception ISR does not exist, MQX RTOS aborts the task by calling `_task_abort()`.
- If an exception occurs when an ISR is running and an ISR exception ISR exists, MQX RTOS aborts the running ISR and runs the ISR's exception ISR.
- The function walks the interrupt stack looking for information about the ISR or task that was running before the exception occurred. If the function determines that the interrupt stack contains incorrect information, it calls `_mqx_fatal_error()` with error code `MQX_CORRUPT_INTERRUPT_STACK`.

### 2.1.32 `_int_get_default_isr`

Gets a pointer to the default ISR that MQX RTOS calls when an unexpected interrupt occurs.

#### Prototype

```
source\kernel\int.c  
INT_ISR_FPTR _int_get_default_isr(void)
```

#### Parameters

None

#### Returns

- Pointer to the default ISR for unhandled interrupts (success)
- *NULL* (failure)

#### See Also

[`\_int\_install\_default\_isr`](#)

### 2.1.33 `_int_get_exception_handler`

Gets a pointer to the current ISR exception handler for the vector number.

#### Prototype

```
source\kernel\int.c  
INT_EXCEPTION_FPTR _int_get_exception_handler(  
    _mqx_uint vector)
```

#### Parameters

*vector* [IN] — Vector number whose exception handler is to be returned

#### Returns

- Pointer to the current exception handler (success)
- *NULL* (failure)

#### Traits

On failure, calls `_task_set_error()` to set the task error code

#### See Also

[`\_int\_set\_exception\_handler`](#)

[`\_int\_exception\_isr`](#)

[`\_task\_set\_error`](#)

#### Description

The returned exception handler is either a default ISR or an ISR that the application installed with `_int_set_exception_handler()`.

## 2.1.34 `_int_get_isr`

Gets the current ISR for the vector number.

### Prototype

```
source\kernel\int.c  
INT_ISR_FPTR _int_get_isr(  
    _mqx_uint vector)
```

### Parameters

*vector* [IN] — Vector number whose ISR is to be returned

### Returns

- Pointer to the ISR (success)
- *NULL* (failure)

### Traits

On failure, calls `_task_set_error()` to set the task error code

### See Also

[\\_int\\_get\\_isr\\_data](#)

[\\_int\\_set\\_isr\\_data](#)

[\\_task\\_set\\_error](#)

### Description

The returned ISR is either a default ISR or an ISR that the application installed with `_int_install_isr()`.

### Example

See `_int_get_kernel_isr()`.



## 2.1.35 `_int_get_isr_data`

Gets the data that is associated with the vector number.

### Prototype

```
source\kernel\int.c  
void *_int_get_isr_data(  
    _mqx_uint  vector)
```

### Parameters

*vector* [IN] — Vector number whose ISR data is to be returned

### Returns

- Pointer to ISR data (success)
- *NULL* (failure)

### Traits

On failure, calls `_task_set_error()` to set the task error code

### See Also

[\\_int\\_get\\_isr](#)

[\\_int\\_install\\_isr](#)

[\\_int\\_set\\_isr\\_data](#)

### Description

An application installs ISR data with `_int_set_isr_data()`.

When MQX RTOS calls `_int_kernel_isr()` or an application ISR, it passes the data as the first parameter to the ISR.

### Example

See `_int_get_kernel_isr()`.

### 2.1.36 `_int_get_isr_depth`

Gets the depth of nesting of the current interrupt stack.

#### Prototype

```
source\kernel\int.c  
_mqx_uint _int_get_isr_depth(void)
```

#### Parameters

None

#### Returns

- 0 (an interrupt is not being serviced)
- 1 (a non-nested interrupt is being serviced)
- $\geq 2$  (a nested interrupt is being serviced)

#### See Also

[\\_int\\_install\\_isr](#)

#### Example

See [\\_int\\_get\\_kernel\\_isr](#).

## 2.1.37 `_int_get_kernel_isr`

Gets a pointer to the kernel ISR for the vector number. The kernel ISR depends on the PSP.

### Prototype

```
source\psp\cpu_family\int_gkis.c
INT_KERNEL_ISR_FPTR _int_get_kernel_isr(
    uint32 vector)
```

### Parameters

*vector* [IN] — Vector number whose kernel ISR is being requested

### Returns

- Pointer to the kernel ISR (success)
- *NULL* (failure)

### Traits

On failure, calls `_task_set_error()` to set the task error code

### See Also

[\\_int\\_kernel\\_isr](#)

[\\_int\\_install\\_kernel\\_isr](#)

### Description

The returned kernel ISR is either the default kernel ISR or an ISR that the application installed with `_int_install_kernel_isr()`.

### Example

Get various ISR info for a specific interrupt.

```
#define      SPECIFIC_INTERRUPT  3
_mqx_uint   depth;
void        *vector_tbl;
INT_KERNEL_ISR_FPTR kernel_isr;
INT_ISR_FPTR my_isr;
void        *my_isr_data;

kernel_isr  = _int_get_kernel_isr(SPECIFIC_INTERRUPT);
my_isr      = _int_get_isr(SPECIFIC_INTERRUPT);
my_isr_dat  = _int_get_isr_data(SPECIFIC_INTERRUPT);
depth       = _int_get_isr_depth();
vector_tbl  = _int_get_vector_table();
```

### 2.1.38 `_int_get_previous_vector_table`

Gets the address of the interrupt vector table that MQX RTOS might have created when it started.

#### Prototype

```
source\psp\cpu_family\int_pvta.c  
_psp_code_addr  
_int_get_previous_vector_table(void)
```

#### Parameters

None

#### Returns

Address of the interrupt vector table that MQX RTOS creates when it starts

#### See Also

[\\_int\\_get\\_vector\\_table](#)

[\\_int\\_set\\_vector\\_table](#)

#### Description

The function is useful if you are installing third-party debuggers or monitors.

## 2.1.39 `_int_get_vector_table`

Gets the address of the current interrupt vector table. The function depends on the PSP.

### Prototype

```
source\psp\cpu_family\int_vtab.c  
_psp_code_addr _int_get_vector_table(void)
```

### Parameters

None

### Returns

Address of the current interrupt vector table

### See also

[\\_int\\_set\\_vector\\_table](#)

[\\_int\\_get\\_previous\\_vector\\_table](#)

### Example

See `_int_get_kernel_isr()`.

## 2.1.40 `_int_install_default_isr`

Installs an application-provided default ISR.

### Prototype

```
source\kernel\int.c  
INT_ISR_FPTR _int_install_default_isr(  
    INT_ISR_FPTR default_isr)
```

### Parameters

*default\_isr [IN]* — New default ISR

### Returns

Pointer to the default ISR before the function was called

### See Also

[\\_int\\_get\\_default\\_isr](#)

[\\_int\\_install\\_isr](#)

### Description

MQX RTOS uses the application-provided default ISR for all interrupts for which the application has not installed an application ISR. The ISR handles all unhandled and unexpected interrupts.

## 2.1.41 `_int_install_exception_isr`

Installs the MQX RTOS-provided `_int_exception_isr()` as the default ISR for unhandled interrupts and exceptions.

### Prototype

```
source\kernel\int.c  
INT_ISR_FPTR _int_install_exception_isr(void)
```

### Parameters

None

### Returns

Pointer to the default exception handler before the function was called

### See Also

[\\_int\\_get\\_default\\_isr](#)

## 2.1.42 `_int_install_isr`

Installs the ISR.

### Prototype

```
source\kernel\int.c
INT_ISR_FPTR _int_install_isr(
    _mqx_uint      vector,
    INT_ISR_FPTR   isr_ptr,
    void           *isr_data)
```

### Parameters

*vector* [IN] — Vector number (not the offset) of the interrupt

*isr\_ptr* [IN] — Pointer to the ISR

*isr\_data* [IN] — Pointer to the data to be passed as the first parameter to the ISR when an interrupt occurs and the ISR runs

### Returns

- Pointer to the ISR for the vector before calling the function (success)
- *NULL* (failure)

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### Task Error Codes

Error Code	Description
MQX_COMPONENT_DOES_NOT_EXIST	Interrupt component is not created.
MQX_INVALID_VECTORED_INTERRUPT	Vector is outside the valid range of interrupt numbers.

### See Also

[\\_int\\_get\\_default\\_isr](#)

[\\_int\\_install\\_default\\_isr](#)

[\\_int\\_get\\_isr\\_data](#)

[\\_int\\_set\\_isr\\_data](#)

[\\_int\\_get\\_isr](#)

[\\_task\\_set\\_error](#)

### Description

The application defines the ISR data, which can be a constant or a pointer to a memory block from `_mem_alloc()`.



MQX RTOS catches all hardware interrupts in the range that the BSP defined and saves the context of the active task. For most interrupts, MQX RTOS calls the ISR that is stored in the interrupt vector table at the location identified by its interrupt vector number.

### Example

In the initialization of a serial I/O handler, install the same ISR for the four channels, assigning a logical interrupt to each one through the third parameter of **\_int\_install\_isr()**.

```
_int_install_isr(SIO_INTERRUPT_A, SIO_isr, LOG_INTA);  
_int_install_isr(SIO_INTERRUPT_B, SIO_isr, LOG_INTB);  
_int_install_isr(SIO_INTERRUPT_C, SIO_isr, LOG_INTC);  
_int_install_isr(SIO_INTERRUPT_D, SIO_isr, LOG_INTD);
```

## 2.1.43 `_int_install_kernel_isr`

Installs the kernel ISR. The kernel ISR depends on the PSP.

### Prototype

```
source\psp\cpu_family\int_kisr.c
INT_KERNEL_ISR_FPTR _int_install_kernel_isr(
    _mqx_uint          vector,
    INT_KERNEL_ISR_FPTR isr_ptr)
```

### Parameters

*vector* [IN] — Vector where the kernel ISR is to be installed  
*isr\_ptr* [IN] — Pointer to the ISR to install in the vector table

### Returns

- Pointer to the kernel ISR for the vector before the function was called (success)
- *NULL* (failure)

### See Also

[`\_int\_kernel\_isr`](#)

[`\_int\_get\_kernel\_isr`](#)

### Description

Some real-time applications need special event handling to occur outside the scope of MQX RTOS. The need might arise that the latency in servicing an interrupt be less than the MQX RTOS interrupt latency. If this is the case, an application can use `_int_install_kernel_isr()` to bypass MQX RTOS and let the interrupt be serviced immediately.

Because the function returns the previous kernel ISR, applications can temporarily install an ISR or chain ISRs so that each new one calls the one installed before it.

A kernel ISR must save the registers that it needs and must service the hardware interrupt. When the kernel ISR is finished, it must restore the registers and perform a return-from-interrupt instruction.

A kernel ISR cannot call MQX RTOS functions. However, it can put data in global data, which a task can access.

### NOTE

The function is not available for all PSPs.

## 2.1.44 `_int_install_unexpected_isr`

Installs the MQX RTOS-provided unexpected ISR, `_int_unexpected_isr()`, for all interrupts that do not have an application-installed ISR.

### Prototype

```
source\kernel\int.c  
INT_ISR_FPTR _int_install_unexpected_isr(void)
```

### Parameters

None

### Returns

Pointer to the unexpected interrupt ISR before the function was called

### See Also

[`\_int\_install\_exception\_isr`](#)

[`\_int\_unexpected\_isr`](#)

### Description

The installed ISR writes the cause of the unexpected interrupt to the standard I/O stream.

## 2.1.45 `_int_kernel_isr`

Default kernel ISR that MQX RTOS calls to intercept all interrupts.

### Prototype

```
\source\psp\

```

### Parameters

None

### Returns

None

### See Also

[`\_int\_install\_kernel\_isr`](#)

[`\_int\_install\_isr`](#)

### Description

The ISR is usually written in assembly language.

It does the following:

- Saves enough registers so that an ISR written in C can be called.
- If the current stack is not the interrupt stack, switches to the interrupt stack.
- Creates an interrupt context on the stack. This lets functions written in C properly access the task error code, `_int_enable()`, and `_int_disable()`.
- Checks for ISRs. If they have not been installed or if the ISR number is outside the range of installed ISRs, calls `DEFAULT_ISR`.
- If ISRs have been installed and if an application C-language ISR has not been installed for the vector, calls `DEFAULT_ISR`.
- After returning from the C-language ISR, does the following:
  - if this is a nested ISR, performs an interrupt return instruction.
  - if the current task is still the highest-priority ready task, performs an interrupt return instruction.
  - otherwise, saves the full context for the current task and enters the scheduler

## 2.1.46 `_int_set_exception_handler`

Sets the ISR exception handler for the interrupt vector.

### Prototype

```
source\kernel\int.c
INT_EXCEPTION_FPTR _int_set_exception_handler
    (_mqx_uint      vector,
     INT_EXCEPTION_FPTR error_handler_address)
```

### Parameters

*vector* [IN] — Interrupt vector that this exception handler is for  
*error\_handler\_address* [IN] — Pointer to the exception handler

### Returns

- Pointer to the exception handler before the function was called (success)
- *NULL* (failure)

### Traits

On failure, does not install the exception handler and calls `_task_set_error()` to set the task error code

### See Also

[\\_int\\_get\\_exception\\_handler](#)

[\\_int\\_exception\\_isr](#)

[\\_task\\_set\\_error](#)

### Description

The function sets the exception handler for an ISR. When an exception (unhandled interrupt) occurs while the ISR is running, MQX RTOS calls the exception handler and terminates the ISR.

An application should install `_int_exception_isr()` as the MQX RTOS default ISR.

The returned exception handler is either the default handler or one that the application previously installed with `_int_set_exception_handler()`.

## 2.1.47 `_int_set_isr_data`

Sets the data associated with the interrupt.

### Prototype

```
source\kernel\int.c  
void *_int_set_isr_data(  
    _mqx_uint vector,  
    void      *data)
```

### Parameters

*vector* [IN] — Interrupt vector that the data is for

*data* [IN] — Data that MQX RTOS passes to the ISR as its first parameter

### Returns

- ISR data before the function was called (success)
- *NULL* (failure)

### Traits

On failure, calls `_task_set_error()` to set the task error code

### See also

[\\_int\\_get\\_isr](#)

[\\_int\\_get\\_isr\\_data](#)

## 2.1.48 `_int_set_vector_table`

Changes the location of the interrupt vector table.

### Prototype

```
source\psp\cpu_family\int_vtab.c  
_psp_code_addr _int_set_vector_table(  
    _psp_code_addr new)
```

### Parameters

new [IN] — Address of the new interrupt vector table

### Returns

Address of the previous vector table

### Traits

Behavior depends on the BSP and the PSP

### See Also

[`\_int\_get\_vector\_table`](#)

[`\_int\_get\_previous\_vector\_table`](#)

## 2.1.49 `_int_unexpected_isr`

An MQX RTOS-provided default ISR for unhandled interrupts. The function depends on the PSP.

### Prototype

```
source\psp\cpu_family\int_unx.c
void _int_unexpected_isr(
    void *parameter)
```

### Parameters

parameter [IN] — Parameter passed to the default ISR

### Returns

None

### Traits

Blocks the active task

### See also

[\\_int\\_install\\_unexpected\\_isr](#)

### Description

The function changes the state of the active task to **UNHANDLED\_INT\_BLOCKED** and blocks the task.

The function uses the default I/O channel to display at least:

- vector number that caused the unhandled exception
- task ID and task descriptor of the active task

Depending on the PSP, more information might be displayed.

### CAUTION

Since the ISR uses `printf()` to display information to the default I/O channel, default I/O must not be on a channel that uses interrupt-driven I/O or the debugger.



## 2.1.50 `_psp_push_fp_context`

Store floating point context (registers).

### Prototype

```
void void _psp_push_fp_context(void)
```

### Parameters

None

### Returns

None

### See Also

[\\_psp\\_pop\\_fp\\_context](#)

### Description

This function must be use in each interrupt handler which use FPU extension. Call it at the beginning, before any other operations to protect system against to corruption of context.

## 2.1.51 `_psp_pop_fp_context`

Restore floating point context (registers) in interrupt handler.

### Prototype

```
void void _psp_pop_fp_context(void)
```

### Parameters

None

### Returns

None

### See Also

[\\_psp\\_push\\_fp\\_context](#)

### Description

This function must be use in each interrupt handler which use FPU extension. Call it at the end, when every operations with the floatpoints finished.

## 2.1.52 `_ipc_add_io_ipc_handler`

Add an IPC handler for the I/O component.

### Prototype

```
source\kernel\ipc.c
#include <ipc.h>
_mqx_uint _ipc_add_io_ipc_handler(
    IPC_HANDLER_FPTR handler,
    _mx_uint component)
```

### Parameters

*handler* [IN] — Pointer to the function that MQX RTOS calls when it receives an IPC request for the component

*component* [IN] — I/O component that the handler is for (see description)

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_IPC_SERVICE_NOT_AVAILABLE	IPC server has not been started.

### See Also

[\\_ipc\\_add\\_ipc\\_handler](#)

### Description

The IPC task calls the function when an IPC message for the specified I/O component is received. The IPC task calls the function once for each component.

The parameter *component* can be one of:

- **IO\_CAN\_COMPONENT**
- **IO\_HDLC\_COMPONENT**
- **IO\_LAPB\_COMPONENT**
- **IO\_LAPD\_COMPONENT**
- **IO\_MFS\_COMPONENT**
- **IO\_PPP\_COMPONENT**
- **IO\_RTCS\_COMPONENT**
- **IO\_SDLN\_COMPONENT**
- **IO\_SNMP\_COMPONENT**
- **IO\_SUBSYSTEM\_COMPONENT**

## 2.1.53 `_ipc_add_ipc_handler`

Adds an IPC handler for the MQX RTOS component.

### Prototype

```
source\kernel\ipc.c
#include <ipc.h>
_mqx_uint _ipc_add_ipc_handler(
    IPC_HANDLER_FPTR handler,
    _mx_uint component)
```

### Parameters

*handler* [IN] — Pointer to the function that MQX RTOS calls when it receives an IPC request for the component

*component* [IN] — MQX RTOS component that the handler is for (see description)

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_IPC_SERVICE_NOT_AVAILABLE	IPC server has not been started.

### See Also

[\\_ipc\\_add\\_io\\_ipc\\_handler](#)

### Description

The IPC task calls the function when an IPC message for the specified MQX RTOS component is received. The IPC task calls the function once for each component.

The parameter *component* can be one of:

- **KERNEL\_EVENTS**
- **KERNEL\_IPC**
- **KERNEL\_IPC\_MSG\_ROUTING**
- **KERNEL\_LOG**
- **KERNEL\_LWLOG**
- **KERNEL\_MESSAGES**
- **KERNEL\_MUTEXES**
- **KERNEL\_NAME\_MANAGEMENT**
- **KERNEL\_PARTITIONS**
- **KERNEL\_SEMAPHORES**
- **KERNEL\_TIMER**
- **KERNEL\_WATCHDOG**

## 2.1.54 `_ipc_msg_processor_route_exists`

Gets a pointer to the route for the processor.

### Prototype

```
source\kernel\ipc.c
#include <ipc.h>
void *_ipc_msg_processor_route_exists(
    _processor_number  proc_number)
```

### Parameters

*proc\_number* [IN] — Processor number to check for a route

### Returns

- Pointer to the route (a route exists)
- *NULL* (a route does not exist)

### See Also

[`\_ipc\_msg\_route\_add`](#)

[`\_ipc\_msg\_route\_remove`](#)

## 2.1.55 `_ipc_msg_route_add`

Adds a route to the message routing table.

### Prototype

```
source\kernel\ipc.c
#include <ipc.h>
_mqx_uint _ipc_msg_route_add(
    _processor_number min_proc_number,
    _processor_number max_proc_number,
    _queue_number     queue)
```

### Parameters

*min\_proc\_number [IN]* — Minimum processor number in the range

*max\_proc\_number [IN]* — Maximum processor number in the range

*queue [IN]* — Queue number of the IPC to use for processor numbers in the range

### Returns

- MQX\_OK
- Errors
  - MQX\_COMPONENT\_DOES\_NOT\_EXIST
  - MQX\_INVALID\_PROCESSOR\_NUMBER
  - MSGQ\_INVALID\_QUEUE\_ID
  - IPC\_ROUTE\_EXISTS

### See Also

[\\_ipc\\_msg\\_route\\_remove](#)

[\\_ipc\\_msg\\_processor\\_route\\_exists](#)

[IPC\\_ROUTING\\_STRUCT](#)

### Description

The IPC component must first be created.

## 2.1.56 `_ipc_msg_route_remove`

Removes a route from the message routing table.

### Prototype

```
source\kernel\ipc.c
#include <ipc.h>
_mqx_uint _ipc_msg_route_remove(
    _processor_number  min_proc_number,
    _processor_number  max_proc_number,
    _queue_number      queue)
```

### Parameters

- *min\_proc\_number [IN]* — Minimum processor number in the range
- *max\_proc\_number [IN]* — Maximum processor number in the range
- *queue [IN]* — Queue number of the IPC to remove

### Returns

- MQX\_OK
- Errors
  - MQX\_COMPONENT\_DOES\_NOT\_EXIST
  - MQX\_INVALID\_PROCESSOR\_NUMBER

### See Also

[\\_ipc\\_msg\\_route\\_add](#)

[\\_ipc\\_msg\\_processor\\_route\\_exists](#)

### IPC\_ROUTING\_STRUCT

### Description

The IPC component must first be installed.

## 2.1.57 `_ipc_pcb_init`

Initializes an IPC for a PCB driver.

### Prototype

```
source\kernel\ipc.c
_mqx_uint _ipc_pcb_init(
    IPC_PROTOCOL_INIT_STRUCT_PTR init_ptr,
    void *info_ptr)
```

### Parameters

*init\_ptr* [IN] — Pointer to an IPC protocol initialization structure (**IPC\_PROTOCOL\_INIT\_STRUCT**)

*info\_ptr* [IN] — Pointer to an IPC protocol information structure

### Returns

- MQX\_OK (success)
- IPC\_LOOPBACK\_INVALID\_QUEUE (failure)

### See Also

[IPC\\_PCB\\_INIT\\_STRUCT](#)

[IPC\\_PROTOCOL\\_INIT\\_STRUCT](#)

### Description

The function is used in structure of type **IPC\_PROTOCOL\_STRUCT** to initialize an IPC that uses the PCB device drivers.

The **IPC\_PROTOCOL\_INIT\_DATA** field in **IPC\_PROTOCOL\_INIT\_STRUCT** must point to a structure of type **IPC\_PCB\_INIT\_STRUCT**.

### Example

Initialize an IPC for the PCB.

```
IPC_PCB_INIT_STRUCT pcb_init =
{
    /* IO_PORT_NAME */           "pcb_mqxa_ittyb:",
    /* DEVICE_INSTALL? */        _io_pcb_mqxa_install,
    /* DEVICE_INSTALL_PARAMETER*/ (void*)&pcb_mqxa_init,
    /* IN_MESSAGES_MAX_SIZE */   sizeof( THE_MESSAGE ),
    /* IN_MESSAGES_TO_ALLOCATE */ 8,
    /* IN_MESSAGES_TO_GROW */     8,
    /* IN_MESSAGES_MAX_ALLOCATE */ 16,
    /* OUT_PCBS_INITIAL */        8,
    /* OUT_PCBS_TO_GROW */        8,
```



```
/* OUT_PCBS_MAX */          16
};

IPC_PROTOCOL_INIT_STRUCT _ipc_init_table[] =
{
    { _ipc_pcb_init, &pcb_init, "Pcb_to_test2", QUEUE_TO_TEST2 },
    { NULL, NULL, NULL, 0}
};
```

## 2.1.58 `_ipc_task`

Task that initializes IPCs and processes remote service requests.

### Prototype

```
source\kernel\ipc.c
#include <ipc.h>
void _ipc_task(
    uint32_t parameter)
```

### Parameters

*parameter [IN]* — pointer to the IPC\_INIT\_STRUCT (task creation parameter)

### Returns

None

### See Also

### [IPC\\_INIT\\_STRUCT](#)

### Description

For applications to use the IPC component, the task must be either specified in the task template list as an autostart task or explicitly created.

The task installs the IPCs that are listed in the IPC initialization structure. Pointer to this initialization structure (IPC\_INIT\_STRUCT\_PTR) is provided as the creation parameter, otherwise default IPC\_INIT\_STRUCT is used (*\_default\_ipc\_init*). When the initialization is finished it waits for service requests from remote processors.

### Example

The task template causes MQX RTOS to create IPC Task.

```
static const IPC_INIT_STRUCT ipc_init = {
    ipc_routing_table,
    ipc_init_table
};

TASK_TEMPLATE_STRUCT  MQX_template_list[] =
{
    { IPC_TTN,    _ipc_task,    IPC_DEFAULT_STACK_SIZE, 6L,
      "_ipc_task", MQX_AUTO_START_TASK, (uint32_t)&ipc_init, 0L },
    ...
    { 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L }
};
```

## 2.1.59 `_klog_control`

Controls logging in kernel log.

### Prototype

```
source\kernel\klog.c
#include <klog.h>
void _klog_control(
    uint32_t bit_mask,
    bool      set_bits)
```

### Parameters

*bit\_mask* [IN] — Which bits of the kernel log control variable to modify

*set\_bits* [IN] – TRUE (bits that are set in *bit\_mask* are set in the control variable)

FALSE (bits that are set in *bit\_mask* are cleared in the control variable)

### Returns

None

### See Also

[\\_klog\\_create, \\_klog\\_create\\_at](#)

[\\_klog\\_disable\\_logging\\_task, \\_klog\\_enable\\_logging\\_task](#)

[\\_lwlog\\_create\\_component](#)

### Description

The application must first create kernel log with `_klog_create()`.

The function `_klog_control()` sets or clears bits in the kernel log control variable, which MQX RTOS uses to control logging. To select which functions to log, set combinations of bits in the **KLOG\_FUNCTIONS\_ENABLED** flag for the *bit\_mask* parameter.

MQX RTOS logs to kernel log only if **KLOG\_ENABLED** is set in *bit\_mask*.

### NOTE

To use kernel logging, MQX RTOS must be configured at compile time with `MQX_KERNEL_LOGGING` set to 1. For information on configuring MQX RTOS, see the MQX RTOS User's Guide.

If this bit is set:	MQX RTOS:
KLOG_ENABLED (log MQX RTOS services)	Logs to kernel log

If combinations of these bits are set:	Select combinations from:
<b>KLOG_FUNCTIONS_ENABLED</b> (log calls to MQX RTOS component APIs)	KLOG_TASKING_FUNCTIONS KLOG_ERROR_FUNCTIONS KLOG_MESSAGE_FUNCTIONS KLOG_INTERRUPT_FUNCTIONS KLOG_MEMORY_FUNCTIONS KLOG_TIME_FUNCTIONS KLOG_EVENT_FUNCTIONS KLOG_NAME_FUNCTIONS KLOG_MUTEX_FUNCTIONS KLOG_SEMAPHORE_FUNCTIONS KLOG_WATCHDOG_FUNCTIONS KLOG_PARTITION_FUNCTIONS KLOG_IO_FUNCTIONS
<b>KLOG_TASK_QUALIFIED</b> (log specific tasks only)	For each task to log, call one of: <u><b>_klog_disable_logging_task()</b></u> <u><b>_klog_enable_logging_task()</b></u>
<b>KLOG_INTERRUPTS_ENABLED</b> (log interrupts) <b>KLOG_SYSTEM_CLOCK_INT_ENABLED</b> (log periodic timer interrupts) <b>KLOG_CONTEXT_ENABLED</b> (log context switches)	—

### Example

Enable logging to kernel log for all calls that this task and its creator make to the semaphore component API.

```

_log_create_component();
_klog_create(4096, LOG_OVERWRITE);

/* Clear all the control bits and then set particular ones: */
_klog_control(0xffffffff, FALSE);
_klog_control(
    KLOG_ENABLED |
    KLOG_TASK_QUALIFIED |
    KLOG_FUNCTIONS_ENABLED | KLOG_SEMAPHORE_FUNCTIONS,
    TRUE);

/* Enable task logging for this task and its creator: */
_klog_enable_logging_task(_task_get_id());
_klog_enable_logging_task(_task_get_creator());
...
/* Disable task logging for this task: */
_klog_disable_logging_task(_task_get_id());

/* Display and delete all entries in kernel log: */
while (_klog_display()) {
}

```

## 2.1.60 `_klog_create`, `_klog_create_at`

<code>_klog_create()</code>	Creates kernel log.
<code>_klog_create_at()</code>	Creates kernel log at the specific location

### Prototype

```
source\kernel\klog.c
#include <log.h>
#include <klog.h>
_mqx_uint _klog_create(
    _mx_uint max_size,
    _mx_uint flags)

source\kernel\klog.c
#include <log.h>
#include <klog.h>
_mqx_uint _klog_create_at(
    _mx_uint max_size,
    _mx_uint flags,
    void *where)
```

### Parameters

*max\_size* [IN] — Maximum size (in **mqx\_max\_types**) of the data to be stored

*flags* [IN] — One of the following:

**LOG\_OVERWRITE** (when the log is full, write new entries over oldest entries)

0 (when the log is full, write no more entries; the default)

*where* [IN] — Where to create the log

### Returns

- **MQX\_OK**
- Errors

Errors from <code>_lwlog_create()</code>	Description
<b>LOG_EXISTS</b>	Kernel log already exists.
<b>MQX_INVALID_COMPONENT_BASE</b>	Log component data is not valid.
<b>MQX_OUT_OF_MEMORY</b>	MQX RTOS cannot allocate memory for kernel log.

### See Also

[\\_klog\\_control](#)

[\\_klog\\_disable\\_logging\\_task](#), [\\_klog\\_enable\\_logging\\_task](#)

[\\_lwlog\\_create\\_component](#)

## [\\_lwlog\\_create, \\_lwlog\\_create\\_at](#)

### Description

If the log component is not created, MQX RTOS creates it. MQX RTOS uses lightweight log number 0 as kernel log.

Each entry in kernel log contains MQX RTOS-specific data, a timestamp (in absolute time), a sequence number, and information specified by **\_klog\_control()**.

The MQX RTOS Embedded Performance Tool uses kernel log to analyze how the application operates and uses resources.

### Example

See [\\_klog\\_control\(\)](#).

## 2.1.61 `_klog_disable_logging_task`, `_klog_enable_logging_task`

<code>_klog_disable_logging_task()</code>	Disables kernel logging for the task.
<code>_klog_enable_logging_task()</code>	Enables kernel logging for the task.

### Prototype

```
source\kernel\klog.c
#include <klog.h>
void _klog_disable_logging_task(
    _task_id task_id)
```

### Parameters

*task\_id* [IN] — Task ID of the task for which kernel logging is to be disabled or enabled

### Returns

None

### Traits

Disables and enables interrupts

### See Also

[\\_klog\\_control](#)

### Description

If the application calls `_klog_control()` with **KLOG\_TASK\_QUALIFIED**, it must call `_klog_enable_logging_task()` for each task for which it wants to log information.

The application disables logging by calling `_klog_disable_logging_task()` for each task for which it wants to stop logging. If the application did not first enable logging for the task, MQX RTOS ignores the request.

### NOTE

To use kernel logging, MQX RTOS must be configured at compile time with `MQX_KERNEL_LOGGING` set to 1. For information on configuring MQX RTOS, see MQX RTOS User's Guide.

### Example

See `_klog_control()`.

## 2.1.62 `_klog_display`

Displays the oldest entry in kernel log and delete the entry.

### Prototype

```
source\kernel\klog.c  
bool _klog_display(void)
```

### Parameters

None

### Returns

- *TRUE* (entry is found and displayed)
- *FALSE* (entry is not found)

### Traits

Depending on the low-level I/O used, the calling task might block and MQX RTOS might perform a dispatch operation.

### See Also

[\\_klog\\_control](#)

[\\_klog\\_create](#), [\\_klog\\_create\\_at](#)

### Description

The function prints the oldest entry in kernel log to the default output stream of the current task and deletes the entry.

### Example

See `_klog_control()`.



## 2.1.63 `_klog_get_interrupt_stack_usage`

Gets the size of the interrupt stack and the total amount of it used.

### Prototype

```
source\kernel\klog.c
_mqx_uint _klog_get_interrupt_stack_usage(
    _mem_size_ptr  stack_size_ptr,
    _mem_size_ptr  stack_used_ptr)
```

### Parameters

*stack\_size\_ptr* [OUT] — Where to write the size (in single-addressable units) of the stack  
*stack\_used\_ptr* [OUT] — Where to write the amount (in single-addressable units) of stack used

### Returns

- MQX\_OK (success)
- MQX\_INVALID\_CONFIGURATION (failure: compile-time configuration option MQX\_MONITOR\_STACK is not set)

### See Also

[\\_klog\\_get\\_task\\_stack\\_usage](#)

[\\_klog\\_show\\_stack\\_usage](#)

### Description

The amount used is a highwater mark—the highest amount of interrupt stack that the application has used so far. It shows only how much of the stack has been written to at this point. If the amount is 0, the interrupt stack is not large enough.

### NOTE

To use kernel logging, MQX RTOS must be configured at compile time with MQX\_MONITOR\_STACK set to 1. For information on configuring MQX RTOS, see MQX RTOS User's Guide.

### Example

Determine the state of all stacks.

```
_mem_size  stack_size;
_mem_size  stack_used;
_mqx_uint  return_value;
...
_klog_get_interrupt_stack_usage(&stack_size, &stack_used);
printf("Interrupt stack size: 0x%x, Stack used: 0x%x",
    stack_size, stack_used);

/* Get stack usage for this task: */
_klog_get_task_stack_usage(_task_get_id(), &stack_size, &stack_used);
printf("Task ID: 0x%lx, Stack size: 0x%x, Stack used: 0x%x",
    _task_get_id(), stack_size, stack_used);
```

```
...  
/* Display all stack usage: */  
_klog_show_stack_usage();
```

## 2.1.64 `_klog_get_task_stack_usage`

Gets the stack size for the task and the total amount of it that the task has used.

### Prototype

```
source\kernel\klog.c
_mqx_uint _klog_get_task_stack_usage(
    _task_id      task_id,
    _mem_size_ptr stack_size_ptr,
    _mem_size_ptr stack_used_ptr)
```

### Parameters

*task\_id* [IN] — Task ID of the task to display

*stack\_size\_ptr* [OUT] — Where to write the size (in single-addressable units) of the stack

*stack\_used\_ptr* [OUT] — Where to write the amount (in single-addressable units) of stack used

### Returns

- **MQX\_OK** (success)
- Errors (failure)

Error	Description
MQX_INVALID_CONFIGURATION	Compile-time configuration option MQX_MONITOR_STACK is not set.
MQX_INVALID_TASK_ID	<i>task_id</i> is not valid.

### See Also

[\\_klog\\_get\\_interrupt\\_stack\\_usage](#)

[\\_klog\\_show\\_stack\\_usage](#)

### NOTE

To use kernel logging, MQX RTOS must be configured at compile time with MQX\_MONITOR\_STACK set to 1. For information on configuring MQX RTOS, see MQX RTOS User's Guide.

### Description

The amount used is a highwater mark—the highest amount of stack that the task has used so far. It might not include the amount that the task is currently using. If the amount is 0, the stack is not large enough.

### Example

See `_klog_get_interrupt_stack_usage()`.

## 2.1.65 `_klog_show_stack_usage`

Displays the amount of interrupt stack used and the amount of stack used by each task.

### Prototype

```
source\kernel\klog.c  
void _klog_show_stack_usage(void)
```

### Parameters

None

### Returns

None

### Traits

Depending on the low-level I/O used, the calling task might block and MQX RTOS might perform a dispatch operation.

### See Also

[\\_klog\\_get\\_interrupt\\_stack\\_usage](#)

[\\_klog\\_get\\_task\\_stack\\_usage](#)

### Description

The function displays the information on the standard output stream for the calling task.

### NOTE

To use kernel logging, MQX RTOS must be configured at compile time with `MQX_MONITOR_STACK` set to 1. For information on configuring MQX RTOS, see MQX RTOS User's Guide.

### Example

See `_klog_get_interrupt_stack_usage()`.

## 2.1.66 `_log_create`

Creates the log.

### Prototype

```
source\kernel\log.c
#include <log.h>
_mqx_uint _log_create(
    _mx_uint    log_number,
    _mx_uint    max_size,
    uint32_t    flags)
```

### Parameters

*log\_number* [IN] — Log number to create (0 through 15)

*max\_size* [IN] — Maximum number of **\_mx\_uint**s to store in the log (includes **LOG\_ENTRY\_STRUCT** headers)

*flags* [IN] — One of the following:

**LOG\_OVERWRITE** (when the log is full, write new entries over oldest ones)

0 (when the log is full, do not write entries)

### Returns

- **MQX\_OK**
- Errors

Error	Description
LOG_EXISTS	Log <i>log_number</i> has already been created.
MQX_OUT_OF_MEMORY	MQX RTOS is out of memory.

### Traits

Creates the log component if it was not created

### See Also

[\\_log\\_create\\_component](#)

[\\_log\\_destroy](#)

[\\_log\\_read](#)

[\\_log\\_write](#)

[LOG\\_ENTRY\\_STRUCT](#)

### Description

Each entry in the log contains application-specified data, a timestamp (in absolute time), and a sequence number.

## Example

```
#define LOG_SIZE 2048 /* Number of long words in the log */
_mqx_uint my_log = 2;
unsigned char log_entry[sizeof(LOG_ENTRY_STRUCT) + DATA_SIZE];
...
if (_log_create(my_log, LOG_SIZE, LOG_OVERWRITE) != MQX_OK) {
    /* The function failed.*/
}
while (quit == FALSE) {
    result = _log_write(my_log, 3, EVENT_B, i, j);
    ...
    result = _log_read(my_log, LOG_READ_OLDEST_AND_DELETE,
        DATA_SIZE, &log_entry);
}
...
log_destroy(my_log);
```

## 2.1.67 `_log_create_component`

Creates the log component.

### Prototype

```
source\kernel\log.c
#include <log.h>
_mqx_uint _log_create_component(void)
```

### Parameters

None

### Returns

- MQX\_OK (success)
- MQX\_OUT\_OF\_MEMORY (failure)

### Traits

Disables and enables interrupts

### See Also

[\\_log\\_create](#)

### Description

The log component provides a maximum of 16 separately configurable user logs (log numbers 0 through 15).

An application subsequently creates user logs with `_log_create()`.

## 2.1.68 `_log_destroy`

Destroys the log.

### Prototype

```
source\kernel\log.c
#include <log.h>
_mqx_uint _log_destroy(
    _mx_uint log_number)
```

### Parameters

*log\_number* [IN] — Log number of a previously created log

### Returns

- MQX\_OK
- Errors

Error	Description
LOG_DOES_NOT_EXIST	<i>log_number</i> was not previously created.
LOG_INVALID	<i>log_number</i> is out of range.
MQX_COMPONENT_DOES_NOT_EXIST	Log component is not created.
MQX_INVALID_COMPONENT_HANDLE	Log component data is not valid.

### See Also

[\\_log\\_create](#)

[\\_log\\_create\\_component](#)

### Example

See `_log_create()`.



## 2.1.69 `_log_disable`, `_log_enable`

<code>_log_disable()</code>	Stops logging to the log.
<code>_log_enable()</code>	Starts logging to the log.

### Prototype

```
source\kernel\log.c
#include <log.h>
_mqx_uint _log_disable(
    _mx_uint log_number)

_mqx_uint _log_enable(
    _mx_uint log_number)
```

### Parameters

*log\_number* [IN] — Log number of a previously created log

### Returns

- MQX\_OK
- Errors

Error	Description
LOG_DOES_NOT_EXIST	<i>log_number</i> was not created.
LOG_INVALID	<i>log_number</i> is out of range.
MQX_COMPONENT_DOES_NOT_EXIST	Log component is not created.
MQX_INVALID_COMPONENT_HANDLE	Log component data is not valid.

### See Also

[\\_log\\_read](#)

[\\_log\\_reset](#)

[\\_log\\_write](#)

### Description

A task can enable a log that has been disabled.

### Example

See `_log_reset()`.

## 2.1.70 `_log_read`

Reads the information in the log.

### Prototype

```
source\kernel\log.c
#include <log.h>
_mqx_uint _log_read(
    _mqx_uint      log_num,
    _mqx_uint      read_type,
    _mqx_uint      size,
    LOG_ENTRY_STRUCT_PTR entry_ptr)
```

### Parameters

*log\_num* [IN] — Log number of a previously created log

*read\_type* [IN] — Type of read operation (see description)

*size* [IN] — Maximum number of **\_mqx\_uints** (not including the entry header) to be read from an entry

*entry\_ptr* [IN] — Where to write the log entry (any structure that starts with **LOG\_STRUCT** or **LOG\_ENTRY\_STRUCT**)

### Returns

- **MQX\_OK**
- Errors

Error	Description
LOG_DOES_NOT_EXIST	<i>log_number</i> was not created.
LOG_ENTRY_NOT_AVAILABLE	Log entry is not available.
LOG_INVALID	<i>log_number</i> is out of range.
LOG_INVALID_READ_TYPE	<i>read_type</i> is not valid.
MQX_COMPONENT_DOES_NOT_EXIST	Log component is not created.
MQX_INVALID_COMPONENT_HANDLE	Log component data is not valid.
MQX_INVALID_POINTER	<i>entry_ptr</i> is <i>NULL</i> .

### See Also

[\\_log\\_create](#)

[\\_log\\_write](#)

**LOG\_STRUCT**

**LOG\_ENTRY\_STRUCT**

## Description

read_type	Returns this entry in the log:
LOG_READ_NEWEST	Newest
LOG_READ_NEXT	Next one after the previous one read (must be used with <b>LOG_READ_OLDEST</b> )
LOG_READ_OLDEST	Oldest
LOG_READ_OLDEST_AND_DELETE	Oldest and deletes it

## Example

See [\\_log\\_create\(\)](#).

## 2.1.71 `_log_reset`

Resets the log to its initial state (remove all entries).

### Prototype

```
source\kernel\log.c
#include <log.h>
_mqx_uint _log_reset(
    _mx_uint log_number)
```

### Parameters

*log\_number* [IN] — Log number of a previously created log

### Returns

- MQX\_OK
- Errors

Error	Description
LOG_DOES_NOT_EXIST	<i>log_number</i> was not created.
LOG_INVALID	<i>log_number</i> is out of range.
MQX_COMPONENT_DOES_NOT_EXIST	Log component is not created.
MQX_INVALID_COMPONENT_HANDLE	Log component data is not valid.

### See Also

[\\_log\\_disable](#), [\\_log\\_enable](#)

### Example

```
_mx_uint my_log = 2;
...
result = _log_disable(my_log);
result = _log_reset(my_log);
if (result != MQX_OK) {
    /* The function failed. */
    return result;
}
result = _log_enable(my_log);
...
```

## 2.1.72 `_log_test`

Tests the log component.

### Prototype

```
source\kernel\log.c
#include <log.h>
_mqx_uint _log_test(
    _mx_uint *log_error_ptr)
```

### Parameters

*log\_error\_ptr* [OUT] — Pointer to the log in error (*NULL* if no error is found)

### Returns

See description

### Traits

Disables and enables interrupts

### See Also

[\\_log\\_create\\_component](#)

[\\_log\\_create](#)

### Description

Return value	<i>*log_error_ptr</i>	Condition
LOG_INVALID	Log number of the first invalid log	Information for a specific log is not valid
MQX_INVALID_COMPONENT_BASE	0	Log component data is not valid
MQX_OK	0	Log component data is valid

### Example

```
_mx_uint bad_log;
...
result = _log_test(&bad_log);
switch (result) {
    case MQX_OK:
        printf("Log component is valid.");
        break;
    case MQX_INVALID_COMPONENT_BASE:
        printf("Log component data is not valid.");
        break;
    case LOG_INVALID:
        printf("Log %ld is not valid.", bad_log);
        break;
}
...
```

## 2.1.73 `_log_write`

Writes to the log.

### Prototype

```
source\kernel\log.c
#include <log.h>
_mqx_uint _log_write(
    _mx_uint log_number,
    _mx_uint num_of_parameters,
    _mx_uint param1, ...)
```

### Parameters

*log\_number* [IN] — Log number of a previously created log

*num\_of\_parameters* [IN] — Number of parameters to write

*param1* [IN] — Value to write (number of parameters depends on *num\_of\_parameters*)

### Returns

- MQX\_OK
- Errors

### See Also

[\\_log\\_create](#)

[\\_log\\_read](#)

[\\_log\\_disable](#), [\\_log\\_enable](#)

### Description

The function writes the log entry only if it returns **MQX\_OK**.

Error	Description
LOG_DISABLED	Log is disabled.
LOG_DOES_NOT_EXIST	<i>log_number</i> was not created.
LOG_FULL	Log is full and LOG_OVERWRITE is not set.
LOG_INVALID	<i>log_number</i> is out of range.
MQX_COMPONENT_DOES_NOT_EXIST	Log component is not created.
MQX_INVALID_COMPONENT_HANDLE	Log component data is not valid.

### Example

See `_log_create()`.

## 2.1.74 `_lwevent_clear`

Clears the specified event bits in the lightweight event group.

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _lwevent_clear(
    LWEVENT_STRUCT_PTR event_group_ptr,
    _mqx_uint           bit_mask)
```

### Parameters

*event\_group\_ptr [IN]* — Pointer to the event group  
*bit\_mask [IN]* — Each set bit represents an event bit to clear

### Returns

- MQX\_OK (success)
- LWEVENT\_INVALID\_EVENT (failure: lightweight event group is not valid)

### Traits

Disables and enables interrupts.

### See Also

[\\_lwevent\\_create](#)

[\\_lwevent\\_destroy](#)

[\\_lwevent\\_set, \\_lwevent\\_set\\_auto\\_clear](#)

[\\_lwevent\\_test](#)

[\\_lwevent\\_wait\\_ ...](#)

[\\_lwevent\\_get\\_signalled](#)

[LWEVENT\\_STRUCT](#)

## 2.1.75 `_lwevent_create`

Initializes the lightweight event group.

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _lwevent_create(
    LWEVENT_STRUCT_PTR lwevent_group_ptr,
    _mqx_uint flags)
```

### Parameters

*lwevent\_group\_ptr* [IN] — Pointer to the lightweight event group to initialize

*flags* [IN] — Creation flag; one of the following:

**LWEVENT\_AUTO\_CLEAR** - all bits in the lightweight event group are made autoclearing  
**0** - lightweight event bits are not set as autoclearing by default

*note:* the autoclearing bits can be changed any time later by calling [\\_lwevent\\_set\\_auto\\_clear](#).

### Returns

- MQX\_OK
- MQX\_EINVAL (lwevent is already initialized)
- MQX\_LWEVENT\_INVALID (If, when in user mode, MQX RTOS tries to access a lwevent with inappropriate access rights.)

### Traits

Disables and enables interrupts.

### See Also

[\\_lwevent\\_destroy](#)

[\\_lwevent\\_set](#), [\\_lwevent\\_set\\_auto\\_clear](#)

[\\_lwevent\\_clear](#)

[\\_lwevent\\_test](#)

[\\_lwevent\\_wait\\_ ...](#)

[\\_lwevent\\_get\\_signalled](#)

[LWEVENT\\_STRUCT](#)



## 2.1.76 `_lwevent_destroy`

Deinitializes the lightweight event group.

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _lwevent_destroy(
    LWEVENT_STRUCT_PTR lwevent_group_ptr)
```

### Parameters

*lwevent\_group\_ptr* [IN] — Pointer to the event group to deinitialize

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_LWEVENT_INVALID	Lightweight event group was not valid.

### Traits

Cannot be called from an ISR.

### See Also

[\\_lwevent\\_create](#)

[\\_lwevent\\_set](#), [\\_lwevent\\_set\\_auto\\_clear](#)

[\\_lwevent\\_clear](#)

[\\_lwevent\\_test](#)

[\\_lwevent\\_wait\\_ ...](#)

[\\_lwevent\\_get\\_signalled](#)

[LWEVENT\\_STRUCT](#)

### Description

To reuse the lightweight event group, a task must reinitialize it.

## 2.1.77 `_lwevent_get_signalled`

Gets which particular bit(s) in the lwevent unblocked recent wait command.

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _lwevent_get_signalled(void)
```

### Parameters

None

### Returns

- lwevent mask from last task's lwevent\_wait\_xxx call that unblocked the task

### See Also

[\\_lwevent\\_create](#)

[\\_lwevent\\_destroy](#)

[\\_lwevent\\_set, \\_lwevent\\_set\\_auto\\_clear](#)

[\\_lwevent\\_clear](#)

[\\_lwevent\\_test](#)

[\\_lwevent\\_wait\\_ ...](#)

[LWEVENT\\_STRUCT](#)

### Description

If `_lwevent_wait_xxx(...)` was recently called in a task, following call of `_lwevent_get_signalled` returns the mask of bit(s) that unblocked the command. User can expect valid data only when the recent `_lwevent_wait_xxx(...)` operation did not return `LWEVENT_WAIT_TIMEOUT` or an error value. This is useful primarily for events that are cleared automatically and thus corresponding `LWEVENT_STRUCT` was automatically reset and holds new value.

**Example**

```

result = _lwevent_wait_ticks(&my_event, MY_EVENT_A | MY_EVENT_B, FALSE, 5);
switch (result)
{
    case MQX_OK:
        /* Don't get value using legacy my_event.VALUE, obsolete */
        mask = _lwevent_get_signalled();
        if (mask & MY_EVENT_A)
        {
            printf("MY_EVENT_A unblocked this task.\n");
        }
        if (mask & MY_EVENT_B)
        {
            printf("MY_EVENT_B unblocked this task.\n");
        }
        break;
    case LWEVENT_WAIT_TIMEOUT:
        printf("The task was unblocked after 5 ticks timeout.\n");
        break;
    default:
        printf("An error %d on lwevent.\n", result);
        break;
}

```

## 2.1.78 `_lwevent_set`

Sets the specified event bits in the lightweight event group.

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _lwevent_set(
    LWEVENT_STRUCT_PTR lwevent_group_ptr,
    _mqx_uint flags)
```

### Parameters

*lwevent\_group\_ptr* [IN] — Pointer to the lightweight event group to set bits in  
*flags* [IN] — Each bit represents an event bit to be set

### Returns

- MQX\_OK (success)
- MQX\_LWEVENT\_INVALID (failure: lightweight event group was invalid)

### Traits

Disables and enables interrupts

### See Also

[\\_lwevent\\_create](#)

[\\_lwevent\\_destroy](#)

[\\_lwevent\\_set\\_auto\\_clear](#)

[\\_lwevent\\_clear](#)

[\\_lwevent\\_test](#)

[\\_lwevent\\_wait\\_ ...](#)

[\\_lwevent\\_get\\_signalled](#)

[LWEVENT\\_STRUCT](#)

## 2.1.79 `_lwevent_set_auto_clear`

Sets autoclearing behavior of event bits in the lightweight event group.

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _lwevent_set_auto_clear(
    LWEVENT_STRUCT_PTR  lwevent_group_ptr,
    _mx_uint             auto_mask)
```

### Parameters

*lwevent\_group\_ptr* [IN] — Pointer to the lightweight event group to set bits in  
*auto\_mask* [IN] — Mask of events, which become auto-clear (if corresponding bit of mask is set) or manual-clear (if corresponding bit of mask is clear)

### Returns

- MQX\_OK (success)
- MQX\_LWEVENT\_INVALID (failure: lightweight event group was invalid)

### Traits

Disables and enables interrupts.

### See Also

[\\_lwevent\\_create](#)

[\\_lwevent\\_destroy](#)

[\\_lwevent\\_set](#)

[\\_lwevent\\_clear](#)

[\\_lwevent\\_test](#)

[\\_lwevent\\_wait\\_ ...](#)

[\\_lwevent\\_get\\_signalled](#)

[LWEVENT\\_STRUCT](#)

## 2.1.80 `_lwevent_test`

Tests the lightweight event component.

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _lwevent_test(
    void *lwevent_error_ptr,
    void *td_error_ptr)
```

### Parameters

*lwevent\_error\_ptr* [OUT] — Pointer to the lightweight event group that has an error if MQX RTOS found an error in the lightweight event component (*NULL* if no error is found)

*td\_error\_ptr* [OUT] — TD on the lightweight event in error (*NULL* if no error is found)

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_LWEVENT_INVALID	A lightweight event group was invalid.
Return code from <code>_queue_test()</code>	Waiting queue for a lightweight event group has an error.

### Traits

Cannot be called from an ISR.

### See Also

[\\_lwevent\\_create](#)

[\\_lwevent\\_destroy](#)

[\\_lwevent\\_set, \\_lwevent\\_set\\_auto\\_clear](#)

[\\_lwevent\\_clear](#)

[\\_lwevent\\_wait\\_ ...](#)

[\\_lwevent\\_get\\_signalled](#)

[LWEVENT\\_STRUCT](#)

## 2.1.81 `_lwevent_wait_ ...`

	Wait for the specified lightweight event bits to be set in the lightweight event group:
<code>_lwevent_wait_for()</code>	For the number of ticks (in tick time)
<code>_lwevent_wait_ticks()</code>	For the number of ticks
<code>_lwevent_wait_until()</code>	Until the specified time (in tick time)

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _lwevent_wait_for(
    LWEVENT_STRUCT_PTR    event_ptr,
    _mx_uint               bit_mask,
    bool                   all,
    MQX_TICK_STRUCT_PTR    tick_ptr)

source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _lwevent_wait_ticks(
    LWEVENT_STRUCT_PTR    event_ptr,
    _mx_uint               bit_mask,
    bool                   all,
    _mx_uint               timeout_in_ticks)

source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _lwevent_wait_until(
    LWEVENT_STRUCT_PTR    event_ptr,
    _mx_uint               bit_mask,
    bool                   all,
    MQX_TICK_STRUCT_PTR    tick_ptr)
```

### Parameters

*event\_ptr* [IN] — Pointer to the lightweight event

*bit\_mask* [IN] — Each set bit represents an event bit to wait for

*all* — One of the following:

- TRUE (wait for all bits in *bit\_mask* to be set)
- FALSE (wait for any bit in *bit\_mask* to be set)

*tick\_ptr* [IN] — One of the following:

- pointer to the maximum number of ticks to wait
- NULL (unlimited wait)

*timeout\_in\_ticks* [IN] — One of the following:

- maximum number of ticks to wait

0 (unlimited wait)

### Returns

- MQX\_OK
- LWEVENT\_WAIT\_TIMEOUT (the time elapsed before an event signalled)
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_LWEVENT_INVALID	Lightweight event group is no longer valid or was never valid.

### Traits

Blocks until the event combination is set or until the timeout expires.

Cannot be called from an ISR.

### See Also

[\\_lwevent\\_create](#)

[\\_lwevent\\_destroy](#)

[\\_lwevent\\_set, \\_lwevent\\_set\\_auto\\_clear](#)

[\\_lwevent\\_clear](#)

[\\_lwevent\\_wait\\_ ...](#)

[\\_lwevent\\_get\\_signalled](#)

[LWEVENT\\_STRUCT](#)

[MQX\\_TICK\\_STRUCT](#)



## 2.1.82 `_lwlog_calculate_size`

Calculates the number of single-addressable units required for the lightweight log.

### Prototype

```
source\kernel\lwlog.c  
_mem_size _lwlog_calculate_size(  
    _mqx_uint  entries)
```

### Parameters

*entries* [IN] — Maximum number of entries in the log

### Returns

Number of single-addressable units required

### See Also

[\\_lwlog\\_create](#), [\\_lwlog\\_create\\_at](#)

[\\_lwlog\\_create\\_component](#)

[\\_klog\\_create](#), [\\_klog\\_create\\_at](#)

### Description

The calculation takes into account all headers.

## 2.1.83 `_lwlog_create`, `_lwlog_create_at`

<code>_lwlog_create()</code>	Creates the lightweight log.
<code>_lwlog_create_at()</code>	Creates the lightweight log at the specific location.

### Prototype

```
source\kernel\lwlog.c
_mqx_uint _lwlog_create(
    _mx_uint log_number,
    _mx_uint max_size,
    _mx_uint flags)

source\kernel\lwlog.c
_mqx_uint _lwlog_create_at(
    _mx_uint log_number,
    _mx_uint max_size,
    _mx_uint flags,
    void *where)
```

### Parameters

*log\_number* [IN] — Log number to create ( 1 through 15; 0 is reserved for kernel log)

*max\_size* [IN] — Maximum number of entries in the log

*flags* [IN] — One of the following:

**LOG\_OVERWRITE** (when the log is full, write new entries over oldest ones)

**NULL** (when the log is full, do not write entries; the default behavior)

*where* [IN] — Where to create the lightweight log

### Returns

- **MQX\_OK**
- Errors

Errors from <code>_lwlog_create_component(   )</code>	Description
<b>LOG_EXISTS</b>	Lightweight log with log number <i>log_number</i> exists.
<b>LOG_INVALID</b>	<i>log_number</i> is out of range.
<b>LOG_INVALID_SIZE</b>	<i>max_size</i> is 0.
<b>MQX_INVALID_COMPONENT_BASE</b>	Data for the lightweight log component is not valid.
<b>MQX_INVALID_POINTER</b>	<i>where</i> is <b>NULL</b> .

**Traits**

Creates the lightweight log component if it was not created

**See Also**

[\\_lwlog\\_create\\_component](#)

[\\_klog\\_create](#), [\\_klog\\_create\\_at](#)

[LWLOG\\_ENTRY\\_STRUCT](#)

**Description**

Each entry in the log is the same size and contains a sequence number, a timestamp, and a seven-element array of application-defined data.

## 2.1.84 `_lwlog_create_component`

Creates the lightweight log component.

### Prototype

```
source\kernel\lwlog.c
#include <lwlog.h>
_mqx_uint _lwlog_create_component(void)
```

### Parameters

None

### Returns

- `MQX_OK`
- Errors

Error	Description
<code>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</code>	Function cannot be called from an ISR.
<code>MQX_OUT_OF_MEMORY</code>	MQX RTOS is out of memory.

### Traits

Cannot be called from an ISR

### See Also

[`\_lwlog\_create`](#), [`\_lwlog\_create\_at`](#)

[`\_klog\_create`](#), [`\_klog\_create\_at`](#)

### Description

The lightweight log component provides a maximum of 16 logs, all with the same size of entries. Log number 0 is reserved for kernel log.

An application subsequently creates lightweight logs with `_lwlog_create()` or `_lwlog_create_at()`.

## 2.1.85 `_lwlog_destroy`

Destroys the lightweight log.

### Prototype

```
source\kernel\lwlog.c
#include <lwlog.h>
_mqx_uint _lwlog_destroy(
    _mx_uint log_number)
```

### Parameters

*log\_number* [IN] — Log number of a previously created lightweight log (if *log\_number* is 0, kernel log is destroyed)

### Returns

- MQX\_OK
- Errors

Error	Description
LOG_DOES_NOT_EXIST	<i>log_number</i> was not previously created.
LOG_INVALID	<i>log_number</i> is out of range.
MQX_COMPONENT_DOES_NOT_EXIST	Lightweight log component is not created.
MQX_INVALID_COMPONENT_HANDLE	Lightweight log component data is not valid.

### Traits

Disables and enables interrupts

### See Also

[\\_lwlog\\_create](#), [\\_lwlog\\_create\\_at](#)  
[\\_lwlog\\_create\\_component](#)

## 2.1.86 `_lwlog_disable`, `_lwlog_enable`

<code>_lwlog_disable()</code>	Stops logging to the lightweight log.
<code>_lwlog_enable()</code>	Starts logging to the lightweight log.

### Prototype

```
source\kernel\lwlog.c
#include <lwlog.h>
_mqx_uint _lwlog_disable(
    _mx_uint log_number)

_mqx_uint _lwlog_enable(
    _mx_uint log_number)
```

### Parameters

*log\_number* [IN] — Log number of a previously created lightweight log (if *log\_number* is 0, kernel log is disabled or enabled)

### Returns

- MQX\_OK
- Errors

Error	Description
LOG_DOES_NOT_EXIST	<i>log_number</i> was not created.
LOG_INVALID	<i>log_number</i> is out of range.
MQX_COMPONENT_DOES_NOT_EXIST	Lightweight log component is not created.
MQX_INVALID_COMPONENT_HANDLE	Lightweight log component data is not valid.

### See Also

[\\_lwlog\\_read](#)

[\\_lwlog\\_reset](#)

[\\_lwlog\\_write](#)

## 2.1.87 `_lwlog_read`

Reads the information in the lightweight log.

### Prototype

```
source\kernel\lwlog.c
#include <lwlog.h>
_mqx_uint _lwlog_read(
    _mx_uint          log_number,
    _mx_uint          read_type,

    LWLOG_ENTRY_STRUCT_PTR entry_ptr)
```

### Parameters

*log\_number* [IN] — Log number of a previously created lightweight log (if *log\_number* is 0, kernel log is read)

*read\_type* [IN] — Type of read operation (see `_log_read()`)

*entry\_ptr* [IN] — Where to write the log entry

### Returns

- `MQX_OK`
- Errors

Error	Description
<code>LOG_DOES_NOT_EXIST</code>	<i>log_number</i> was not created.
<code>LOG_ENTRY_NOT_AVAILABLE</code>	Log entry is not available.
<code>LOG_INVALID</code>	<i>log_number</i> is out of range.
<code>LOG_INVALID_READ_TYPE</code>	<i>read_type</i> is not valid.
<code>MQX_COMPONENT_DOES_NOT_EXIST</code>	Lightweight log component is not created.
<code>MQX_INVALID_COMPONENT_HANDLE</code>	Lightweight log component data is not valid.
<code>MQX_INVALID_POINTER</code>	<i>entry_ptr</i> is <code>NULL</code> .

### See Also

[\\_lwlog\\_create](#), [\\_lwlog\\_create\\_at](#)

[\\_lwlog\\_write](#)

[\\_klog\\_display](#)

## 2.1.88 `_lwlog_reset`

Resets the lightweight log to its initial state (remove all entries).

### Prototype

```
source\kernel\lwlog.c
#include <lwlog.h>
_mqx_uint _lwlog_reset(
    _mx_uint log_number)
```

### Parameters

*log\_number* [IN] — Log number of a previously created lightweight log (if *log\_number* is 0, kernel log is reset)

### Returns

- MQX\_OK
- Errors

Error	Description
LOG_DOES_NOT_EXIST	<i>log_number</i> was not created.
LOG_INVALID	<i>log_number</i> is out of range.
MQX_COMPONENT_DOES_NOT_EXIST	Log component is not created.
MQX_INVALID_COMPONENT_HANDLE	Log component data is not valid.

### Traits

Disables and enables interrupts

### See Also

[\\_lwlog\\_disable](#), [\\_lwlog\\_enable](#)



## 2.1.89 \_lwlog\_test

Tests the lightweight log component.

### Prototype

```
source\kernel\lwlog.c
#include <lwlog.h>
_mqx_uint _lwlog_test(
    _mx_uint *log_error_ptr)
```

### Parameters

*log\_error\_ptr* [OUT] — Pointer to the lightweight log in error (*NULL* if no error is found)

### Returns

See description

### Traits

Disables and enables interrupts

### See Also

[\\_lwlog\\_create\\_component](#)

[\\_lwlog\\_create, \\_lwlog\\_create\\_at](#)

### Description

Return value	<i>*log_error_ptr</i>	Condition
LOG_INVALID	Log number of the first invalid lightweight log	Information for a specific lightweight log is not valid
MQX_INVALID_COMPONENT_BASE	0	Lightweight log component data is not valid
MQX_OK	0	Lightweight log component data is valid

## 2.1.90 `_lwlog_write`

Writes to the lightweight log.

### Prototype

```
source\kernel\lwlog.c
#include <lwlog.h>
_mqx_uint _lwlog_write(
    _mx_uint      log_number,
    _mx_max_type  p1,
    _mx_max_type  p2,
    _mx_max_type  p3,
    _mx_max_type  p4,
    _mx_max_type  p5,
    _mx_max_type  p6,
    _mx_max_type  p7)
```

### Parameters

`log_number` [IN] — Log number of a previously created lightweight log

`p1 .. p7` [IN] — Data to be written to the log entry. If `log_number` is 0 and `p1` is  $\geq 10$  (0 through 9 are reserved for MQX RTOS), data specified by `p2` through `p7` is written to kernel log.

### Returns

- `MQX_OK`
- Errors

Error	Description
<code>LOG_DISABLED</code>	Log is disabled.
<code>LOG_DOES_NOT_EXIST</code>	<code>log_number</code> was not created.
<code>LOG_FULL</code>	Log is full and <code>LOG_OVERWRITE</code> is not set.
<code>LOG_INVALID</code>	<code>log_number</code> is out of range.
<code>MQX_COMPONENT_DOES_NOT_EXIST</code>	Log component is not created.
<code>MQX_INVALID_COMPONENT_HANDLE</code>	Log component data is not valid.

### See Also

[\\_lwlog\\_create](#), [\\_lwlog\\_create\\_at](#)

[\\_lwlog\\_read](#)

[\\_lwlog\\_disable](#), [\\_lwlog\\_enable](#)

### Description

The function writes the log entry only if it returns **MQX\_OK**.

## 2.1.91 `_lwmem_alloc ...`

	Allocate this type of lightweight-memory block from the default memory pool
<code>_lwmem_alloc()</code>	Private
<code>_lwmem_alloc_system()</code>	System
<code>_lwmem_alloc_system_zero()</code>	System (zero-filled)
<code>_lwmem_alloc_zero()</code>	Private (zero-filled)
<code>_lwmem_alloc_at()</code>	Private (start address defined)
<code>_lwmem_alloc_align()</code>	Private (aligned)
<code>_lwmem_alloc_system_align()</code>	System (aligned)

### Prototype

```

source\kernel\lwmem.c
void *_lwmem_alloc(
    _mem_size  size)

void *_lwmem_alloc_zero(
    _mem_size  size)

void *_lwmem_alloc_system(
    _mem_size  size)

void *_lwmem_alloc_system_zero(
    _mem_size  size)

void *_lwmem_alloc_at(
    _mem_size  size
    void      *addr)

void *_lwmem_alloc_align(
    _mem_size  requested_size
    void      *req_align)

void *_lwmem_alloc_system_align(
    _mem_size  requested_size
    mem_size  req_align)

```

### Parameter

*size* [IN] — Number of single-addressable units to allocate

*addr* [IN] — Start address of the memory block

*requested\_size* [IN] — Number of single-addressable units to allocate

*req\_align* [IN] — Align requested value

### Returns

- Pointer to the lightweight-memory block (success)
- *NULL* (failure: see task error codes)

#### Task error codes

- `MQX_OUT_OF_MEMORY` — MQX RTOS cannot find a block of the requested size
- `MQX_LWMEM_POOL_INVALID` — Memory pool to allocate from is invalid

#### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

#### See Also

[`\_lwmem\_create\_pool`](#)

[`\_lwmem\_free`](#)

[`\_lwmem\_get\_size`](#)

[`\_lwmem\_set\_default\_pool`](#)

[`\_lwmem\_transfer`](#)

[`\_lwmem\_alloc\_\*\_from`](#)

[`\_msg\_alloc`](#)

[`\_msg\_alloc\_system`](#)

[`\_task\_set\_error`](#)

#### Description

The application must first set a value for the default lightweight-memory pool by calling `_lwmem_set_default_pool()`.

The `_lwmem_alloc` functions allocate at least *size* single-addressable units; the actual number might be greater. The start address of the block is aligned so that tasks can use the returned pointer as a pointer to any data type without causing an error.

Tasks cannot use lightweight-memory blocks as messages. Tasks must use `_msg_alloc()` or `_msg_alloc_system()` to allocate messages.

Only the task that owns a lightweight-memory block that was allocated with one of the following functions can free the block:

- `_lwmem_alloc()`
- `_lwmem_alloc_zero()`
- `_lwmem_alloc_at()`
- `_lwmem_alloc_align()`

Any task can free a lightweight-memory block that is allocated with one of the following functions:

- `_lwmem_alloc_system()`

- `_lwmem_alloc_system_zero()`
- `_lwmem_alloc_system_align()`

## 2.1.92 `_lwmem_alloc_*_from`

	Allocate this type of lightweight-memory block from the specified lightweight-memory pool:
<code>_lwmem_alloc_from()</code>	Private
<code>_lwmem_alloc_system_from()</code>	System
<code>_lwmem_alloc_system_zero_from()</code>	System (zero-filled)
<code>_lwmem_alloc_zero_from()</code>	Private (zero-filled)
<code>_lwmem_alloc_align_from()</code>	Private (aligned)
<code>_lwmem_alloc_system_align_from()</code>	System (aligned)

### Prototype

```

source\kernel\lwmem.c
void *_lwmem_alloc_from(
    _lwmem_pool_id pool_id
    _mem_size      size)

void *_lwmem_alloc_zero_from(
    _lwmem_pool_id pool_id,
    _mem_size      size)

void *_lwmem_alloc_system(
    _mem_size size)

void *_lwmem_alloc_system_zero(
    _mem_size size)

void *_lwmem_alloc_align_from(
    _lwmem_pool_id pool_id,
    _mem_size      requested_size,
    _mem_size      req_align)

void *_lwmem_alloc_system_align_from(
    _lwmem_pool_id pool_id,
    _mem_size      requested_size,
    _mem_size      req_align)

```

### Parameters

*pool\_id* [IN] — Lightweight-memory pool from which to allocate the lightweight-memory block (from `_lwmem_create_pool()`)

*size* [IN] — Number of single-addressable units to allocate

*requested\_size* [IN] — Number of single-addressable units to allocate

*req\_align* [IN] — Align requested value

### Returns

- Pointer to the lightweight-memory block (success)
- *NULL* (failure: see task error codes)

### Task error codes

- MQX\_OUT\_OF\_MEMORY — MQX RTOS cannot find a block of the requested size
- MQX\_LWMEM\_POOL\_INVALID — Memory pool to allocate from is invalid
- MQX\_INVALID\_PARAMETER - Requested alignment is not power of 2

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_lwmem\\_alloc ...](#)

[\\_lwmem\\_create\\_pool](#)

[\\_lwmem\\_free](#)

[\\_lwmem\\_transfer](#)

[\\_msg\\_alloc](#)

[\\_msg\\_alloc\\_system](#)

[\\_task\\_set\\_error](#)

### Description

The functions are similar to `_lwmem_alloc()`, `_lwmem_alloc_system()`, `_lwmem_alloc_system_zero()`, `_lwmem_alloc_zero()`, `_lwmem_alloc_system_align`, and `_lwmem_alloc_align()`, except that the application does not call `_lwmem_set_default_pool()` first.

Only the task that owns a lightweight-memory block that was allocated with one of the following functions can free the block:

- `_lwmem_alloc_from()`
- `_lwmem_alloc_zero_from()`
- `_lwmem_alloc_align()`

Any task can free a lightweight-memory block that is allocated with one of the following functions:

- `_lwmem_alloc_system_from()`
- `_lwmem_alloc_system_zero_from()`
- `_lwmem_alloc_system_align_from()`

## 2.1.93 `_lwmem_create_pool`

Creates the lightweight-memory pool from memory that is outside the default memory pool.

### Prototype

```
source\kernel\lwmem.c
_lwmem_pool_id _lwmem_create_pool(
    LWMEM_POOL_STRUCT_PTR  mem_pool_ptr,
    void                    *start,
    _mem_size               size)
```

### Parameters

*mem\_pool\_ptr* [IN] — Pointer to the definition of the pool  
*start* [IN] — Start of the memory for the pool  
*size* [IN] — Number of single-addressable units in the pool

### Returns

- Pool ID

### See Also

[\\_lwmem\\_alloc\\_\\*\\_from](#)

[\\_lwmem\\_alloc ...](#)

### Description

Tasks use the pool ID to allocate (variable-size) lightweight-memory blocks from the pool.



## 2.1.94 `_lwmem_free`

Free the lightweight-memory block.

### Prototype

```
source\kernel\lwmem.c
_mqx_uint _lwmem_free(
    void *mem_ptr)
```

### Parameters

*mem\_ptr* [IN] — Pointer to the block to free

### Returns

- MQX\_OK (success)
- Errors (failure)

Error/Task Error Codes	Description
MQX_INVALID_POINTER	<i>mem_ptr</i> is <i>NULL</i> .
MQX_LWMEM_POOL_INVALID	Pool that contains the block is not valid.
MQX_NOT_RESOURCE_OWNER	If the block was allocated with <code>_lwmem_alloc()</code> or <code>_lwmem_alloc_zero()</code> , only the task that allocated it can free part of it.

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_lwmem\\_alloc ...](#)

[\\_lwmem\\_free](#)

[\\_task\\_set\\_error](#)

### Description

If the block was allocated with one of the following functions, only the task that owns the block can free it:

- `_lwmem_alloc()`
- `_lwmem_alloc_from()`
- `_lwmem_alloc_zero()`
- `_lwmem_alloc_zero_from()`

Any task can free a block that was allocated with one of the following functions:

- `_lwmem_alloc_system()`
- `_lwmem_alloc_system_from()`
- `_lwmem_alloc_system_zero()`

- `_lwmem_alloc_system_zero_from()`
- `_lwmem_alloc_system_align()`
- `_lwmem_alloc_system_align_from()`

## 2.1.95 `_lwmem_get_size`

Gets the size of the lightweight-memory block.

### Prototype

```
source\kernel\lwmem.c
_mem_size _lwmem_get_size(
    void *mem_ptr)
```

### Parameters

*mem\_ptr* [IN] — Pointer to the lightweight-memory block

### Returns

- Number of single-addressable units in the block (success)
- 0 (failure)

### Task Error Codes

- MQX\_INVALID\_POINTER — *mem\_ptr* is *NULL*.

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_lwmem\\_free](#)

[\\_lwmem\\_alloc ...](#)

[\\_task\\_set\\_error](#)

### Description

The size is the actual size of the block and might be larger than the size that a task requested.

## 2.1.96 `_lwmem_set_default_pool`

Sets the value of the default lightweight-memory pool.

### Prototype

```
source\kernel\lwmem.c  
_lwmem_pool_id _lwmem_set_default_pool(  
    _lwmem_pool_id pool_id)
```

### Parameters

*pool\_id* [IN] — New pool ID

### Returns

Former pool ID

### See Also

[\\_lwmem\\_alloc ...](#)

[\\_lwsem\\_destroy](#)

[\\_lwsem\\_post](#)

[\\_lwsem\\_test](#)

[\\_lwsem\\_wait ...](#)

### Description

Because MQX RTOS allocates lightweight memory blocks from the default lightweight-memory pool when an application calls `_lwmem_alloc()`, `_lwmem_alloc_system()`, `_lwmem_alloc_system_zero()`, or `_lwmem_alloc_zero()`, the application must first call `_lwmem_set_default_pool()`.

## 2.1.97 `_lwmem_test`

Tests all lightweight memory.

### Prototype

```
source\kernel\lwmem.c
_mqx_uint _lwmem_test(
    _lwmem_pool_id *pool_error_ptr,
    void *block_error_ptr)
```

### Parameters

*pool\_error\_ptr* [OUT] — Pointer to the pool in error (points to *NULL* if no error was found)

*block\_error\_ptr* [OUT] — Pointer to the block in error (points to *NULL* if no error was found)

### Returns

- MQX\_OK (no blocks had errors)
- Errors

Error	Description
MQX_CORRUPT_STORAGE_POOL	A memory pool pointer is not correct.
MQX_CORRUPT_STORAGE_POOL_FREE_LIST	Memory pool freelist is corrupted.
MQX_LWMEM_POOL_INVALID	Lightweight-memory pool is corrupted.

### Traits

- Can be called by only one task at a time (see description)
- Disables and enables interrupts

### See Also

[\\_lwmem\\_alloc ...](#) family of functions

### Description

The function checks the checksums in the headers of all lightweight-memory blocks.

The function can be called by only one task at a time because it keeps state-in-progress variables that MQX RTOS controls. This mechanism lets other tasks allocate and free lightweight memory while `_lwmem_test()` runs.

## 2.1.98 `_lwmem_transfer`

Transfers the ownership of the lightweight-memory block from one task to another.

### Prototype

```
source\kernel\lwmem.c
_mqx_uint _lwmem_transfer(
    void      *block_ptr,
    _task_id  source,
    _task_id  target)
```

### Parameters

*block\_ptr* [IN] — Block whose ownership is to be transferred

*source* [IN] — Task ID of the current owner

*target* [IN] — Task ID of the new owner

### Returns

- **MQX\_OK** (success)
- Errors (failure)

Errors/Task Error Codes	Description
MQX_INVALID_POINTER	<i>block_ptr</i> is <i>NULL</i> .
MQX_INVALID_TASK_ID	<i>source</i> or <i>target</i> does not represent a valid task.
MQX_NOT_RESOURCE_OWNER	Block is not a resource of the task represented by <i>source</i>

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_lwmem\\_alloc ...](#) family of functions

[\\_task\\_set\\_error](#)

## 2.1.99 `_lwmsgq_deinit`

Deinitializes a lightweight message queue.

### Synopsis

```
source\kernel\lwmsgq.c
#include <lwmsgq.h>
#include <lwmsgq_prv.h>
_mqx_uint _lwmsgq_deinit(
    void      *location)
```

### Parameters

*location* [IN] — Pointer to memory to create a message queue.

### Returns

- MQX\_OK
- MQX\_EINVAL (The location already points to a valid lightweight message queue.)

### See also

[\\_lwmsgq\\_receive](#)

[\\_lwmsgq\\_send](#)

### Description

This function deinitializes a message queue at location. All the tasks waiting for a message or are blocked upon full message queue are released.

## 2.1.100 `_lwmsgq_init`

Create a lightweight message queue.

### Synopsis

```
source\kernel\lwmsgq.c
#include <lwmsgq.h>
#include <lwmsgq_prv.h>
_mqx_uint _lwmsgq_init(
    void      *location,
    _mx_uint  num_messages,
    _mx_uint  msg_size)
```

### Parameters

- location* [IN] — Pointer to memory to create a message queue.
- num\_message* [IN] — Number of messages in the queue.
- msg\_size* [IN] — Specifies message size as a multiplier factor of *\_mx\_max\_type* items.

### Returns

- MQX\_OK
- See error codes.

### Traits

Disables and enables interrupts.

### See also

[\\_lwmsgq\\_receive](#)

[\\_lwmsgq\\_send](#)

The function creates a message queue at *location*. There must be sufficient memory allocated to hold *num\_messages* of *msg\_size \* sizeof(\_mx\_max\_type)* plus the size of `LWMSGQ_STRUCT`.

### Task error codes

MQX\_EINVAL — The *location* already points to a valid lightweight message queue.



## 2.1.101 `_lwmsgq_receive`

Get a message from a lightweight message queue.

### Synopsis

```
source\kernel\lwmsgq.c
#include <lwmsgq.h>
#include <lwmsgq_prv.h>
_mqx_uint _lwmsgq_receive(
    void                *handle,
    _mqx_max_type_ptr    message,
    _mqx_uint            flags,
    _mqx_uint            ticks,
    MQX_TICK_STRUCT_PTR tick_ptr)
```

### Parameters

*handle* [IN] — Pointer to the message queue created by `_lwmsgq_init`

*message* [OUT] — Received message

*flags* [IN] — LWMSGQ\_RECEIVE\_BLOCK\_ON\_EMPTY (block the reading task if msgq is empty), LWMSGQ\_TIMEOUT\_UNTIL (perform a timeout using the tick structure as the absolute time), LWMSGQ\_TIMEOUT\_FOR (perform a timeout using the tick structure as the relative time)

*ticks* [IN] — The maximum number of ticks to wait. If set to zero waiting time is controlled by the *tick\_ptr* input parameter value. When just LWMSGQ\_RECEIVE\_BLOCK\_ON\_EMPTY flag is set and ticks input parameters is set to zero the receive function waits for an unlimited amount of time.

*tick\_ptr* [IN] — Pointer to the tick structure to use when ticks input parameter is set to zero.

### Returns

- MQX\_OK
- See error codes

### Traits

Disables and enables interrupts

### See also

[\\_lwmsgq\\_deinit](#)

[\\_lwmsgq\\_send](#)

The function removes the first message from the queue and copies the message to the user buffer. The message becomes a resource of the task.

### Task error codes

- LWMSGQ\_INVALID  
The *handle* was not valid.
- LWMSGQ\_EMPTY

The `LWMSGQ_RECEIVE_BLOCK_ON_EMPTY` flag was not used and no messages were in the message queue.

- `LWMSGQ_TIMEOUT`

No messages were in the message queue before the timeout expired.

- `MQX_CANNOT_CALL_FUNCTION_FROM_ISR`

Function cannot be called from an ISR.

## 2.1.102 `_lwmsgq_send`

Put a message on a lightweight message queue.

### Synopsis

```
source\kernel\lwmsgq.c
#include <lwmsgq.h>
#include <lwmsgq_prv.h>
_mqx_uint _lwmsgq_send(
    void                *handle,
    _mqx_max_type_ptr message,
    _mqx_uint            flags)
```

### Parameters

*handle* [IN] — Pointer to the message queue created by `_lwmsgq_init`

*message* [IN] — Pointer to the message to send.

*flags* [IN] — `LWMSGQ_SEND_BLOCK_ON_FULL` — Block the task if queue is full.

`LWMSGQ_SEND_BLOCK_ON_SEND` — Block the task after the message is sent.

### Returns

- `MQX_OK`
- See error codes

### Traits

Disables and enables interrupts

### See also

[\\_lwmsgq\\_deinit](#)

[\\_lwmsgq\\_receive](#)

The function posts a message on the queue. If the queue is full, the task can block and wait or the function returns with `LWMSGQ_FULL`.

### Task error codes

- `LWMSGQ_INVALID`  
The *handle* was not valid.
- `LWMSGQ_FULL`  
The `LWMSGQ_SEND_BLOCK_ON_FULL` flag was NOT USED and message queue was full.
- `MQX_CANNOT_CALL_FUNCTION_FROM_ISR`  
The function cannot be called from ISR when using inappropriate blocking *flags*.

## 2.1.103 `_lwsem_create`

Creates the lightweight semaphore.

### Prototype

```
source\kernel\lwsem.c
_mqx_uint _lwsem_create(
    LWSEM_STRUCT_PTR  lwsem_ptr,
    _mqx_int           initial_count)
```

### Parameters

*lwsem\_ptr* [IN] — Pointer to the lightweight semaphore to create  
*initial\_count* [IN] — Initial semaphore counter

### Returns

- MQX\_OK
- MQX\_EINVAL (lwsem is already initialized)
- MQX\_INVALID\_LWSEM (If, when in user mode, MQX RTOS tries to access a lwsem with inappropriate access rights)

### See Also

[\\_lwsem\\_destroy](#)

[\\_lwsem\\_post](#)

[\\_lwsem\\_test](#)

[\\_lwsem\\_wait ...](#)

### Description

Because lightweight semaphores are a core component, an application need not create the component before it creates lightweight semaphores.

### Example

```
LWSEM_STRUCT  my_lwsem;
void *lwsem_error_ptr;
void *td_error_ptr;
...
_lwsem_create(&my_lwsem, 10);
...
result = _lwsem_wait(&my_lwsem);
if (result != MQX_OK) {
    /* The function failed. */
    result = _lwsem_test(&lwsem_error_ptr, &td_error_ptr);
    if (result != MQX_OK) {
        /* Lightweight semaphore component is valid. */
    }
}
...
```

```
result = _lwsem_post(&my_lwsem);  
...  
_lwsem_destroy(&my_lwsem);  
...
```

## 2.1.104 `_lwsem_destroy`

Destroys the lightweight semaphore.

### Prototype

```
source\kernel\lwsem.c  
_mqx_uint _lwsem_destroy(  
    LWSEM_STRUCT_PTR  lwsem_ptr)
```

### Parameters

*lwsem\_ptr* [IN] — Pointer to the created lightweight semaphore

### Returns

- `MQX_OK` (success)
- `MQX_INVALID_LWSEM` (failure: *lwsem\_ptr* does not point to a valid lightweight semaphore)

### Traits

- Puts all waiting tasks in their ready queues
- Cannot be called from an ISR

### See Also

[\\_lwsem\\_create](#)

### Example

See `_lwsem_create()`.

## 2.1.105 `_lwsem_poll`

Poll for the lightweight semaphore.

### Prototype

```
source\kernel\lwsem.c  
bool _lwsem_poll(  
    LWSEM_STRUCT_PTR  lwsem_ptr)
```

### Parameters

*lwsem\_ptr* [IN] — Pointer to the created lightweight semaphore

### Returns

- TRUE (task got the lightweight semaphore)
- FALSE (lightweight semaphore was not available)

### See Also

[\\_lwsem\\_create](#)

[\\_lwsem\\_wait ...](#) family

### Description

The function is the nonblocking alternative to the `_lwsem_wait` family of functions.

## 2.1.106 `_lwsem_post`

Posts the lightweight semaphore.

### Prototype

```
source\kernel\lwsem.c  
_mqx_uint _lwsem_post(  
    LWSEM_STRUCT_PTR  lwsem_ptr)
```

### Parameters

*lwsem\_ptr* [IN] — Pointer to the created lightweight semaphore

### Returns

- MQX\_OK (success)
- MQX\_INVALID\_LWSEM (failure: *lwsem\_ptr* does not point to a valid lightweight semaphore)

### Traits

Might put a waiting task in the task's ready queue

### See Also

[\\_lwsem\\_create](#)

[\\_lwsem\\_wait ...](#)

### Description

If tasks are waiting for the lightweight semaphore, MQX RTOS removes the first one from the queue and puts it in the task's ready queue.

### Example

See `_lwsem_create()`.



## 2.1.107 \_lwsem\_test

Tests the data structures (including queues) of the lightweight semaphores component.

### Prototype

```
source\kernel\lwsem.c
_mqx_uint _lwsem_test(
    void *lwsem_error_ptr,
    void *td_error_ptr)
```

### Parameters

*lwsem\_error\_ptr* [OUT] — Pointer to the lightweight semaphore in error (*NULL* if no error is found)

*td\_error\_ptr* [OUT] — Pointer to the task descriptor of waiting task that has an error (*NULL* if no error is found)

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_INVALID_LWSEM	Results of _queue_test()

### Traits

- Cannot be called from an ISR
- Disables and enables interrupts

### See Also

[\\_lwsem\\_create](#)

[\\_lwsem\\_destroy](#)

[\\_queue\\_test](#)

### Example

See `_lwsem_create()`.

## 2.1.108 `_lwsem_wait ...`

	Wait (in FIFO order) for the lightweight semaphore:
<code>_lwsem_wait()</code>	Until it is available
<code>_lwsem_wait_for()</code>	For the number of ticks (in tick time)
<code>_lwsem_wait_ticks()</code>	For the number of ticks
<code>_lwsem_wait_until()</code>	Until the specified time (in tick time)

### Prototype

```
source\kernel\lwsem.c
#include <lwsem.h>
_mqx_uint _lwsem_wait(
    LWSEM_STRUCT_PTR sem_ptr)

_mqx_uint _lwsem_wait_for(
    LWSEM_STRUCT_PTR sem_ptr,
    MQX_TICK_STRUCT_PTR tick_time_timeout_ptr)

_mqx_uint _lwsem_wait_ticks(
    LWSEM_STRUCT_PTR sem_ptr,
    _mqx_uint tick_timeout)

_mqx_uint _lwsem_wait_until(
    LWSEM_STRUCT_PTR sem_ptr,
    MQX_TICK_STRUCT_PTR tick_time_ptr)
```

### Parameters

*sem\_ptr* [IN] — Pointer to the lightweight semaphore

*tick\_time\_timeout\_ptr* [IN] — One of the following:

pointer to the maximum number of ticks to wait  
 NULL (unlimited wait)

*tick\_timeout* [IN] — One of the following:

maximum number of ticks to wait  
 0 (unlimited wait)

*tick\_time\_ptr* [IN] — One of the following:

pointer to the time (in tick time) until which to wait  
 NULL (unlimited wait)

### Returns

- `MQX_OK`
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_INVALID_LWSEM	<i>sem_ptr</i> is for a lightweight semaphore that is not longer valid.
MQX_LWSEM_WAIT_TIMEOUT	Timeout expired before the task could get the lightweight semaphore.

**Traits**

- Might block the calling task
- Cannot be called from an ISR

**See Also**[\\_lwsem\\_create](#)[\\_lwsem\\_post](#)[LWSEM\\_STRUCT](#)[MQX\\_TICK\\_STRUCT](#)**TIP**

Because priority inversion might occur if tasks with different priorities access the same lightweight semaphore, we recommend under these circumstances that you use the semaphore component.

**Example**

See `_lwsem_create()`.

## 2.1.109 `_lwtimer_add_timer_to_queue`

Adds the lightweight timer to the periodic queue.

### Prototype

```
source\kernel\lwtimer.c
#include <lwtimer.h>
_mqx_uint _lwtimer_add_timer_to_queue(
    LWTIMER_PERIOD_STRUCT_PTR period_ptr,
    LWTIMER_STRUCT_PTR        timer_ptr,
    _mx_uint                   ticks,
    LWTIMER_ISR_FPTR          function,
    void                       *parameter)
```

### Parameters

*period\_ptr* [IN] — Pointer to the periodic queue

*timer\_ptr* [IN] — Pointer to the lightweight timer to add to the queue

*ticks* [IN] — Offset (in ticks) from the queues' period to expire at, must be smaller than queue period

*function* [IN] — Function to call when the timer expires

*parameter* [IN] — Parameter to pass to function

### Returns

- MQX\_OK (success)
- Errors

Error	Description
MQX_LWTIMER_INVALID	<i>period_ptr</i> points to an invalid periodic queue.
MQX_INVALID_PARAMETER	<i>ticks</i> is greater than or equal to the periodic queue's period.

### Traits

Disables and enables interrupts

### See Also

[\\_lwtimer\\_cancel\\_period](#)

[\\_lwtimer\\_cancel\\_timer](#)

[\\_lwtimer\\_create\\_periodic\\_queue](#)

[LWTIMER\\_PERIOD\\_STRUCT](#)

[LWTIMER\\_STRUCT](#)

## Description

The function inserts the timer in the queue in order of increasing offset from the queue's start time.

## 2.1.110 `_lwtimer_cancel_period`

Cancels all the lightweight timers in the periodic queue.

### Prototype

```
source\kernel\lwtimer.c
#include <lwtimer.h>
_mqx_uint _lwtimer_cancel_period(
    LWTIMER_PERIOD_STRUCT_PTR period_ptr)
```

### Parameters

*period\_ptr* [IN] — Pointer to the periodic queue to cancel

### Returns

- MQX\_OK (success)
- MQX\_LWTIMER\_INVALID (failure; *period\_ptr* points to an invalid periodic queue)

### Traits

Disables and enables interrupts

### See Also

[`\_lwtimer\_add\_timer\_to\_queue`](#)

[`\_lwtimer\_cancel\_timer`](#)

[`\_lwtimer\_create\_periodic\_queue`](#)

[`LWTIMER\_PERIOD\_STRUCT`](#)

### 2.1.111 `_lwtimer_cancel_timer`

Cancels the outstanding timer request.

#### Prototype

```
source\kernel\lwtimer.c
#include <lwtimer.h>
mqx_uint _lwtimer_cancel_timer(
    LWTIMER_STRUCT_PTR timer_ptr)
```

#### Parameters

*timer\_ptr* [IN] — Pointer to the lightweight timer to cancel

#### Returns

- **MQX\_OK** (success)
- **MQX\_LWTIMER\_INVALID** (failure; *timer\_ptr* points to either an invalid timer or to a timer with an periodic queue)

#### Traits

Disables and enables interrupts

#### See Also

[\\_lwtimer\\_add\\_timer\\_to\\_queue](#)

[\\_lwtimer\\_cancel\\_period](#)

[\\_lwtimer\\_create\\_periodic\\_queue](#)

[LWTIMER\\_STRUCT](#)

## 2.1.112 `_lwtimer_create_periodic_queue`

Creates the periodic timer queue.

### Prototype

```
source\kernel\lwtimer.c
#include <lwtimer.h>
_mqx_uint _lwtimer_create_periodic_queue(
    LWTIMER_PERIOD_STRUCT_PTR period_ptr,
    _mx_uint period,
    _mx_uint wait_ticks)
```

### Parameters

*timer\_ptr* [IN] — Pointer to the periodic queue

*period* [IN] — Cycle length (in ticks) of the queue

*wait\_ticks* [IN] — Number of ticks to wait before starting to process the queue

### Returns

**MQX\_OK** (success)

### Traits

Disables and enables interrupts

### See Also

[\\_lwtimer\\_add\\_timer\\_to\\_queue](#)

[\\_lwtimer\\_cancel\\_period](#)

[\\_lwtimer\\_cancel\\_timer](#)

[\\_lwtimer\\_create\\_periodic\\_queue](#)

[LWTIMER\\_PERIOD\\_STRUCT](#)



### 2.1.113 `_lwtimer_test`

Tests all the periodic queues and their lightweight timers for validity and consistency.

#### Prototype

```
source\kernel\lwtimer.c
#include <lwtimer.h>
_mqx_uint _lwtimer_test(
    void *period_error_ptr,
    void *timer_error_ptr)
```

#### Parameters

*period\_error\_ptr* [OUT] — Pointer to the first periodic queue that has an error (*NULL* if no error is found)

*timer\_error\_ptr* [OUT] — Pointer to the first timer that has an error (*NULL* if no error is found)

#### Returns

- MQX\_OK (no periodic queues have been created or no errors found in any periodic queues or timers )
- Errors (an error was found in a periodic queue or a timer)

Error	Description
Error from <code>_queue_test()</code>	A periodic queue or its queue was in error.
MQX_LWTIMER_INVALID	Invalid periodic queue.

#### Traits

Disables and enables interrupts

#### See Also

[\\_lwtimer\\_add\\_timer\\_to\\_queue](#)

[\\_lwtimer\\_cancel\\_period](#)

[\\_lwtimer\\_cancel\\_timer](#)

[\\_lwtimer\\_create\\_periodic\\_queue](#)

2.1.114 `_mem_alloc ...`

	Allocate this type of memory block:	From:
<code>_mem_alloc()</code>	Private	Default memory pool
<code>_mem_alloc_from()</code>	Private	Specified memory pool
<code>_mem_alloc_system()</code>	System	Default memory pool
<code>_mem_alloc_system_from()</code>	System	Specified memory pool
<code>_mem_alloc_system_zero()</code>	System (zero-filled)	Default memory pool
<code>_mem_alloc_system_zero_from()</code>	System (zero-filled)	Specified memory pool
<code>_mem_alloc_zero()</code>	Private (zero-filled)	Default memory pool
<code>_mem_alloc_zero_from()</code>	Private (zero-filled)	Specified memory pool
<code>_mem_alloc_align()</code>	Private (aligned)	Default memory pool
<code>_mem_alloc_align_from()</code>	Private (aligned)	Specified memory pool
<code>_mem_alloc_at()</code>	Private (start address defined)	Default memory pool
<code>_mem_alloc_system_align()</code>	System (aligned)	Default memory pool
<code>_mem_alloc_system_align_from()</code>	System (aligned)	Specified memory pool

## Prototype

```

source\kernel\mem.c
void *_mem_alloc(
    _mem_size  size)

void *_mem_alloc_from(
    _mem_pool_id  pool_id,
    _mem_size      size)

void *_mem_alloc_zero(
    _mem_size  size)

void *_mem_alloc_zero_from(
    _mem_pool_id  pool_id,
    _mem_size      size)

void *_mem_alloc_system(
    _mem_size  size)

void *_mem_alloc_system_from(
    _mem_pool_id  pool_id,
    _mem_size      size)

void *_mem_alloc_system_zero(
    _mem_size  size)

```

```

void *_mem_alloc_system_zero_from(
    _mem_pool_id  pool_id,
    _mem_size     size)

void *_mem_alloc_align(
    _mem_size     size,
    _mem_size     align)

void *_mem_alloc_align_from(
    _mem_pool_id  pool_id,
    _mem_size     size,
    _mem_size     align)

void *_mem_alloc_at(
    _mem_size     size,
    void          *addr)

void *_mem_alloc_system_align(
    _mem_size     size,
    _mem_size     align)

void *_mem_alloc_system_align_from(
    _mem_pool_id  pool_id,
    _mem_size     size,
    _mem_size     align)

```

## Parameters

*size* [IN] — Number of single-addressable units to allocate

*pool\_id* [IN] — Pool from which to allocate the memory block (from **\_mem\_create\_pool()**)

*align* [IN] — Alignment of the memory block

*addr* [IN] — Start address of the memory block

## Returns

- Pointer to the memory block (success)
- NULL (failure: see task error codes)

Task Error Codes	Description
MQX_CORRUPT_STORAGE_POOL_FREE_LIST	Memory pool freelist is corrupted.
MQX_INVALID_CHECKSUM	Checksum of the current memory block header is incorrect.
MQX_OUT_OF_MEMORY	MQX RTOS cannot find a block of the requested size.
MQX_INVALID_CONFIGURATION	User area not aligned on a cache line boundary.
MQX_INVALID_PARAMETER	Requested alignment is not power of 2.

## Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

## See Also

[`\_mem\_create\_pool`](#)

[`\_mem\_free`](#)

[`\_mem\_get\_highwater`](#)

[`\_mem\_get\_highwater\_pool`](#)

[`\_mem\_get\_size`](#)

[`\_mem\_transfer`](#)

[`\_mem\_free\_part`](#)

[`\_msg\_alloc`](#)

[`\_msg\_alloc\_system`](#)

[`\_task\_set\_error`](#)

## Description

The functions allocate at least *size* single-addressable units; the actual number might be greater. The start address of the block is aligned so that tasks can use the returned pointer as a pointer to any data type without causing an error.

Tasks cannot use memory blocks as messages. Tasks must use `_msg_alloc()` or `_msg_alloc_system()` to allocate messages.

Only the task that allocates a memory block with one of the following functions can free the memory block:

- `_mem_alloc()`
- `_mem_alloc_from()`
- `_mem_alloc_zero()`
- `_mem_alloc_zero_from()`
- `_mem_alloc_align()`
- `_mem_alloc_align_from()`
- `_mem_alloc_at()`

Any task can free a memory block that is allocated with one of the following functions:

- `_mem_alloc_system()`
- `_mem_alloc_system_from()`
- `_mem_alloc_system_zero()`
- `_mem_alloc_system_zero_from()`
- `_mem_alloc_system_align()`
- `_mem_alloc_system_align_from()`

## Example

Allocate a memory block for configuration data.

```
config_ptr = _mem_alloc(sizeof(CONFIGURATION_DATA));  
if (config_ptr == NULL) {  
    puts("\nCould not allocate memory.");  
}  
...  
_mem_free(config_ptr);
```

## 2.1.115 `_mem_copy`

Copies the number of single-addressable units.

### Prototype

```
source\psp\cpu_family\mem_copy.c
void _mem_copy(
    void      *src_ptr,
    void      *dest_ptr,
    _mem_size  num_units)
```

### Parameters

*src\_ptr* [IN] — Source address

*dest\_ptr* [IN] — Destination address

*num\_units* [IN] — Number of single-addressable units to copy

### Returns

None

### Traits

Behavior depends on the PSP and the compiler

### See Also

[\\_mem\\_zero](#)

### Description

When possible, MQX RTOS uses an algorithm that is faster than a simple byte-to-byte copy operation. MQX RTOS optimizes the copy operation to avoid alignment problems.

### CAUTION

If the destination address is within the block to copy, MQX RTOS overwrites the overlapping area. Under these circumstances, data is lost.

### Example

```
char src_rqst[100];
char dst_rqst[100];

_mem_copy((void*)&src_rqst, (void*)&dst_rqst, sizeof(100));
```

## 2.1.116 `_mem_create_pool`

Creates the memory pool from memory that is outside the default memory pool.

### Prototype

```
source\kernel\mem.c
_mem_pool_id _mem_create_pool(
    void      *start,
    _mem_size  size)
```

### Parameters

*start* [IN] — Address of the start of the memory pool  
*size* [IN] — Number of single-addressable units in the pool

### Returns

- Pool ID (success)
- NULL (failure: see task error codes)

Task error codes	Description
MQX_MEM_POOL_TOO_SMALL	<i>size</i> is less than the minimum allowable message-pool size
MQX_CORRUPT_MEMORY_SYSTEM	Internal data for the message component is corrupted

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_mem\\_alloc ...](#)

[\\_task\\_set\\_error](#)

### Description

Tasks use the pool ID to allocate (variable-size) memory blocks from the pool.

## 2.1.117 `_mem_extend`

Adds physical memory to the default memory pool.

### Prototype

```
source\kernel\mem.c
_mqx_uint _mem_extend(
    void      *start_of_pool,
    _mem_size size)
```

### Parameters

*start\_of\_pool* [IN] — Pointer to the start of the memory to add  
*size* [IN] — Number of single-addressable units to add

### Returns

- MQX\_OK (success)
- MQX\_INVALID\_SIZE (failure: see description)
- MQX\_INVALID\_COMPONENT\_HANDLE (Memory pool to extend is not valid.)

### See also

[\\_mem\\_get\\_highwater](#)

## MQX\_INITIALIZATION\_STRUCT

### Description

The function adds the specified memory to the default memory pool.

The function fails if *size* is less than (3 \* **MQX\_MIN\_MEMORY\_STORAGE\_SIZE**), as defined in `mem_prv.h`

### Example

Add 16 KB, starting at 0x2000, to the default memory pool.

```
...
_mem_extend((void*)0x2000, 0x4000);
...
```



## 2.1.118 `_mem_extend_pool`

Adds physical memory to the memory pool, which is outside the default memory pool.

### Prototype

```
source\kernel\mem.c
_mqx_uint _mem_extend_pool(
    _mem_pool_id pool_id,
    void *start_of_pool,
    _mem_size size)
```

### Parameters

*pool\_id* [IN] — Pool to which to add memory (from `_mem_create_pool()`)  
*start\_of\_pool* [IN] — Pointer to the start of the memory to add  
*size* [IN] — Number of single-addressable units to add

### Returns

- MQX\_OK (success)
- MQX\_INVALID\_SIZE (failure: see description)
- MQX\_INVALID\_COMPONENT\_HANDLE (Memory pool to extend is not valid.)

### See Also

[\\_mem\\_create\\_pool](#)

[\\_mem\\_get\\_highwater\\_pool](#)

### Description

The function adds the specified memory to the memory pool.

The function fails if *size* is less than  $(3 * \text{MIN\_MEMORY\_STORAGE\_SIZE})$ , as defined in *mem\_prv.h*.

## 2.1.119 `_mem_free`

Frees the memory block.

### Prototype

```
source\kernel\mem.c
_mqx_uint _mem_free(
    void *mem_ptr)
```

### Parameters

*mem\_ptr* [IN] — Pointer to the memory block to free

### Returns

- MQX\_OK (success)
- Errors (failure)

Errors/Task Error Codes	Description
MQX_INVALID_CHECKSUM	Block's checksum is not correct, indicating that at least some of the block was overwritten.
MQX_INVALID_POINTER	<i>mem_ptr</i> is <i>NULL</i> , not in the pool, or misaligned.
MQX_NOT_RESOURCE_OWNER	If the block was allocated with <code>_mem_alloc()</code> or <code>_mem_alloc_zero()</code> , only the task that allocated it can free part of it.

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_mem\\_alloc ...](#)

[\\_mem\\_free\\_part](#)

[\\_task\\_set\\_error](#)

### Description

If the memory block was allocated with one of the following functions, only the task that owns the block can free it:

- `_mem_alloc()`
- `_mem_alloc_from()`
- `_mem_alloc_zero()`
- `_mem_alloc_zero_from()`

Any task can free a memory block that was allocated with one of the following functions:

- `_mem_alloc_system()`

- `_mem_alloc_system_from()`
- `_mem_alloc_system_zero()`
- `_mem_alloc_system_zero_from()`
- `_mem_alloc_system_align`
- `_mem_alloc_system_align_from`

### Example

See `_mem_alloc()`.

## 2.1.120 `_mem_free_part`

Free part of the memory block.

### Prototype

```
source\kernel\mem.c
_mqx_uint _mem_free_part(
    void      *mem_ptr,
    _mem_size requested_size)
```

### Parameters

*mem\_ptr* [IN] — Pointer to the memory block to trim  
*requested\_size* [IN] — Size (in single-addressable units) to make the block

### Returns

- MQX\_OK (success)
- See errors (failure)

### Errors and task error codes

- MQX\_INVALID\_SIZE — One of the following:
  - *requested\_size* is less than 0
  - Size of the original block is less than *requested\_size*

### Task error codes from `_mem_free()`

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_mem\\_free](#)

[\\_mem\\_alloc ...](#)

[\\_mem\\_get\\_size](#)

[\\_task\\_set\\_error](#)

### Description

Under the same restriction as for `_mem_free()`, the function trims from the end of the memory block.

A successful call to the function frees memory only if *requested\_size* is sufficiently smaller than the size of the original block. To determine whether the function freed memory, call `_mem_get_size()` before and after calling `_mem_free_part()`.

### Example

See `_mem_get_size()`.

## 2.1.121 `_mem_get_error`

Gets a pointer to the memory block that is corrupted.

### Prototype

```
source\kernel\mem.c
void *_mem_get_error(void)
```

### Parameters

None

### Returns

Pointer to the memory block that is corrupted

### See Also

[\\_mem\\_test](#)

### Description

If `_mem_test()` indicates an error in the default memory pool, `_mem_get_error()` indicates which block has the error.

In each memory block header, MQX RTOS maintains internal information, including a checksum of the information. As tasks call functions from the `_mem_` family, MQX RTOS recalculates the checksum and compares it with the original. If the checksums do not match, MQX RTOS marks the block as corrupted.

A block is corrupted if:

- A task writes past the end of an allocated memory block and into the header information in the next block. This can occur if:
  - the task allocated a block smaller than it needed
  - a task overflows its stack
  - a pointer is out of range
- A task randomly overwrites memory in the default memory pool

### Example

A low-priority task tests the default memory pool.

```
void Memory_Check_Task(void)
{
    _mqx_uint result;
    while (1)
    {
        result = _mem_test();
        if (result != MQX_OK)
        {
            printf("\nTest of default memory pool failed.");
            printf("\n    error = %x", result);
            printf("\n    block = %x", _mem_get_error());
            printf("\n    Highwater = 0x%lx", _mem_get_highwater());
        }
    }
}
```

```
}  
}  
}
```

## 2.1.122 `_mem_get_error_pool`

Gets the last memory block that caused a memory-pool error in the pool.

### Prototype

```
source\kernel\mem.c  
void *_mem_get_error_pool(  
    mem_pool_id    pool_id)
```

### Parameters

*pool\_id* [IN] — Memory pool from which to get the block

### Returns

Pointer to the memory block

### See Also

[\\_mem\\_test\\_pool](#)

### Description

If `_mem_test_pool()` indicates an error, `_mem_get_error_pool()` indicates which block has the error.

### 2.1.123 **\_mem\_get\_highwater**

Gets the highest memory address that MQX RTOS has allocated in the default memory pool.

#### Prototype

```
source\kernel\mem.c  
void *_mem_get_highwater(void)
```

#### Parameters

None

#### Returns

Highest address allocated in the default memory pool

#### See Also

[\\_mem\\_alloc ...](#)

[\\_mem\\_extend](#)

[\\_mem\\_get\\_highwater\\_pool](#)

#### Description

The function gets the highwater mark; that is, the highest memory address ever allocated by MQX RTOS in the default memory pool. The mark does not decrease if tasks free memory in the default memory pool.

If a task extends the default memory pool ([\\_mem\\_extend\(\)](#)) with an area above the highwater mark and MQX RTOS subsequently allocates memory from the extended memory, the function returns an address from the extended memory.

#### Example

See [\\_mem\\_get\\_error\(\)](#).



## 2.1.124 `_mem_get_highwater_pool`

Gets the highest memory address that MQX RTOS has allocated in the pool.

### Prototype

```
source\kernel\mem.c  
void *_mem_get_highwater_pool(  
    _mem_pool_id pool_id)
```

### Parameters

*pool\_id* [IN] — Pool for which to get the highwater mark (from `_mem_create_pool()`)

### Returns

Highest address allocated in the memory pool

### See Also

[\\_mem\\_alloc ...](#)

[\\_mem\\_create\\_pool](#)

[\\_mem\\_extend\\_pool](#)

[\\_mem\\_get\\_highwater](#)

### Description

The function gets the highwater mark; that is, the highest memory address ever allocated in the memory pool. The mark does not decrease if tasks free blocks in the pool.

If a task extends the memory pool (`_mem_extend_pool()`) with an area above the highwater mark and MQX RTOS subsequently allocates memory from the extended memory, the function returns an address from the extended memory.

### Example

See `_mem_get_error()`.

## 2.1.125 `_mem_get_size`

Gets the size of the memory block.

### Prototype

```
source\kernel\mem.c
_mem_size _mem_get_size(
    void      *mem_ptr)
```

### Parameters

*mem\_ptr* [IN] — Pointer to the memory block

### Returns

- Number of single-addressable units in the block (success)
- 0 (failure)

### Task Error Codes

Error	Description
MQX_CORRUPT_STORAGE_POOL	One of the following: <ul style="list-style-type: none"> <li>• <i>mem_ptr</i> does not point to a block that was allocated with a function from the <code>_mem_alloc</code> family</li> <li>• memory is corrupted</li> </ul>
MQX_INVALID_CHECKSUM	Checksum is not correct because part of the memory block header was overwritten.
MQX_INVALID_POINTER	<i>mem_ptr</i> is NULL or improperly aligned.

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_mem\\_free](#)

[\\_mem\\_alloc ...](#)

[\\_mem\\_free\\_part](#)

[\\_task\\_set\\_error](#)

### Description

The size is the actual size of the memory block and might be larger than the size that a task requested.

### Example

```
original_size = _mem_get_size(ptr);
if (_mem_free_part(ptr, original_size - 40) == MQX_OK) {
```

```
new_size = mem_get_size(ptr);  
if (new_size == original_size) {  
    printf("Block was not large enough to trim.");  
}  
}
```

## 2.1.126 `_mem_set_pool_access`

Sets (lightweight) memory pool access rights for User-mode tasks.

### Prototype

```
source\kernel\lwmem.c
_mqx_uint _mem_set_pool_access(
    _lwmem_pool_id mem_pool_id,
    uint32_t access)
```

### Parameters

*mem\_pool\_id* [IN] — (lightweight) memory pool for access rights to set (returned by [\\_lwmem\\_create\\_pool](#))

*access* [IN] — Access rights to set. Possible values:

- POOL\_USER\_RW\_ACCESS
- POOL\_USER\_RO\_ACCESS
- POOL\_USER\_NO\_ACCESS

### Returns

- MQX\_OK

### Description

This function sets access rights for a (lightweight) memory pool. Setting correct access rights is important for tasks and other code running in the User-mode. User-mode access to a memory pool whose access rights are not set properly causes memory protection exception to be risen.

## 2.1.127 **\_mem\_sum\_ip**

Gets the one's complement checksum over the block of memory.

### Prototype

```
source\psp\cpu_family\ipsum.C
uint32_t mem_sum_ip(
    uint32_t    initial_value,
    _mem_size   length,
    void        *location)
```

### Parameters

*initial\_value [IN]* — Value at which to start the checksum

*length [IN]* — Number of units, each of which is of the type that can hold the maximum data address for the processor

*location [IN]* — Start of the block of memory

### Returns

- Checksum (between 0 and 0xFFFF)
- 0 if and only if all summands are 0

### Description

The checksum is used for packets in Internet protocols. The checksum is the 16-bit one's complement of the one's complement sum of all 16-bit words in the block of memory (as defined in RFC 791).

To get one checksum for multiple blocks, set *initial\_value* to 0, call **\_mem\_sum\_ip()** for the first block, set *initial\_value* to the function's return value, call **\_mem\_sum\_ip()** for the next block, and so on.

## 2.1.128 `_mem_swap_endian`

Converts data to the other endian format.

### Prototype

```
source\kernel\mem.c
void _mem_swap_endian(
    unsigned char* definition,
    void          *data)
```

### Parameters

*definition* [IN] — Pointer to a *NULL*-terminated array, each element of which defines the size (in single-addressable units) of each field in the data structure that defines the data to convert

*data* [IN] — Pointer to the data to convert

### Returns

None

### See Also

[`\_msg\_swap\_endian\_data`](#)

[`\_msg\_swap\_endian\_header`](#)

### Example

```
typedef struct
{
    _task_id    INFO[ARRAY_SIZE];
    _mqx_uint   READ_INDEX;
    _mqx_uint   WRITE_INDEX;
} MY_MSG_DATA;

MY_MSG_DATA msg_data;

unsigned char my_data_def[] =
{sizeof(msg_data.INFO), sizeof(msg_data.READ_INDEX),
  sizeof(msg_data.WRITE_INDEX), 0};
...
_mem_swap_endian((unsigned char *)my_data_def, &msg_data);
...
```

## 2.1.129 `_mem_test`

Tests memory that the memory component uses to allocate memory from the default memory pool.

### Prototype

```
source\kernel\mem.c
_mqx_uint _mem_test(void)
```

### Parameters

None

### Returns

- MQX\_OK (no errors found)
- Errors

Error	Description
MQX_CORRUPT_STORAGE_POOL	A memory pool pointer is not correct.
MQX_CORRUPT_STORAGE_POOL_FREE_LIST	Memory pool freelist is corrupted.
MQX_INVALID_CHECKSUM	Checksum of the current memory block header is incorrect (header is corrupted).

### Traits

- Can be called by only one task at a time (see description)
- Disables and enables interrupts

### See Also

[\\_mem\\_alloc ...](#)

[\\_mem\\_get\\_error](#)

[\\_mem\\_test\\_pool](#)

### Description

The function checks the checksums of all memory-block headers. If the function detects an error, `_mem_get_error()` gets the block in error.

The function can be called by only one task at a time because it keeps state-in-progress variables that MQX RTOS controls. This mechanism lets other tasks allocate and free memory while `_mem_test()` runs.

### Example

See `_mem_get_error()`.

## 2.1.130 `_mem_test_all`

Tests the memory in all memory pools.

### Prototype

```
source\kernel\mem.c  
_mqx_uint _mem_test_all(  
    _mem_pool_id _PTR pool_id)
```

### Parameters

*pool\_id* [OUT] — Pointer to the memory pool in error (initialized only if an error was found):

### Returns

- MQX\_OK (no errors found)
- Errors

Error	Description
Errors from <code>_mem_test()</code>	A memory pool has an error.
Errors from <code>_queue_test()</code>	Memory-pool queue has an error.

### See Also

[\\_mem\\_test](#)

[\\_mem\\_test\\_pool](#)

[\\_queue\\_test](#)



## 2.1.131 `_mem_test_and_set`

Tests and sets a memory location.

### Prototype

```
<MQX_KSDK_DIR>\rtos\mqx\mqx\source\psp\cortex_m\core\M4\dispatch.S
_mqx_uint _mem_test_and_set(
    unsigned char *location_ptr)
```

### Parameters

*location\_ptr* [IN] — Pointer to the single-addressable unit to be set

### Returns

- 0 (location is modified)
- 0x80 (location is not modified)

### Traits

Behavior depends on the PSP

### Description

The function can be used to implement mutual exclusion between tasks.

If the single-addressable unit was 0, the function sets the high bit. If possible, the function uses a bus-cycle indivisible instruction.

### Example

```
char my_mutex;
if (_mem_test_and_set(&my_mutex) == 0){
    /*It was available, now I have it, and I can do some work. */
    ...
}
```

## 2.1.132 `_mem_test_pool`

Tests the memory in the memory pool

### Prototype

```
source\kernel\mem.c  
_mqx_uint _mem_test_pool(  
    _mem_pool_id pool_id)
```

### Parameters

*pool\_id* [IN] — Memory pool to test

### Returns

- MQX\_OK (no errors found)
- See `_mem_test()` (errors found)

### See Also

[\\_mem\\_get\\_error\\_pool](#)

[\\_mem\\_test](#)

[\\_task\\_set\\_error](#)

### Description

If `_mem_test_pool()` indicates an error, `_mem_get_error_pool()` indicates which block has the error.

## 2.1.133 `_mem_transfer`

Transfers the ownership of the memory block from one task to another.

### Prototype

```
source\kernel\mem.c
_mqx_uint _mem_transfer(
    void      *block_ptr,
    _task_id  source,
    _task_id  target)
```

### Parameters

*block\_ptr* [IN] — Memory block whose ownership is to be transferred

*source* [IN] — Task ID of the current owner

*target* [IN] — Task ID of the new owner

### Returns

- MQX\_OK (success)
- Errors (failure)

Error / Task Error Code	Description
MQX_INVALID_CHECKSUM	Block's checksum is not correct, indicating that at least some of the block was overwritten.
MQX_INVALID_POINTER	<i>block_ptr</i> is <i>NULL</i> or misaligned.
MQX_INVALID_TASK_ID	<i>source</i> or <i>target</i> does not represent a valid task.
MQX_NOT_RESOURCE_OWNER	Memory block is not a resource of the task represented by <i>source</i> .

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_mem\\_alloc ...](#)

[\\_mqx\\_get\\_system\\_task\\_id](#)

[\\_task\\_set\\_error](#)

### Example

Transfers memory-block ownership from this task to the system and back.

```
/* Make a memory block a system block so that Task B can use it: */
_mem_transfer(ptr, _task_get_id(), _mqx_get_system_task_id());

/* Task B said it was finished using the block. */
_mem_transfer(ptr, _mqx_get_system_task_id(), _task_get_id());
...
```

## 2.1.134 `_mem_zero`

Fills the region of memory with 0x0.

### Prototype

```
source\psp\cpu_family\mem_zero.c
void _mem_zero(
    void      *ptr,
    _mem_size num_units)
```

### Parameters

*ptr* [IN] — Start address of the memory to be filled

*num\_units* [IN] — Number of single-addressable units to fill

### Returns

None

### See also

[\\_mem\\_copy](#)

### Example

```
char my_array[BUFSIZE];
...
_mem_zero(my_array, sizeof(my_array));
```

## 2.1.135 \_mmu\_add\_vregion

Adds the physical memory region to the MMU page tables. If level 2 translation required and enabled, L2 table is allocated here.

### Prototype

```
source\psp\cpu_family\mmu_xxx.c
#include <psp.h>
_mqx_uint _mmu_add_vregion(
    void      *paddr,
    void      *vaddr,
    _mem_size size,
    _mx_uint  flags)
```

### Parameters

*paddr [IN]* — Physical address of the start of the memory region to be added  
*vaddr [IN]* — Virtual address to correspond to *paddr*  
*size [IN]* — Number of single-addressable units in the memory region  
*flags [IN]* — Flags to be associated with the memory region (PSP related)

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_COMPONENT_DOES_NOT_EXIST	_mmu_vinit() was not previously called.
MQX_INVALID_PARAMETER	Incorrect input parameter.
MQX_OUT_OF_MEMORY	Unable to allocate L2 table.

### See Also

[\\_mmu\\_vinit](#)

### Example

Adds a memory region that includes a flash device. The physical memory region and virtual memory region are the same.

```
uint32_t result;
...
result = _mmu_add_vregion(BSP_FLASH_BASE, BSP_FLASH_BASE, BSP_FLASH_SIZE,
    PSP_PAGE_TABLE_SECTION_SIZE(PSP_PAGE_TABLE_SECTION_SIZE_1MB)
    PSP_PAGE_TYPE(PSP_PAGE_TYPE_CACHE_NON)
    PSP_PAGE_DESCR(PSP_PAGE_DESCR_ACCESS_RW_ALL) );
```

## 2.1.136 `_mmu_vdisable`

Disables (stop) and deinitializes the MMU. Free all level 2 tables if level 2 is supported.

### Prototype

```
source\psp\cpu_family\mmu_xxx.c
#include <psp.h>
_mqx_uint _mmu_vdisable(void)
```

### Parameters

None

### Returns

- `MQX_OK`
- `MQX_COMPONENT_DOES_NOT_EXIST` (`_mmu_vinit()` was not previously called)

### See Also

[`\_mmu\_vinit`](#)

[`\_mmu\_venable`](#)

### Description

The function disables all virtual addresses; applications can access physical addresses only.

## 2.1.137 `_mmu_venable`

Enables (starts) the MMU to provide the virtual memory component.

### Prototype

```
source\psp\cpu_family\mmu_xxx.c
#include <psp.h>
_mqx_uint _mmu_venable(void)
```

### Parameters

None

### Returns

- **MQX\_OK**
- **MQX\_COMPONENT\_DOES\_NOT\_EXIST** (`_mmu_vinit()` was not previously called)

### See Also

[\\_mmu\\_vinit](#)

[\\_mmu\\_vdisable](#)

### Description

The function enables the MMU, allowing an application to access virtual addresses.

## 2.1.138 `_mmu_vinit`

Initializes the MMU to provide the virtual memory component. This function prepares MMU L1 table with default flag.

### Prototype

```
source\psp\cpu_family\vmmu_xxx.c
#include <psp.h>
_mqx_uint _mmu_vinit(
    _mx_uint flags,
    void *base_ptr)
```

### Parameters

*flags* [IN] — Flags that are specific to the CPU type; they might be used, for example, to select the MMU page size (see your PSP release note)

*base\_ptr* [IN] — Base address of the MMU L1 table.

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_COMPONENT_DOES_NOT_EXIST	<code>_cpu_type_initialize_support()</code> was not previously called (see the PSP release note).
MQX_INVALID_PARAMETER	One or both of the following: <ul style="list-style-type: none"> <li>• L1 table points to NULL</li> <li>• Address of L1 table is not aligned</li> <li>• Invalid flags</li> </ul>

### See Also

[`\_mmu\_venable`](#)

[`\_mmu\_vdisable`](#)

### Description

The function initializes the MMU and the MMU page tables, but does not enable the MMU.

### Example

Initialize the MMU on the Vybrid ARM<sup>®</sup> Cortex<sup>®</sup>-A5 processor.

```
_mx_uint result;
...
result = _mmu_vinit(PSP_PAGE_TABLE_SECTION_SIZE(PSP_PAGE_TABLE_SECTION_SIZE_1MB)
PSP_PAGE_DESCR(PSP_PAGE_DESCR_ACCESS_RW_ALL)
PSP_PAGE_TYPE(PSP_PAGE_TYPE_STRONG_ORDER), (void*)L1PageTable);
```



## 2.1.139 \_mmu\_vtop

Gets the physical address that corresponds to the virtual address.

### Prototype

```
source\psp\cpu_family\mmmu_xxx.c
#include <psp.h>
_mqx_uint _mmu_vtop(
void *va,
void *pa)
```

### Parameters

*va* [IN] — Virtual address  
*pa* [OUT] — Physical address

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_INVALID_POINTER	<i>vaddr</i> is invalid.

### See Also

[\\_mmu\\_vinit](#)

### Example

Get the physical address that corresponds to the virtual address of a DMA device.

```
void *addr;
...
if (_mmu_vtop(virtual_addr, &addr) == MQX_OK) {
    _dma_set_start(addr);
}
...
```

## 2.1.140 `_mqx`

Initializes and starts MQX RTOS on the processor.

### Prototype

```
source\kernel\mqx.c
_mqx_uint _mqx(
    MQX_INITIALIZATION_STRUCT_PTR init_struct_ptr)
```

### Parameters

*init\_struct\_ptr* [IN] — Pointer to the MQX RTOS initialization structure for the processor

### Returns

- Does not return (success)
- If application called `_mqx_exit()`, error code that it passed to `_mqx_exit()` (success)
- Errors (failure)

Error	Description
Errors from <code>_int_install_isr()</code>	MQX RTOS cannot install the interrupt subsystem.
Errors from <code>_io_init()</code>	MQX RTOS cannot install the I/O subsystem.
Errors from <code>_mem_alloc_system()</code>	There is not enough memory to allocate either the interrupt stack or the interrupt table.
Errors from <code>_mem_alloc_zero()</code>	There is not enough memory to allocate the ready queues.
MQX_KERNEL_MEMORY_TOO_SMALL	<i>init_struct_ptr</i> does not specify enough kernel memory.
MQX_OUT_OF_MEMORY	There is not enough memory to allocate either the ready queues, the interrupt stack, or the interrupt table.
MQX_TIMER_ISR_INSTALL_FAIL	MQX RTOS cannot install the periodic timer ISR.

### Traits

Must be called exactly once per processor

### See Also

[\\_mqx\\_exit](#)

[\\_int\\_install\\_isr](#)

[\\_mem\\_alloc ...](#)

[MQX\\_INITIALIZATION\\_STRUCT](#)  
[TASK\\_TEMPLATE\\_STRUCT](#)

## Description

The function does the following:

- initializes the default memory pool and memory components
- initializes kernel data
- performs BSP-specific initialization, which includes installing the periodic timer
- performs PSP-specific initialization
- creates the interrupt stack
- creates the ready queues
- starts MQX RTOS tasks
- starts autostart application tasks

## Example

Start MQX RTOS.

```
extern MQX_INITIALIZATION_STRUCT MQX_init_struct;

result = _mqx(&MQX_init_struct);
if (result != MQX_OK) {
    /*An error occurred. */
    ...
}
```

## 2.1.141 `_mqx_bsp_revision`

Pointer to the global string that represents the version of the BSP.

### Prototype

```
<MQX_KSDK_DIR>\rtos\mqx\mqx\source\bsp\init_bsp.c  
const char *_mqx_bsp_revision
```

### See Also

[\\_mqx\\_copyright](#)

[\\_mqx\\_date](#)

[\\_mqx\\_generic\\_revision](#)

[\\_mqx\\_version](#)

[\\_mqx\\_psp\\_revision](#)

### Example

```
puts(_mqx_bsp_revision);
```

## 2.1.142 `_mqx_copyright`

Pointer to the global MQX RTOS copyright string.

### Prototype

```
source\kernel\mqx.c  
const char *_mqx_copyright
```

### See Also

[\\_mqx\\_bsp\\_revision](#)

[\\_mqx\\_date](#)

[\\_mqx\\_generic\\_revision](#)

[\\_mqx\\_version](#)

[\\_mqx\\_psp\\_revision](#)

### Example

```
puts(_mqx_copyright);
```

## 2.1.143 `_mqx_date`

Pointer to the string that indicates the date and time when the MQX RTOS library was built.

### Prototype

```
source\kernel\mqx.c  
const char *_mqx_date
```

### See also

[\\_mqx\\_bsp\\_revision](#)

[\\_mqx\\_copyright](#)

[\\_mqx\\_generic\\_revision](#)

[\\_mqx\\_version](#)

[\\_mqx\\_psp\\_revision](#)

### Example

```
puts(_mqx_date);
```

## 2.1.144 `_mqx_exit`

Terminate the MQX RTOS application and return to the environment that started the application.

### Prototype

```
source\kernel\mqx.c
void _mqx_exit(
    _mqx_uint error_code)
```

### Parameters

*error\_code* [IN] — Error code to return to the function that called `_mqx()`

### Returns

None

### Traits

Behavior depends on the BSP

### See Also

[\\_mqx](#)

### Description

The function returns back to the environment that called `_mqx()`. If the application has installed the MQX RTOS exit handler ([\\_mqx\\_set\\_exit\\_handler](#)), `_mqx_exit()` calls the MQX RTOS exit handler before it exits. By default, `_bsp_exit_handler` is installed as the MQX RTOS exit handler in each BSP.

### NOTE

It is important to ensure that the environment (boot call stack) the MQX RTOS is returning to is in the consistent state. This is not provided by distributed MQX RTOS BSPs, because the boot stack is reused (rewritten) by MQX RTOS Kernel data. Set the boot stack outside of Kernel data section to support correct `_mqx_exit` functionality.

### Example

```
#define FATAL_ERROR 1

if (task_id == NULL) {
    printf("Application error.\n");
    _mqx_exit(FATAL_ERROR);
}
```

## 2.1.145 `_mqx_fatal_error`

Indicates that an error occurred that is so severe that MQX RTOS or the application can no longer function.

### Prototype

```
source\kernel\mqx.c
void _mqx_fatal_error(
    _mqx_uint  error)
```

### Parameters

*error* [IN] — Error code

### Returns

None

### Traits

Terminates the application by calling `_mqx_exit()`

### See Also

[`\_mqx\_exit`](#)

[`\_mqx`](#)

[`\_int\_exception\_isr`](#)

### Description

The function logs an error in kernel log (if it has been created and configured to log errors) and calls `_mqx_exit()`.

MQX RTOS calls `_mqx_fatal_error()` if it detects an unhandled interrupt while it is in `_int_exception_isr()`.

If an application calls `_mqx_fatal_error()` when it detects a serious error, you can use this to help you debug by setting a breakpoint in the function.

### Example

MQX RTOS detects a fatal error.

```
if ((unsigned char*)function_call_frame_ptr >
    (unsigned char*)kernel_data->INTERRUPT_STACK_PTR)
{
    /* MQX walked past the end of the interrupt stack and
    ** therefore the default memory pool is corrupted.
    */
    _mqx_fatal_error(MQX_CORRUPT_INTERRUPT_STACK);
}
```



## 2.1.146 `_mqx_generic_revision`

Pointer to the global string that indicates the revision number of generic MQX RTOS code.

### Prototype

```
source\kernel\mqx.c  
const char *_mqx_generic_revision
```

### See Also

[\\_mqx\\_bsp\\_revision](#)

[\\_mqx\\_copyright](#)

[\\_mqx\\_date](#)

[\\_mqx\\_version](#)

[\\_mqx\\_psp\\_revision](#)

### Example

```
puts(_mqx_generic_revision);
```

## 2.1.147 `_mqx_get_counter`

Gets a unique number.

### Prototype

```
source\kernel\mqx.c  
_mqx_uint _mqx_get_counter(void)
```

### Parameters

None

### Returns

- 16-bit number for 16-bit processors or a 32-bit number for 32-bit processors (unique for the processor and never 0)

## 2.1.148 `_mqx_get_exit_handler`

Gets a pointer to the MQX RTOS exit handler, which MQX RTOS calls when it exits.

### Prototype

```
source\kernel\mqx.c  
MQX_EXIT_FPTR mqx_get_exit_handler(void)
```

### Parameters

None

### Returns

Pointer to the MQX RTOS exit handler

### See Also

[\\_mqx\\_exit](#)

[\\_mqx\\_set\\_exit\\_handler](#)

### Example

See `_mqx_set_exit_handler()`.

## 2.1.149 `_mqx_get_initialization`

Gets a pointer to the MQX RTOS initialization structure.

### Prototype

```
source\kernel\mqx.c  
MQX_INITIALIZATION_STRUCT_PTR _mqx_get_initialization(void)
```

### Parameters

None

### Returns

Pointer to the MQX RTOS initialization structure in kernel data

### See Also

[\\_mqx](#)

[MQX\\_INITIALIZATION\\_STRUCT](#)

## 2.1.150 `_mqx_get_kernel_data`

Gets a pointer to kernel data.

### Prototype

```
source\kernel\mqx.c  
void *_mqx_get_kernel_data(void)
```

### Parameters

None

### Returns

Pointer to kernel data

### See Also

[\\_mqx](#)

[MQX\\_INITIALIZATION\\_STRUCT](#)

### Description

The address of kernel data corresponds to **START\_OF\_KERNEL\_MEMORY** in the MQX RTOS initialization structure that the application used to start MQX RTOS on the processor.

### Example

Check the default I/O channel.

```
kernel_data = _mqx_get_kernel_data();  
if (kernel_data->INIT.IO_CHANNEL) {  
    ...  
}
```

## 2.1.151 `_mqx_get_system_task_id`

Gets the task ID of System Task.

### Prototype

```
source\kernel\mqx.c  
_task_id _mqx_get_system_task_id(void)
```

### Parameters

None

### Returns

Task ID of System Task

### See Also

[\\_mem\\_transfer](#)

### Description

System resources are owned by System Task.

### Example

See [\\_mem\\_transfer\(\)](#).

## 2.1.152 `_mqx_get_tad_data`, `_mqx_set_tad_data`

<code>_mqx_get_tad_data()</code>	Gets the <b>TAD_RESERVED</b> field from the task descriptor.
<code>_mqx_set_tad_data()</code>	Sets the <b>TAD_RESERVED</b> field in the task descriptor.

### Prototype

```
source\kernel\mqx.c
void *_mqx_get_tad_data(
    void      *td)

_mqx_set_tad_data(
    void      *td,
    void      *tad_data)
```

### Parameters

*td* [IN] — Task descriptor  
*tad\_data* [IN] — New value for TAD\_RESERVED

### Returns

- `_mqx_get_tad_data()`: TAD\_RESERVED for *td*

### Description

Third-party compilers can use the functions in their runtime libraries.

## 2.1.153 \_mqx\_idle\_task

Idle Task.

### Prototype

```
source\kernel\idletask.c  
void _mqx_idle_task(  
    uint32_t parameter)
```

### Parameters

*parameter [IN]* — Not used

### Returns

None

### Description

Idle Task is an MQX RTOS task that runs if all application tasks are blocked.

The function implements a simple counter, whose size depends on the CPU.

CPU	Number of bits in the counter
16-bit	64
32-bit	128

You can read the counter from a debugger and calculate idle CPU time.



## 2.1.154 \_mqx\_monitor\_type

The type of monitor used.

### Prototype

```
source\kernel\mqx.c
const _mqx_uint _mqx_monitor_type
```

### Parameters

None

### Returns

None

### Description

Monitor types are defined in: *source\include\mqx.h*.

#### TIP

On some targets, you can use this variable to turn off caches and MMUs if they are present. For details, see your BSP release notes.

### Example

```
#include <mcebx860.h>
...
if ((_mqx_monitor_type == MQX_MONITOR_TYPE_NONE) ||
    (_mqx_monitor_type == MQX_MONITOR_TYPE_BDM))
{
    ...
}
```

## 2.1.155 `_mqx_psp_revision`

Pointer to the global string that indicates the PSP revision number.

### Prototype

```
source\kernel\mqx.c  
const char *_mqx_psp_revision
```

### See Also

[\\_mqx\\_bsp\\_revision](#)

[\\_mqx\\_copyright](#)

[\\_mqx\\_date](#)

[\\_mqx\\_generic\\_revision](#)

[\\_mqx\\_version](#)

### Example

```
puts(_mqx_psp_revision);
```

## 2.1.156 `_mqx_set_context_switch_handler`

Set the address of the MQX RTOS context switch handler, which MQX RTOS calls during the context switching operation.

### Prototype

```
source\kernel\mqx.c
void* _mqx_set_context_switch_handler(
MQX_CONTEXT_SWITCH_FPTR entry)
```

### Parameters

Entry [IN] – Pointer to the context switch handler

### Returns

Pointer to the previous context switch handler

### Description

When the task switching operation occurred, MQX RTOS calls the context switch handler with three parameters passed to the handler:

- The *task\_id* of the task has context restored
- The *task\_id* of the task has context saved
- The *bool* flag indicate whether the stack of the task (which has context saved) has been overflowed or not.

### Example

```
/* The user callback for context switching */
void task_switch_callback(_task_id tid_restored, _task_id tid_saved, bool
stack_error)
{
...
}
...
/* Set the callback for context switching */
_mqx_set_context_switch_handler(task_switch_callback);
```

## 2.1.157 `_mqx_set_exit_handler`

Sets the address of the MQX RTOS exit handler, which MQX RTOS calls when it exits.

### Prototype

```
source\kernel\mqx.c
void _mqx_set_exit_handler(
    MQX_EXIT_FPTR entry)
```

### Parameters

*entry [IN]* — Pointer to the exit handler

### Returns

None

### See Also

[\\_mqx\\_get\\_exit\\_handler](#)

[\\_mqx\\_exit](#)

### Example

Set and get the exit handler.

```
/* Set the BSP exit handler, which is called by _mqx_exit(): */
_mqx_set_exit_handler(_bsp_exit_handler);
...
printf("Exit handler is 0x%lx", (uint32_t)mqx_get_exit_handler());
```

## 2.1.158 `_mqx_version`

A string that indicates the version of MQX RTOS.

### Prototype

```
source\kernel\mqx.c  
const char *_mqx_version
```

### See Also

[\\_mqx\\_bsp\\_revision](#)

[\\_mqx\\_copyright](#)

[\\_mqx\\_date](#)

[\\_mqx\\_generic\\_revision](#)

[\\_mqx\\_psp\\_revision](#)

### Example

```
puts(_mqx_version);
```

## 2.1.159 `_mqx_zero_tick_struct`

A constant zero-initialized tick structure that an application can use to initialize one of its tick structures to zero.

### Prototype

```
source\kernel\mqx.c  
const MQX_TICK_STRUCT  _mqx_zero_tick_struct
```

### See Also

[\\_time\\_add ...](#)

[\\_ticks\\_to\\_time](#)

[\\_time\\_diff, \\_time\\_diff\\_ticks](#)

[\\_time\\_get\\_elapsed, \\_time\\_get\\_elapsed\\_ticks](#)

[\\_time\\_init\\_ticks](#)

[\\_time\\_set, \\_time\\_set\\_ticks](#)

### Description

The constant can be used in conjunction with the **\_time\_add** family of functions to convert units to tick time.

### Example

See `_time_add_day_to_ticks()`.

## 2.1.160 `_msg_alloc`

Allocates a message from the private message pool.

### Prototype

```
source\kernel\msg.c
include <message.h>
void *_msg_alloc(
    _pool_id pool_id)
```

### Parameters

*pool\_id* [IN] — A pool ID from `_msgpool_create()`

### Returns

- Pointer to a message (success)
- NULL (failure)

Task Error Codes	Description
MQX_COMPONENT_DOES_NOT_EXIST	Message component is not created.
MSGPOOL_INVALID_POOL_ID	<i>pool_id</i> is not valid.
MSGPOOL_OUT_OF_MESSAGES	All the messages in the pool are allocated .
Task error codes from <code>_mem_alloc_system()</code>	(If MQX RTOS needs to grow the pool.)

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[`\_msg\_alloc\_system`](#)

[`\_msg\_free`](#)

[`\_msgpool\_create`](#)

[`\_msgpool\_destroy`](#)

[`\_task\_set\_error`](#)

[`\_mem\_alloc ...`](#)

## MESSAGE\_HEADER\_STRUCT

### Description

The size of the message is determined by the message size that a task specified when it called `_msgpool_create()`. The message is a resource of the task until the task either frees it (`_msg_free()`) or puts it on a message queue (`_msgq_send` family of functions.)

## Example

See `_msgpool_create()`.



## 2.1.161 `_msg_alloc_system`

Allocates a message from a system message pool.

### Prototype

```
source\kernel\msg.c
#include <message.h>
void *_msg_alloc_system(
    _msg_size  message_size)
```

### Parameters

*message\_size [IN]* — Maximum size (in single-addressable units) of the message

### Returns

- Pointer to a message of at least *message\_size* single-addressable units (success)
- NULL (failure: message component is not created)

Task Error Codes	Description
MQX_COMPONENT_DOES_NOT_EXIST	Message component is not created.
Task error codes from <code>_mem_alloc_system()</code>	(If MQX RTOS needs to grow the pool.)

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_mem\\_alloc ...](#)  
[\\_msg\\_alloc](#)  
[\\_msg\\_free](#)  
[\\_msgpool\\_create\\_system](#)  
[\\_msgq\\_send](#)  
[\\_task\\_set\\_error](#)

## MESSAGE\_HEADER\_STRUCT

### Description

The size of the message is determined by the message size that a task specified when it called `_msgpool_create_system()`.

The message is a resource of the task until the task either frees it (`_msg_free()`) or puts it on a message queue (`_msgq_send` family of functions.)

### Example

See `_msgq_send()`.

## 2.1.162 `_msg_available`

Gets the number of free messages in the message pool.

### Prototype

```
source\kernel\msg.c
#include <message.h>
_mqx_uint _msg_available(
    _pool_id pool_id)
```

### Parameters

*pool\_id* [IN] — One of the following:

- private message pool for which to get the number of free messages
- MSGPOOL\_NULL\_POOL\_ID** (for system message pools)

### Returns

- Depending on *pool\_id* (success):
- number of free messages in the private message pool
- number of free messages in all system message pools
- 0 (success: no free messages)
- 0 (failure: see description)

### Traits

If *pool\_id* does not represent a valid private message pool, calls **\_task\_set\_error()** to set the task error code to **MSGPOOL\_INVALID\_POOL\_ID**

### See Also

[\\_msgpool\\_create](#)  
[\\_msgpool\\_destroy](#)  
[\\_msg\\_free](#)  
[\\_msg\\_alloc\\_system](#)  
[\\_task\\_set\\_error](#)  
[\\_msg\\_create\\_component](#)

### Description

The function fails if either:

- message component is not created
- *pool\_id* is for a private message pool, but does not represent a valid one

### Example

See `_msgpool_create()`.

## 2.1.163 \_msg\_create\_component

Creates the message component.

### Prototype

```
source\kernel\msg.c
#include <message.h>
_mqx_uint _msg_create_component(void)
```

### Parameters

None

### Returns

- MQX\_OK (success)
- Errors (failure)

Error	Description
MSGPOOL_POOL_NOT_CREATED	MQX RTOS cannot allocate the data structures for message pools.
MSGQ_TOO_MANY_QUEUES	MQX RTOS cannot allocate the data structures for message queues.

### Task Error Codes

- Task error codes from \_mem\_alloc\_system\_zero()
- Task error codes from \_mem\_free()

### Traits

On failure, sets the task error code (see task error codes)

### See Also

[\\_msgq\\_open](#)

[\\_msgpool\\_create](#)

[\\_msgq\\_open\\_system](#)

[\\_msgpool\\_create\\_system](#)

[\\_mem\\_alloc ...](#)

[\\_mem\\_free](#)

### Description

The function uses fields in the MQX RTOS initialization structure to create the number of message pools (**MAX\_MSGPOOLS**) and message queues (**MAX\_MSGQS**). MQX RTOS creates the message component if it is not created when an application calls one of:

- `_msgpool_create()`
- `_msgpool_create_system()`
- `_msgq_open()`
- `_msgq_open_system()`

### 2.1.163.1 Example

See `_msgpool_create()`.

## 2.1.164 `_msg_free`

Free the message.

### Prototype

```
source\kernel\msg.c
#include <message.h>
void _msg_free(
    void *msg_ptr)
```

### Parameters

msg\_ptr [IN] — Pointer to the message to be freed

### Returns

None

### Task Error Codes

- MQX\_INVALID\_POINTER — *msg\_ptr* does not point to a valid message.
- MQX\_NOT\_RESOURCE\_OWNER — Message is already freed.
- MSGQ\_MESSAGE\_IS\_QUEUED — Message is in a queue.

### Traits

If the function does not free the message, it calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[`\_msgpool\_create`](#)

[`\_msgpool\_create\_system`](#)

[`\_msgpool\_destroy`](#)

[`\_msg\_alloc\_system`](#)

[`\_msg\_alloc`](#)

[`\_task\_set\_error`](#)

## MESSAGE\_HEADER\_STRUCT

### Description

Only the task that has the message as its resource can free the message. A message becomes a task's resource when the task allocates the message, and it continues to be a resource until the task either frees it or puts it in a message queue. A message becomes a resource of the task that got it from a message queue.

The function returns the message to the message pool from which it was allocated.

### Example

See `_msgpool_create()`.

## 2.1.165 `_msg_swap_endian_data`

Converts the data portion of the message to the other endian format.

### Prototype

```
source\kernel\msg.c
#include <message.h>
void _msg_swap_endian_data(
    unsigned char      *definition,
    MESSAGE_HEADER_STRUCT_PTR msg_ptr)
```

### Parameters

*definition* [IN] — Pointer to an array (*NULL*-terminated), each element of which defines the size (in single-addressable units) of fields in the data portion of the message

*msg\_ptr* [IN] — Pointer to the message whose data is to be converted

### Returns

None

### Traits

Sets `CONTROL` in the message header to indicate the correct endian format for the processor

### See also

[\\_mem\\_swap\\_endian](#)

[MSG\\_MUST\\_CONVERT\\_DATA\\_ENDIAN](#)

[MESSAGE\\_HEADER\\_STRUCT](#)

### Description

The function calls `_mem_swap_endian()` and uses *definition* to swap single-addressable units:

*message\_ptr + sizeof(MESSAGE\_HEADER\_STRUCT)*

The macro `MSG_MUST_CONVERT_DATA_ENDIAN` determines whether the data portion of the message needs to be converted to the other endian format.

### Example

Compare with the example for `_mem_swap_endian()`.

Determine whether the message comes from a processor with the other endian format and convert the data portion of the message to the other endian format if necessary.

```
typedef struct my_msg_data
{
    _task_id    INFO[ARRAY_SIZE];
    _mqx_uint    READ_INDEX;
    _mqx_uint    WRITE_INDEX;
} MY_MSG_DATA;
```

```

typedef struct my_msg_struct
{
    MSG_HEADER_STRUCT  HEADER;
    MY_MSG_DATA        DATA;
} MY_MSG_STRUCT;

MY_MSG_STRUCT *my_msg_ptr;

_mem_size my_data_def[] =
{
    sizeof(my_msg_ptr->DATA.INFO),
    sizeof(my_msg_ptr->DATA.READ_INDEX),
    sizeof(my_msg_ptr->DATA.WRITE_INDEX),
    0
};

if MSG_MUST_CONVERT_DATA_ENDIAN((my_msg_ptr->HEADER.CONTROL) {
    _msg_swap_endian_data((unsigned char *)my_data_def,
        MESSAGE_HEADER_STRUCT_PTR)my_msg_ptr);
};

```

## 2.1.166 `_msg_swap_endian_header`

Converts the message header to the other endian format.

### Prototype

```
source\kernel\msg.c
#include <message.h>
void _msg_swap_endian_header(
    MESSAGE_HEADER_STRUCT_PTR message_ptr)
```

### Parameters

*message\_ptr* [IN] — Pointer to a message whose header is to be converted

### Returns

None

### Traits

Sets CONTROL in the message header to indicate the correct endian format for the processor

### See Also

[\\_mem\\_swap\\_endian](#)

[\\_msg\\_swap\\_endian\\_data](#)

[MSG\\_MUST\\_CONVERT\\_HDR\\_ENDIAN](#)

[MESSAGE\\_HEADER\\_STRUCT](#)

### Description

The function is not needed for general application code because the IPC component converts the message header. Use it only if you are writing IPC message drivers for a new BSP.

The function calls `_mem_swap_endian()` and uses the field sizes of `MESSAGE_HEADER_STRUCT` to convert the header to the other endian format.

The macro `MSG_MUST_CONVERT_HDR_ENDIAN` determines whether the message header needs to be converted to the other endian format.

### Example

```
MSG_HEADER_STRUCT_PTR msg_ptr;

if (MSG_MUST_CONVERT_HDR_ENDIAN(msg_ptr->CONTROL)) {
    _msg_swap_endian_header(msg_ptr);
}
```



## 2.1.167 \_msgpool\_create

Creates a private message pool.

### Prototype

```
source\kernel\msgpool.c
#include <message.h>
_pool_id _msgpool_create(
    uint16_t message_size,
    uint16_t num_messages,
    uint16_t grow_number,
    uint16_t grow_limit)
```

### Parameters

*message\_size* [IN] — Size (in single-addressable units) of the messages (including the message header) to be created for the message pool

*num\_messages* [IN] — Initial number of messages to be created for the message pool

*grow\_number* [IN] — Number of messages to be added if all the messages are allocated

*grow\_limit* [IN] — If *grow\_number* is not equal to 0; one of the following:

maximum number of messages that the pool can have

0 (unlimited growth)

### Returns

- Pool ID to access the message pool (success)
- 0 (failure)

### Task error codes

Error	Description
MSGPOOL_MESSAGE_SIZE_TOO_SMALL	<i>message_size</i> is less than the size of the message header structure
MQX_OUT_OF_MEMORY	MQX RTOS cannot allocate memory to create the message pool
MSGPOOL_OUT_OF_POOLS	Maximum number of message pools have been created, where the number is defined at initialization time in MAX_MSGPOOLS in the MQX RTOS initialization structure
Task error codes from _mem_alloc_system()	—
Task error codes from _msg_create_component()	—

### Traits

- Creates the message component if it was not previously created
- On failure, calls **\_task\_set\_error()** to set the task error code (see task error codes)

**See Also**[\\_msgpool\\_create\\_system](#)[\\_msgpool\\_destroy](#)[\\_msg\\_alloc](#)[\\_task\\_set\\_error](#)[\\_mem\\_alloc ...](#)[\\_msg\\_create\\_component](#)**MQX\_INITIALIZATION\_STRUCT****Description**

Any task can allocate messages from the pool by calling **\_msg\_alloc()** with the pool ID.

**Example**

Create a private message pool and allocate a message from it.

```
_pool_id          pool;
MESSAGE_HEADER_STRUCT_PTR msg_ptr;

_msg_create_component();
pool = _msgpool_create(100, 10, 10, 50);
...
if (_msg_available(pool)) {
    msg_ptr = _msg_alloc(pool);
    ...
    _msg_free(msg_ptr);
}
...
_msgpool_destroy(pool);
```

## 2.1.168 `_msgpool_create_system`

Creates a system message pool.

### Prototype

```
source\kernel\msgpool.c
#include <message.h>
bool _msgpool_create_system(
    uint16_t message_size,
    uint16_t num_messages,
    uint16_t grow_number,
    uint16_t grow_limit)
```

### Parameters

*message\_size* [IN] — Size (in single-addressable units) of the messages (including the message header) to be created for the message pool

*num\_messages* [IN] — Initial number of messages to be created for the pool

*grow\_number* [IN] — Number of messages to be added if all the messages are allocated

*grow\_limit* [IN] — If *grow\_number* is not 0; one of the following:

- maximum number of messages that the pool can have
- 0 (unlimited growth)

### Returns

- TRUE (success)
- FALSE (failure)

### Traits

- Creates the message component if it was not previously created
- On failure, calls `_task_set_error()` to set the task error code as described for `_msgpool_create()`

### See Also

[\\_msgpool\\_create](#)

[\\_msgpool\\_destroy](#)

[\\_msg\\_alloc\\_system](#)

[\\_task\\_set\\_error](#)

[MQX\\_INITIALIZATION\\_STRUCT](#)

### Description

Tasks can subsequently allocate messages from the pool by calling `_msg_alloc_system()`.

### Example

See `_msgq_send()`.

## 2.1.169 `_msgpool_destroy`

Destroys the private message pool.

### Prototype

```
source\kernel\msgpool.c
#include <message.h>
_mqx_uint _msgpool_destroy(
    _pool_id pool_id)
```

### Parameters

*pool\_id* [IN] — Pool to destroy

### Returns

- MQX\_OK
- Errors

Error	Description
MSGPOOL_ALL_MESSAGES_NOT_FREE	All messages in the message pool have not been freed.
MQX_COMPONENT_DOES_NOT_EXIST	Message component is not created.
MSGPOOL_INVALID_POOL_ID	<i>pool_id</i> does not represent a message pool that was created by <code>_msgpool_create()</code> .

### Traits

Calls `_mem_free()`, which on error sets the task error code

### See Also

[\\_msgpool\\_create](#)

[\\_msg\\_free](#)

[\\_msg\\_alloc](#)

[\\_mem\\_free](#)

### Description

Any task can destroy the private message pool as long as all its messages have been freed.

### Example

See `_msgpool_create()`.

## 2.1.170 \_msgpool\_test

Tests all the message pools.

### Prototype

```
source\kernel\msgpool.c
#include <message.h>
_mqx_uint _msgpool_test(
    void *pool_error_ptr,
    void *msg_error_ptr)
```

### Parameters

*pool\_error\_ptr* [OUT] — (Initialized only if an error is found) If the message in a message pool has an error; one of the following:

pointer to a pool ID if the message is from a private message pool

pointer to a system message pool if the message is from a system message pool

*msg\_error\_ptr* [OUT] — Pointer to the message that has an error (initialized only if an error is found)

### Returns

- MQX\_OK (all messages in all message pools passed)
- Errors

Errors	Description
MQX_COMPONENT_DOES_NOT_EXIST	Message component is not created.
MSGQ_INVALID_MESSAGE	At least one message in at least one message pool failed.

### Traits

Disables and enables interrupts

### See also

[\\_msgpool\\_create](#)

[\\_msgpool\\_create\\_system](#)

### Description

The function checks the validity of each message in each private and system message pool. It reports the first error that it finds.

## 2.1.171 `_msgq_close`

Closes the message queue.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
bool _msgq_close(
    _queue_id queue_id)
```

### Parameters

*queue\_id* [IN] — Queue ID of the message queue to be closed

### Returns

- TRUE (success)
- FALSE (failure)

### Task Error Codes

Error	Description
MQX_COMPONENT_DOES_NOT_EXIST	Message component is not created.
MSGQ_INVALID_QUEUE_ID	<i>queue_id</i> is not valid for this processor.
MSGQ_NOT_QUEUE_OWNER	Task that got <i>queue_id</i> did so by opening a private message queue ( <code>_msgq_open()</code> ) and is not the task calling <code>_msgq_close()</code> .
MSGQ_QUEUE_IS_NOT_OPEN	<i>queue_id</i> does not represent a queue that is open.
Task error codes from <code>_msg_free()</code>	(If MQX RTOS cannot free messages that are in the queue.)

### Traits

- Calls `_msg_free()` to free messages that are in the queue
- On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See also

[\\_msgq\\_open\\_system](#)

[\\_msgq\\_open](#)

[\\_msg\\_free](#)

[\\_msgq\\_send](#)

[\\_task\\_set\\_error](#)

## Description

Only the task that opens a private message queue (**\_msgq\_open()**) can close it. Any task can close an opened system message queue (**\_msgq\_open\_system()**).

- If **\_msgq\_close()** closes the message queue, it frees any messages that are in the queue.
- If **\_msgq\_close()** closes the message queue, a task can no longer use *queue\_id* to access the message queue.
- The message queue can subsequently be opened again with **\_msgq\_open()** or **\_msgq\_open\_system()**.

## 2.1.172 `_msgq_get_count`

Gets the number of messages in the message queue.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
_mqx_uint _msgq_get_count(
    _queue_id queue_id)
```

### Parameters

*queue\_id* [IN] — One of the following:

queue ID of the queue to be checked

**MSGQ\_ANY\_QUEUE** (get the number of messages waiting in all message queues that the task has open)

### Returns

- Number of messages (success)
- 0 (success: queue is empty)
- 0 (failure)

### Task Error Codes

Error	Description
MQX_COMPONENT_DOES_NOT_EXIST	Message component is not created.
MSGQ_INVALID_QUEUE_ID	<i>queue_id</i> is not valid for this processor.
MSGQ_QUEUE_IS_NOT_OPEN	<i>queue_id</i> does not represent a message queue that is open.

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See also

[\\_msgq\\_open](#)

[\\_msgq\\_open\\_system](#)

[\\_msgq\\_receive ...](#)

[\\_msgq\\_poll](#)

[\\_task\\_set\\_error](#)

### Description

The message queue must be previously opened on this processor.



## 2.1.173 `_msgq_get_id`

Converts a message-queue number and processor number to a queue ID.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
_queue_id _msgq_get_id(
    _processor_number processor_number,
    _queue_number queue_number)
```

### Parameters

*processor\_number* [IN] — One of the following:

- processor on which the message queue resides
- 0 (indicates the local processor)

*queue\_number* [IN] — Image-wide unique number that identifies the message queue

### Returns

- Queue ID for the queue (success)
- **MSGQ\_NULL\_QUEUE\_ID** (failure: *\_processor\_number* is not valid)

### See Also

[\\_msgq\\_open\\_system](#)

[\\_msgq\\_open](#)

### Description

The queue ID might not represent an open message queue. The queue ID can be used with functions that access message queues.

### Example

See `_msgq_send()`.

## 2.1.174 \_msgq\_get\_notification\_function

Gets the notification function and its data that are associated with the private or the system message queue.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
_mqx_uint _msgq_get_notification_function(
    _queue_id    queue_id,
    MSGQ_NOTIFICATION_FPTR
                *notification_function_ptr,
    void *notification_data_ptr)
```

### Parameters

*queue\_id* [IN] — Queue ID of the message queue for which to get the notification function

*notification\_function\_ptr* [OUT] — Pointer (which might be *NULL*) to the function that MQX RTOS calls when it puts a message in the message queue

*notification\_data\_ptr* [OUT] — Pointer (which might be *NULL*) to data that MQX RTOS passes to the notification function

### Returns

- MQX\_OK
- Errors

Error	Description
MSGQ_INVALID_QUEUE_ID	<i>queue_id</i> does not represent a valid message queue on this processor.
MQX_COMPONENT_DOES_NOT_EXIST	Message component is not created.
MSGQ_QUEUE_IS_NOT_OPEN	<i>queue_id</i> does not represent an open message queue.

### Traits

On error, does not initialize *notification\_function\_ptr* or *notification\_data\_ptr*

### See Also

[\\_msgq\\_open\\_system](#)

[\\_msgq\\_open](#)

[\\_msgq\\_set\\_notification\\_function](#)

## 2.1.175 `_msgq_get_owner`

Gets the task ID of the task that owns the message queue.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
_task_id _msgq_get_owner(
    _queue_id queue_id)
```

### Parameters

*queue\_id* [IN] — Queue ID of the message queue

### Returns

- Task ID (success)
- MQX\_NULL\_TASK\_ID (failure)

### Task Error Codes

Task Error Codes	Description
MQX_COMPONENT_DOES_NOT_EXIST	Message component is not created.
MQX_INVALID_PROCESSOR_NUMBER	Processor number that <i>queue_id</i> specifies is not valid.
MSGQ_QUEUE_IS_NOT_OPEN	Message queue with queue ID <i>queue_id</i> is not open.

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_msgq\\_open](#)

[\\_msgq\\_open\\_system](#)

[\\_msgq\\_receive ...](#)

[\\_msgq\\_send family](#)

[\\_task\\_set\\_error](#)

## 2.1.176 `_msgq_open`

Opens the private message queue.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
_queue_id _msgq_open(
    _queue_number queue_number,
    uint16_t max_queue_size)
```

### Parameters

*queue\_number* [IN] — One of the following:

queue number of the message queue to be opened on this processor (min. 8, max. as defined in the MQX RTOS initialization structure)

MSGQ\_FREE\_QUEUE (MQX RTOS opens an unopened message queue)

*max\_queue\_size* [IN] — One of the following:

maximum queue size

0 (unlimited size)

### Returns

- Queue ID (success)
- MSGQ\_NULL\_QUEUE\_ID (failure)

### Task Error Codes

Task Error Codes	Description
MSGQ_INVALID_QUEUE_NUMBER	<i>queue_number</i> is out of range
MSGQ_QUEUE_IN_USE	One of the following: <ul style="list-style-type: none"> <li>• message queue is already open</li> <li>• MQX RTOS cannot get a queue number for an unopened queue</li> </ul>

### Task error codes from `_msg_create_component()`

### Traits

- Creates the message component if it was not previously created
- On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_msgq\\_close](#)

[\\_msgq\\_open\\_system](#)

[\\_msg\\_create\\_component](#)

[\\_msgq\\_set\\_notification\\_function](#)  
[\\_task\\_set\\_error](#)

### Description

The open message queue has a *NULL* notification function.

Only the task that opens a private message queue can receive messages from the queue.

A task can subsequently attach a notification function and notification data to the message queue with [\\_msgq\\_set\\_notification\\_function\(\)](#).

### Example

See [\\_msgq\\_send\(\)](#).

## 2.1.177 \_msgq\_open\_system

Opens the system message queue.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
_queue_id _msgq_open_system(
    _queue_number    queue_number,
    uint16_t         max_queue_size,
    MSGQ_NOTIFICATION_FPTR notification_function,
    void              *notification_data)
```

### Parameters

*queue\_number* [IN] — One of the following:

system message queue to be opened (min. 8, max. as defined in the MQX RTOS initialization structure)

MSGQ\_FREE\_QUEUE (MQX RTOS chooses an unopened system queue number)

*max\_queue\_size* [IN] — One of the following:

maximum queue size

0 (unlimited size)

*notification\_function* [IN] — One of the following:

pointer to the function that MQX RTOS calls when it puts a message in the queue

NULL (MQX RTOS does not call a function when it puts a message in the queue)

*notification\_data* [IN] — Data that MQX RTOS passes when it calls *notification\_function*

### Returns

- Queue ID (success)
- 0 (failure)

### Task Error Codes

Task Error Codes	Description
MQX_COMPONENT_DOES_NOT_EXIST	Message component is not created.
MSGQ_MESSAGE_NOT_AVAILABLE	There are no messages in the message queue.
MSGQ_NOT_QUEUE_OWNER	Task is not the owner of the private message queue.
MSGQ_QUEUE_IS_NOT_OPEN	Queue is not open.

### Traits

- Creates the message component if it was not previously created
- On failure, calls **\_task\_set\_error()** to set the task error code as described for **\_msgq\_open()**

### See Also

[\*\*\\_msgq\\_close\*\*](#)

[\*\*\\_msgq\\_open\*\*](#)

[\*\*\\_msgq\\_poll\*\*](#)

[\*\*\\_msgq\\_set\\_notification\\_function\*\*](#)

[\*\*\\_task\\_set\\_error\*\*](#)

### **Description**

Once a system message queue is opened, any task can use the queue ID to receive messages with **\_msgq\_poll()**.

- Tasks cannot receive messages from system message queues with **\_msgq\_receive()**.
- The notification function can get messages from the message queue with **\_msgq\_poll()**.
- A task can change the notification function and its data with **\_msgq\_set\_notification\_function()**.

## 2.1.178 \_msgq\_peek

Gets a pointer to the message that is at the start of the message queue, but do not remove the message.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
void *_msgq_peek(
    _queue_id queue_id)
```

### Parameters

*queue\_id* [IN] — Queue to look at

### Returns

- Pointer to the message that is at the start of the message queue (success)
- NULL (failure)

### Task Error Codes

Task Error Codes	Description
MQX_COMPONENT_DOES_NOT_EXIST	Message component is not created.
MSGQ_INVALID_QUEUE_ID	queue_id is not valid.
MSGQ_MESSAGE_NOT_AVAILABLE	There are no messages in the message queue.
MSGQ_NOT_QUEUE_OWNER	Task is not the owner of the private message queue.
MSGQ_QUEUE_IS_NOT_OPEN	Queue is not open.

### Traits

On failure, calls **\_task\_set\_error()** to set the task error code (see task error codes)

### See Also

[\\_msgq\\_get\\_count](#)

[\\_msgq\\_open\\_system](#)

[\\_msgq\\_receive ...](#)

[\\_msgq\\_send](#)

[\\_task\\_set\\_error](#)

[\\_msg\\_create\\_component](#)

### MESSAGE\_HEADER\_STRUCT

### Description

Call **\_msgq\_get\_count()** first to determine whether there are messages in the queue. If there are no messages, **\_msgq\_peek()** calls **\_task\_set\_error()** with **MSGQ\_MESSAGE\_NOT\_AVAILABLE**.



## 2.1.179 \_msgq\_poll

Polls the message queue for a message, but do not wait if a message is not in the queue. The function is a non-blocking alternative to **\_msgq\_receive()**; therefore, ISRs can use it.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
void *_msgq_poll(
    _queue_id queue_id)
```

### Parameters

*queue\_id* [IN] — Private or system message queue from which to receive a message

### Returns

- Pointer to a message (success)
- NULL (failure)

### Task Error Codes

Task Error Codes	Description
MQX_COMPONENT_DOES_NOT_EXIST	Message component is not created.
MSGQ_INVALID_QUEUE_ID	<i>queue_id</i> is not valid or is not on this processor.
MSGQ_MESSAGE_NOT_AVAILABLE	There are no messages in the message queue.
MSGQ_NOT_QUEUE_OWNER	Queue is a private message queue that the task does not own.
MSGQ_QUEUE_IS_NOT_OPEN	Queue is not open.

### Traits

On failure, calls **\_task\_set\_error()** to set the task error code (see task error codes)

### See Also

[\\_msgq\\_get\\_count](#)

[\\_msgq\\_open\\_system](#)

[\\_msgq\\_receive ...](#)

[\\_msgq\\_send](#)

[\\_task\\_set\\_error](#)

[\\_msg\\_create\\_component](#)

[MESSAGE\\_HEADER\\_STRUCT](#)

## Description

The function is the only way for tasks to receive messages from a system message queue.

- If a system message queue has a notification function, the function can get messages from the queue with **\_msgq\_poll()**.
- If a message is returned, the message becomes a resource of the task.

## Example

```
#define    TEST_QUEUE    16
#define    MAX_SIZE      10
void      *msg_ptr;
_queue_id my_qid;

my_qid = _msgq_open(TEST_QUEUE, MAX_SIZE);

msg_ptr = _msgq_poll(my_qid);
```

## 2.1.180 \_msgq\_receive ...

	Wait for a message from the private message queue:
<code>_msgq_receive()</code>	For the number of milliseconds
<code>_msgq_receive_for()</code>	For the number of ticks (in tick time)
<code>_msgq_receive_ticks()</code>	For the number of ticks
<code>_msgq_receive_until()</code>	Until the specified time (in tick time)

### Prototype

```

source\kernel\msgq.c
#include <message.h>
void *_msgq_receive(
    _queue_id  queue_id,
    uint32_t   ms_timeout)

void *_msgq_receive_for(
    _queue_id      queue_id,
    MQX_TICK_STRUCT_PTR tick_time_timeout_ptr)

void *_msgq_receive_ticks(
    _queue_id  queue_id,
    _mqx_uint  tick_timeout)

void *_msgq_receive_until(
    _queue_id      queue_id,
    MQX_TICK_STRUCT_PTR tick_time_ptr)

```

### Parameters

*queue\_id* [IN] — One of the following:

- private message queue from which to receive a message
- MSGQ\_ANY\_QUEUE (any queue that the task owns)

*ms\_timeout* [IN] — One of the following:

- maximum number of milliseconds to wait. After the timeout elapses without the message, the function returns.
- 0 (unlimited wait)

*tick\_time\_timeout\_ptr* [IN] — One of the following:

- pointer to the maximum number of ticks to wait
- NULL (unlimited wait)

*tick\_timeout* [IN] — One of the following:

- maximum number of ticks to wait
- 0 (unlimited wait)

*tick\_time\_ptr* [IN] — One of the following:

- Pointer to the time (in tick time) until which to wait
- NULL (unlimited wait)

### Returns

- Pointer to a message (success)
- NULL (failure)

### Task Error Codes

Task Error Codes	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_COMPONENT_DOES_NOT_EXIST	Message component is not created.
MSGQ_INVALID_QUEUE_ID	<i>queue_id</i> is for a specific queue, but the ID is not valid.
MSGQ_MESSAGE_NOT_AVAILABLE	No messages were in the message queue before the timeout expired.
MSGQ_NOT_QUEUE_OWNER	Message is not a resource of the task.
MSGQ_QUEUE_IS_NOT_OPEN	One of the following: <ul style="list-style-type: none"> <li>• specific queue is not open</li> <li>• <i>queue_id</i> is MSGQ_ANY_QUEUE, but the task has no queues open</li> </ul>

### Traits

- If no message is available, blocks the task until the message queue gets a message or the timeout expires
- Cannot be called from an ISR
- On failure, calls **\_task\_set\_error()** to set the task error code (see task error codes)

### See Also

[\\_msgq\\_get\\_count](#)

[\\_msgq\\_open](#)

[\\_msgq\\_poll](#)

[\\_msgq\\_send](#)

[\\_task\\_set\\_error](#)

[MESSAGE\\_HEADER\\_STRUCT](#)

## Description

The function removes the first message from the queue and returns a pointer to the message. The message becomes a resource of the task.

The function cannot be used to receive messages from system message queues; this must be done with `_msgq_poll()`.

## Example

See `_msgq_send()`.

## 2.1.181 `_msgq_send`

Sends the message to the message queue.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
bool _msgq_send(
    void *msg_ptr)
```

### Parameters

*msg\_ptr IN* — Pointer to the message to be sent

### Returns

- TRUE (success: see description)
- FALSE (failure)

### Task error codes

Task error code	Meaning	Msg. accepted	Msg. freed
MQX_COMPONENT_DOES_NOT_EXIST	Message component is not created	No	No
MSGQ_INVALID_MESSAGE	<i>msg_ptr</i> is <i>NULL</i> or points to a message that is one of: <ul style="list-style-type: none"> <li>• not valid</li> <li>• on a message queue</li> <li>• free</li> </ul>	No	No
MSGQ_INVALID_QUEUE_ID	Target ID is not a valid queue ID	No	Yes
MSGQ_QUEUE_FULL	Target message queue has reached its maximum size	No	Yes
MSGQ_QUEUE_IS_NOT_OPEN	Target ID does not represent an open message queue	No	Yes
Task error codes from <code>_msg_free()</code>	(If message needs to be freed)	Yes	No

### Traits

- Might dispatch a task
- On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_msg\\_alloc\\_system](#)

[\\_msg\\_alloc](#)

[\\_msgq\\_open](#)

`_msgq_receive ...`

`_msgq_poll`

`_msgq_send_priority`

`_msgq_send_urgent`

`_msg_free`

`_task_set_error`

`MESSAGE_HEADER_STRUCT`

## Description

The function sends a message (priority 0) to a private message queue or a system message queue. The function does not block. The message must be from one of:

- `_msg_alloc()`
- `_msg_alloc_system()`
- `_msgq_poll()`
- `_msgq_receive()`

The message must be overlaid with **MESSAGE\_HEADER\_STRUCT**, with the data portion following the header. In the header, the sending task sets:

- **TARGET\_ID** to a valid queue ID for the local processor or for a remote processor (if **TARGET\_ID** is for a remote processor, the function cannot verify the ID or determine whether the maximum size of the queue is reached)
- **SIZE** to the number of single-addressable units in the message, including the header

If the message is for a message queue on:	MQX RTOS sends the message to:
Local processor	The message queue
Remote processor	The remote processor

If the function returns successfully, the message is no longer a resource of the task.

## Example

```
void TaskB(void)
{
    MESSAGE_HEADER_STRUCT_PTR msg_ptr;
    _queue_id taskb_qid;
    _queue_id main_qid;
    _pool_id pool;

    _msgpool_create_system(sizeof(MESSAGE_HEADER_STRUCT), 4, 0, 0);

    taskb_qid = _msgq_open(TASKB_QUEUE, 0);
    main_qid = _msgq_get_id(0, MAIN_QUEUE);

    msg_ptr = _msg_alloc_system(sizeof(MESSAGE_HEADER_STRUCT));
    while (TRUE) {
        msg_ptr->TARGET_QID = main_qid;
        msg_ptr->SOURCE_QID = taskb_qid;
        if (_msgq_send(msg_ptr) == FALSE){
            /* There was an error sending the message. */
        }
        msg_ptr = _msgq_receive(taskb_qid, 0);
    }
}
```



## 2.1.182 \_msgq\_send\_broadcast

Sends the message to multiple message queues.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
_mqx_uint _msgq_send_broadcast(
    void          *input_msg_ptr,
    _queue_id     *qid_ptr,
    _pool_id      pool_id)
```

### Parameters

*input\_msg\_ptr* [IN] — Pointer to the message to be sent

*qid\_ptr* [IN] — Pointer to an array of queue IDs, terminated by MSGQ\_NULL\_QUEUE\_ID, to which a copy of the message is to be sent

*pool\_id* [IN] — One of the following:

pool ID to allocate messages from

MSGPOOL\_NULL\_POOL\_ID (messages is allocated from a system message pool)

### Returns

- Number that represents the size of the array of queue IDs (success)
- Number less than the size of the array of queue IDs (failure)

### Task Error Codes

Task Error Codes	Description
MQX_INVALID_PARAMETER	<i>qid_ptr</i> does not point to a valid queue ID.
MSGPOOL_OUT_OF_MESSAGES	MQX RTOS could not allocate a message from the message pool.
MSGQ_INVALID_MESSAGE	<i>msg_ptr</i> does not point to a message that was allocated as described for _msgq_send().

- Task error codes from \_msg\_alloc() — (If *pool\_id* represents a private message pool.)
- Task error codes from \_msg\_alloc\_system() — (If *pool\_id* represents a system message pool.)

### Traits

- Calls \_mem\_copy()
- Calls \_mem\_alloc() or \_mem\_alloc\_system() depending on whether *pool\_id* represents a private or system message pool
- Might dispatch one or more tasks
- On failure, calls \_task\_set\_error() to set the task error code (see task error codes)

**See Also**[\\_msgq\\_send](#)[\\_msgq\\_receive ...](#)[\\_msgq\\_poll](#)[\\_msgq\\_send\\_priority](#)[\\_msgq\\_send\\_urgent](#)[\\_task\\_set\\_error](#)[\\_mem\\_alloc ...](#)[\\_mem\\_copy](#)**MESSAGE\_HEADER\_STRUCT****Description**

For conditions on the message, see [\\_msgq\\_send\(\)](#).

The function sends a priority 0 message.

For each copy of the message, the function sets the target queue ID in the message header with a queue ID from the array of queue IDs.

The function does not block.

If the function returns successfully, the message is no longer a resource of the task.

It is the responsibility of the application to handle the consequences of messages being lost.

**Example**

```

MESSAGE_HEADER_STRUCT_PTR  msg_ptr;
_queue_id                  bcast_list = {taskb_qid,
                                         main_qid,
                                         MSGQ_NULL_QUEUE_ID};

_pool_id                   pool;
...
pool = _msgpool_create(sizeof(MESSAGE_HEADER_STRUCT), 4, 0, 0);
...
msg_ptr->SOURCE_QID = taskb_qid;
if (_msgq_send_broadcast(msg_ptr, bcast_list, pool) == 2) {
    /* All the messages were sent. */
}

```

## 2.1.183 `_msgq_send_priority`

Sends the priority message to the message queue.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
bool _msgq_send_priority(
    void      *input_msg_ptr,
    _mqx_uint priority)
```

### Parameters

*input\_msg\_ptr* [IN] — Pointer to the message to be sent

*priority* [IN] — Priority of the message, between:

0 (lowest)

MSG\_MAX\_PRIORITY (highest; 15)

### Returns

- TRUE (success)
- FALSE (failure)

### Task error codes

As described for `_msgq_send()`

### MSGQ\_INVALID\_MESSAGE\_PRIORITY

Priority is greater than **MSG\_MAX\_PRIORITY** (message is not accepted and is not freed).

### Traits

- Might dispatch a task
- On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[`\_msgq\_send`](#)

[`\_msg\_alloc\_system`](#)

[`\_msg\_alloc`](#)

[`\_msgq\_send\_broadcast`](#)

[`\_msgq\_send\_urgent`](#)

[`\_msgq\_receive ...`](#)

[`\_msgq\_poll`](#)

[`\_task\_set\_error`](#)

### MESSAGE\_HEADER\_STRUCT

## Description

The function inserts the message in a message queue based on the priority of the message; it inserts higher-priority messages ahead of lower-priority ones. Messages with the same priority are inserted in FIFO order.

If the function returns successfully, the message is no longer a resource of the task.

Messages sent with `_msgq_send()` and `_msgq_send_broadcast()` are priority 0 messages.

## Example

Task B sends a priority-one message and an urgent message to main queue. If the task that owns main queue is not waiting for a message or is of equal or lower priority than Task B, it receives the urgent message before the priority-one message.

```
void TaskB(void)
{
    MESSAGE_HEADER_STRUCT_PTR  priority_msg_ptr;
    MESSAGE_HEADER_STRUCT_PTR  urgent_msg_ptr;
    _queue_id                   taskb_qid;
    _queue_id                   main_qid;

    taskb_qid = _msgq_open(TASKB_QUEUE, 0);
    main_qid = _msgq_get_id(0, MAIN_QUEUE);
    ...
    while (TRUE) {
        priority_msg_ptr->TARGET_QID = urgent_msg_ptr->TARGET_QID =
            main_qid;
        priority_msg_ptr->SOURCE_QID = urgent_msg_ptr->SOURCE_QID =
            taskb_qid;
        if (_msgq_send_priority(priority_msg_ptr, 1)){
            _msgq_send_urgent(urgent_msg_ptr);
        }
        ...
    }
}
```

## 2.1.184 `_msgq_send_queue`

Sends the message directly to the private or system message queue.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
bool _msgq_send_queue(
    void      *msg_ptr,
    _queue_id qid)
```

### Parameters

*msg\_ptr* [IN] — Pointer to the message to be sent  
*qid* [IN] — Message queue into which to put the message

### Returns

- TRUE (success)
- FALSE (failure)

### Traits

- Might dispatch a task
- On failure, calls `_task_set_error()` to set the task error code as described for `_msgq_send()`

### See Also

[\\_msgq\\_send](#)  
[\\_msgq\\_send\\_broadcast](#)  
[\\_msgq\\_send\\_urgent](#)  
[\\_msgq\\_send\\_priority](#)  
[\\_msg\\_alloc\\_system](#)  
[\\_msg\\_alloc](#)  
[\\_msgq\\_open](#)  
[\\_msgq\\_receive ...](#)  
[\\_msgq\\_poll](#)  
[\\_task\\_set\\_error](#)

## MESSAGE\_HEADER\_STRUCT

### Description

The function sends the message as described for [\\_msgq\\_send](#) to the queue specified by parameter *qid* despite the target queue ID in the message header.

Target queue ID of the message must be always filled up before sending.

If the function returns successfully, the message is no longer a resource of the task.

### Example

IPC router sends messages with different TARGET\_QID into the routing queue.

```
_mqx_uint _ipc_msg_route_internal
{
    ...
    route_ptr = (IPC_MSG_ROUTING_STRUCT_PTR)_ipc_msg_processor_route_exists(pnum);
    if (!route_ptr) {
        _task_set_error(MSGQ_INVALID_QUEUE_ID);
        return(FALSE);
    }
    queue = route_ptr->QUEUE;
    result = _msgq_send_queue(message, BUILD_QID(kernel_data->INIT.PROCESSOR_NUMBER,
queue));
    ...
}
```

## 2.1.185 `_msgq_send_urgent`

Sends the urgent message to the message queue.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
bool _msgq_send_urgent(
    void *msg_ptr)
```

### Parameters

*msg\_ptr* [IN] — Pointer to the message to be sent

### Returns

- TRUE (success)
- FALSE (failure)

### Traits

- Might dispatch a task
- On failure, calls `_task_set_error()` to set the task error code as described for `_msgq_send()`

### See Also

[\\_msgq\\_send](#)

[\\_msgq\\_send\\_priority](#)

[\\_msgq\\_send\\_queue](#)

[\\_msg\\_alloc\\_system](#)

[\\_msg\\_alloc](#)

[\\_msgq\\_receive ...](#)

[\\_msgq\\_poll](#)

[\\_task\\_set\\_error](#)

## MESSAGE\_HEADER\_STRUCT

### Description

The function sends the message as described for `_msgq_send()`.

The function puts the message at the start of the message queue, ahead of any other urgent messages.

If the function returns successfully, the message is no longer a resource of the task.

### Example

See `_msgq_send_priority()`.

## 2.1.186 \_msgq\_set\_notification\_function

Sets the notification function for the private or the system message queue.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
MSGQ_NOTIFICATION_FPTR _msgq_set_notification_function(
    _queue_id      qid,
    MSGQ_NOTIFICATION_FPTR notification_function,
    void           *notification_data)
```

### Parameters

*qid* [IN] — Private or system message queue for which to install the notification function

*notification\_function* [IN] — Function that MQX RTOS calls when MQX RTOS puts a message in the queue

*notification\_data* [IN] — Data that MQX RTOS passes when it calls *notification\_function*

### Returns

See description

Return value	Meaning	Notification function installed?
Pointer to the previous notification function	Success	Yes
NULL	Success: Previous notification function was <i>NULL</i>	Yes
NULL	Failure	No

### Task Error Codes

Task Error Codes	Description
MQX_OK	Notification function is installed; the previous function was <i>NULL</i> .
MSGQ_INVALID_QUEUE_ID	<i>qid</i> is not valid.
MSGQ_QUEUE_IS_NOT_OPEN	Queue is not open.
MQX_COMPONENT_DOES_NOT_EXIST	Message component is not created.

### Traits

On failure, calls **\_task\_set\_error()** to set the task error code (see description and task error codes)



## See Also

[\\_msgq\\_open\\_system](#)

[\\_msgq\\_open](#)

[\\_msgq\\_poll](#)

[\\_msgq\\_get\\_notification\\_function](#)

[\\_task\\_set\\_error](#)

## Description

If the message queue is a system message queue, the function replaces the notification function and data that were installed with **\_msgq\_open\_system()**.

The notification function for a system message queue can get messages from the queue with **\_msgq\_poll()**.

The notification function for a private message queue cannot get messages from the queue.

## 2.1.187 \_msgq\_test

Tests all messages in all open message queues.

### Prototype

```
source\kernel\msgq.c
#include <message.h>
_mqx_uint _msgq_test(
    void *queue_error_ptr,
    void *msg_error_ptr)
```

### Parameters

*queue\_error\_ptr* [OUT] — Pointer to the message queue that has a message with an error (initialized only if an error is found)

*msg\_error\_ptr* [OUT] — Pointer to the message that has an error (initialized only if an error is found)

### Returns

- MQX\_OK (success: no errors are found)
- MSGQ\_INVALID\_MESSAGE (success: an error is found)
- MQX\_COMPONENT\_DOES\_NOT\_EXIST (Failure: Message component is not created.)

### Traits

Disables and enables interrupts

### See Also

[\\_msgq\\_open](#)

[\\_msgq\\_open\\_system](#)

### Description

The function checks the consistency and validity of all messages in all private and system message queues that are open.

### Example

A low-priority task tests message queues. If the task finds an invalid message, it exits MQX RTOS.

```
MESSAGE_HEADER_STRUCT_PTR msg_ptr;
_mqx_uint queue_number;
...
if (_msgq_test(&queue_number, &msg_ptr) != MQX_OK) {
    printf("Message queue %ld, msg_ptr 0x%lx is not valid.",
        queue_number, msg_ptr);
    _mqx_exit();
}
...
```

## 2.1.188 `_mutatr_destroy`

Deinitializes the mutex attributes structure.

### Prototype

```
source\kernel\mutex.c
#include <mutex.h>
_mqx_uint _mutatr_destroy(
    MUTEX_ATTR_STRUCT_PTR attr_ptr)
```

### Parameters

*attr\_ptr [IN]* — Pointer to the mutex attributes structure; initialized with `_mutatr_init()`

### Returns

- `MQX_EOK` (success)
- `MQX_EINVAL` (failure: *attr\_ptr* is `NULL` or points to an invalid attributes structure)

### See Also

[`\_mutatr\_init`](#)

[`MUTEX\_ATTR\_STRUCT`](#)

### Description

To reuse the mutex attributes structure, a task must reinitialize the structure.

### Example

See `_mutatr_get_priority_ceiling()`.

## 2.1.189 `_mutatr_get_priority_ceiling`, `_mutatr_set_priority_ceiling`

<code>_mutatr_get_priority_ceiling()</code>	Gets the priority value of the mutex attributes structure.
<code>_mutatr_set_priority_ceiling()</code>	Sets the priority value of the mutex attributes structure.

### Prototype

```
source\kernel\mutex.c
#include <mutex.h>
_mqx_uint _mutatr_get_priority_ceiling(
    MUTEX_ATTR_STRUCT_PTR attr_ptr,
    _mqx_uint               *priority_ptr)

_mqx_uint _mutatr_set_priority_ceiling(
    MUTEX_ATTR_STRUCT_PTR attr_ptr,
    _mqx_uint             priority)
```

### Parameters

*attr\_ptr* [IN] — Pointer to an initialized mutex attributes structure  
*priority\_ptr* [OUT] — Pointer to the current priority  
*priority* [IN] — New priority

### Returns

- MQX\_EOK (success)
- MQX\_EINVAL (failure: *attr\_ptr* is NULL or points to an invalid attributes structure)

### See Also

[\\_mutatr\\_init](#)

## MUTEX\_ATTR\_STRUCT

### Description

Priority applies only to mutexes whose scheduling protocol is priority protect.

### Example

```
MUTEX_ATTR_STRUCT mutex_attributes;
_mqx_uint priority;
...
if (_mutatr_init(&mutex_attributes) != MQX_EOK) {
    result = _mutatr_set_sched_protocol(&mutex_attributes,
        MUTEX_PRIO_PROTECT | MUTEX_PRIO_INHERIT);
    result = _mutatr_set_priority_ceiling(&mutex_attributes, 6);
    ...
    result = _mutatr_get_priority_ceiling(&mutex_attributes,
        &priority);
```

```
if (result == MQX_EOK) {
    printf("\nPriority ceiling is %ld", priority);
    result = _mutex_init(&mutex, &mutex_attributes);
    result = _mutatr_destroy(&mutex_attributes);
    if (result != MQX_EOK) {
        /* Could not initialize the mutex. */
    }
}
```

## 2.1.190 `_mutatr_get_sched_protocol`, `_mutatr_set_sched_protocol`

<code>_mutatr_get_sched_protocol()</code>	Gets the scheduling protocol of the mutex attributes structure.
<code>_mutatr_set_sched_protocol()</code>	Sets the scheduling protocol of the mutex attributes structure.

### Prototype

```
source\kernel\mutex.c
#include <mutex.h>
_mqx_uint _mutatr_get_sched_protocol(
    MUTEX_ATTR_STRUCT_PTR attr_ptr,
    _mqx_uint               *protocol_ptr)

_mqx_uint _mutatr_set_sched_protocol(
    MUTEX_ATTR_STRUCT_PTR attr_ptr,
    _mqx_uint             protocol)
```

### Parameters

*attr\_ptr* [IN] — Pointer to an initialized mutex attributes structure  
*protocol\_ptr* [OUT] — Pointer to the current scheduling protocol  
*protocol* [IN] — New scheduling protocol (see scheduling protocols)

### Returns

- MQX\_EOK (success)
- MQX\_EINVAL (failure: *attr\_ptr* is NULL or points to an invalid attributes structure)

### See Also

[\\_mutatr\\_init](#)

[\\_mutatr\\_get\\_priority\\_ceiling](#), [\\_mutatr\\_set\\_priority\\_ceiling](#)

[MUTEX\\_ATTR\\_STRUCT](#)

### Scheduling Protocols

Protocol	Description
MUTEX_PRIO_INHERIT	(Priority inheritance) If the task that locks the mutex has a lower priority than any task that is waiting for the mutex, MQX RTOS temporarily raises the task priority to the level of the highest-priority waiting task while the task locks the mutex.

Protocol	Description
MUTEX_PRIO_PROTECT	(Priority protect) If the task that locks the mutex has a lower priority than the mutex, MQX RTOS temporarily raises the task priority to the level of the mutex while the task locks the mutex. If this is set, priority inheritance must be set.
MUTEX_NO_PRIO_INHERIT	(Priority none) Priority of the mutex or of tasks waiting for the mutex does not affect the priority of the task that locks the mutex.

**Example**

See `_mutatr_get_priority_ceiling()`.

## 2.1.191 `_mutatr_get_spin_limit`, `_mutatr_set_spin_limit`

<code>_mutatr_get_spin_limit()</code>	Gets the spin limit of the mutex attributes structure.
<code>_mutatr_set_spin_limit()</code>	Sets the spin limit of the mutex attributes structure.

### Prototype

```
source\kernel\mutex.c
#include <mutex.h>
_mqx_uint _mutatr_get_spin_limit(
    MUTEX_ATTR_STRUCT_PTR attr_ptr,
    _mqx_uint               *spin_count_ptr)

_mqx_uint _mutatr_set_spin_limit(
    MUTEX_ATTR_STRUCT_PTR attr_ptr,
    _mqx_uint             spin_count)
```

### Parameters

*attr\_ptr* [IN] — Pointer to an initialized mutex attributes structure  
*spin\_count\_ptr* [OUT] — Pointer to the current spin limit  
*spin\_count* [IN] — New spin limit

### Returns

- MQX\_EOK (success)
- MQX\_EINVAL (failure: *attr\_ptr* is NULL or points to an invalid attributes structure)

### See Also

[\\_mutatr\\_init](#)

[\\_mutatr\\_get\\_wait\\_protocol](#), [\\_mutatr\\_set\\_wait\\_protocol](#)

[MUTEX\\_ATTR\\_STRUCT](#)

### Description

Spin limit applies only to mutexes whose waiting policy is limited spin. Spin limit is the number of times that a task spins (is rescheduled) while it waits for the mutex.

### Example

```
MUTEX_ATTR_STRUCT mutex_attributes;
_mqx_uint spin;
...
if (_mutatr_init(&mutex_attributes) != MQX_EOK) {
    result = _mutatr_set_wait_protocol(&mutex_attributes,
        MUTEX_LIMITED_SPIN);
    result = _mutatr_set_spin_limit(&mutex_attributes, 20);
    ...
}
```



```
result = _mutatr_get_spin_limit(&mutex_attributes, &spin);
if (result == MQX_EOK) {
    printf("\nSpin count is %ld", spin);
    result = _mutex_init(&mutex, &mutex_attributes);
}
}
```

## 2.1.192 `_mutatr_get_wait_protocol`, `_mutatr_set_wait_protocol`

<code>_mutatr_get_wait_protocol()</code>	Gets the waiting policy of the mutex attributes structure.
<code>_mutatr_set_wait_protocol()</code>	Sets the waiting policy of the mutex attributes structure.

### Prototype

```
source\kernel\mutex.c
#include <mutex.h>
_mqx_uint _mutatr_get_wait_protocol(
    MUTEX_ATTR_STRUCT_PTR attr_ptr,
    _mqx_uint               *waiting_protocol_ptr)

_mqx_uint _mutatr_set_wait_protocol(
    MUTEX_ATTR_STRUCT_PTR attr_ptr,
    _mqx_uint               waiting_protocol)
```

### Parameters

*attr\_ptr* [IN] — Pointer to an initialized mutex attributes structure  
*waiting\_protocol\_ptr* [OUT] — Pointer to the current waiting protocol  
*waiting\_protocol* [IN] — New waiting protocol (see waiting protocols)

### Returns

- MQX\_EOK (success)
- MQX\_EINVAL (failure: *attr\_ptr* is NULL or points to an invalid attribute structure)

### See Also

[\\_mutatr\\_init](#)

[\\_mutatr\\_get\\_spin\\_limit](#), [\\_mutatr\\_set\\_spin\\_limit](#)

[MUTEX\\_ATTR\\_STRUCT](#)

## 2.1.192.1 Waiting protocols

Waiting Protocols	Description
MUTEX_SPIN_ONLY	If the mutex is already locked, MQX RTOS timeslices the task until another task unlocks the mutex
MUTEX_LIMITED_SPIN	If the mutex is already locked, MQX RTOS timeslices the task for a number of times before the lock attempt fails. If this is set, the spin limit should be set.
MUTEX_QUEUEING	If the mutex is already locked, MQX RTOS blocks the task until another task unlocks the mutex, at which time MQX RTOS gives the mutex to the first task that requested it.
MUTEX_PRIORITY_QUEUEING	If the mutex is already locked, MQX RTOS blocks the task until another task unlocks the mutex, at which time MQX RTOS gives the mutex to the highest-priority task that is waiting for it.

### Example

See `_mutatr_get_spin_limit()`.

## 2.1.193 \_mutatr\_init

Initializes the mutex attributes structure to default values.

### Prototype

```
source\kernel\mutex.c
#include <mutex.h>
_mqx_uint _mutatr_init(
    MUTEX_ATTR_STRUCT_PTR attr_ptr)
```

### Parameters

*attr\_ptr [IN]* — Pointer to the mutex attributes structure to initialize

### Returns

- MQX\_EOK (success)
- MQX\_EINVAL (failure: *attr\_ptr* is *NULL*)

### See Also

[\\_mutex\\_init](#)

[\\_mutatr\\_destroy](#)

## MUTEX\_ATTR\_STRUCT

### Description

The function initializes the mutex attributes structure to default values and validates the structure. It must be called before a task can modify the values of the mutex attributes structure.

The function does not affect any mutexes already initialized with this structure.

Mutex attribute	Field in MUTEX_ATTR_STRUCT	Default value
Scheduling protocol	POLICY	MUTEX_NO_PRIO_INHERIT
—	VALID	TRUE
Priority	PRIORITY	0
Spin limit	COUNT	0
Waiting protocol	WAITING_POLICY	MUTEX_QUEUEING

### Example

See [\\_mutatr\\_get\\_spin\\_limit\(\)](#).

## 2.1.194 **\_mutex\_create\_component**

Creates the mutex component.

### Prototype

```
source\kernel\mutex.c  
#include <mutex.h>  
_mqx_uint  _mutex_create_component(void)
```

### Parameters

None

### Returns

- MQX\_OK (success)
- MQX\_OUT\_OF\_MEMORY (failure)

### SeeAlso

[\\_mutex\\_init](#)

[\\_mutatr\\_init](#)

### Description

MQX RTOS calls the function if the mutex component is not created when a task calls **\_mutex\_init()**.

## 2.1.195 \_mutex\_destroy

Deinitializes the mutex.

### Prototype

```
source\kernel\mutex.c
#include <mutex.h>
_mqx_uint _mutex_destroy(
    MUTEX_STRUCT_PTR  mutex_ptr)
```

### Parameters

*mutex\_ptr* [IN] — Pointer to the mutex to be deinitialized

### Returns

- MQX\_EOK
- Errors

Error	Description
MQX_EINVAL	<i>mutex_ptr</i> does not point to a valid mutex (mutex is locked).
MQX_INVALID_COMPONENT_BASE	Mutex component data is not valid.
MQX_COMPONENT_DOES_NOT_EXIST	Mutex component is not created.

### Traits

Puts in their ready queues all tasks that are waiting for the mutex; their call to `_mutex_lock()` returns MQX\_EINVAL

### See Also

[\\_mutex\\_init](#)

### Description

To reuse the mutex, a task must reinitialize it.

## 2.1.196 `_mutex_get_priority_ceiling`, `_mutex_set_priority_ceiling`

<code>_mutex_get_priority_ceiling()</code>	Gets the priority of the mutex.
<code>_mutex_set_priority_ceiling()</code>	Sets the priority of the mutex.

### Prototype

```
source\kernel\mutex.c
#include <mutex.h>
_mqx_uint _mutex_get_priority_ceiling(
    MUTEX_STRUCT_PTR  mutex_ptr,
    _mqx_uint          *priority_ptr)

_mqx_uint _mutex_set_priority_ceiling(
    MUTEX_STRUCT_PTR  mutex_ptr,
    _mqx_uint          priority,
    _mqx_uint          *old_priority_ptr)
```

### Parameters

*mutex\_ptr* [IN] — Pointer to the mutex  
*priority\_ptr* [OUT] — Pointer to the current priority  
*priority* [IN] — New priority  
*old\_priority\_ptr* [OUT] — Pointer to the previous priority

### Returns

- MQX\_EOK
- Errors

### Errors

- MQX\_EINVAL — One of the following:
  - *mutex\_ptr* does not point to a valid mutex structure
  - *priority\_ptr* is *NULL*

### See Also

[\\_mutex\\_init](#)

### Description

The functions operate on an initialized mutex; whereas, `_mutatr_get_priority_ceiling()` and `_mutatr_set_priority_ceiling()` operate on an initialized mutex attributes structure.

## Example

```
MUTEX_STRUCT  mutex;
_mqx_uint     priority;

if (_mutex_set_priority_ceiling(&mutex, 6, &priority) == MQX_EOK){
    result = _mutex_get_priority_ceiling(&mutex, &priority);
    if (result == MQX_EOK) {
        printf("\nCurrent priority of mutex is %lx", priority);
    }
}
```



## 2.1.197 `_mutex_get_wait_count`

Gets the number of tasks that are waiting for the mutex.

### Prototype

```
source\kernel\mutex.c
#include <mutex.h>
_mqx_uint _mutex_get_wait_count(
    MUTEX_STRUCT_PTR  mutex_ptr)
```

### Parameters

*mutex\_ptr* [IN] — Pointer to the mutex

### Returns

- Number of tasks that are waiting for the mutex (success)
- MAX\_MQX\_UINT (failure)

### Traits

On failure, calls `_task_set_error()` to set the task error code to `MQX_EINVAL`

### See Also

[\\_mutex\\_lock](#)

[\\_task\\_set\\_error](#)

## 2.1.198 \_mutex\_init

Initializes the mutex.

### Prototype

```
source\kernel\mutex.c
#include <mutex.h>
_mqx_uint _mutex_init(
    MUTEX_STRUCT_PTR    mutex_ptr,
    MUTEX_ATTR_STRUCT_PTR attr_ptr)
```

### Parameters

*mutex\_ptr* [IN] — Pointer to the mutex to be initialized

*attr\_ptr* [IN] — One of the following:

pointer to an initialized mutex attributes structure

NULL (use default attributes as defined for \_mutatr\_init())

### Returns

- MQX\_EOK
- Errors

Error	Description
MQX_EINVAL	One of the following: <ul style="list-style-type: none"> <li>• <i>mutex_ptr</i> is <i>NULL</i></li> <li>• <i>attr_ptr</i> is not initialized</li> <li>• a value in <i>attr_ptr</i> is not correct</li> </ul>
MQX_INVALID_COMPONENT_BASE	Mutex component data is not valid.

### Traits

Creates the mutex component if it was not previously created

### See Also

[\\_mutex\\_destroy](#)

[\\_mutatr\\_init](#)

### Example

See [\\_mutatr\\_get\\_spin\\_limit\(\)](#).

## 2.1.199 \_mutex\_lock

Locks the mutex.

### Prototype

```
source\kernel\mutex.c
#include <mutex.h>
_mqx_uint _mutex_lock(
    MUTEX_STRUCT_PTR  mutex_ptr)
```

### Parameters

*mutex\_ptr* [IN] — Pointer to the mutex to be locked

### Returns

- MQX\_EOK
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_EBUSY	Mutex is already locked.
MQX_EDEADLK	Task already has the mutex locked.
MQX_EINVAL	One of the following: <ul style="list-style-type: none"> <li>• <i>mutex_ptr</i> is <i>NULL</i></li> <li>• mutex was destroyed</li> </ul>

### Traits

- Might block the calling task
- Cannot be called from an ISR

### See Also

[\\_mutex\\_init](#)

[\\_mutex\\_try\\_lock](#)

[\\_mutex\\_unlock](#)

[\\_mutatr\\_init](#)

[\\_mutatr\\_get\\_wait\\_protocol](#), [\\_mutatr\\_set\\_wait\\_protocol](#)

[\\_mutex\\_destroy](#)

### Description

If the mutex is already locked, the task waits according to the waiting protocol of the mutex.

**Example**

```
MUTEX_STRUCT mutex;  
...  
result = _mutex_lock(&mutex);  
if (result == MQX_EOK) {  
    ...  
    result = _mutex_unlock(&mutex);  
}
```

## 2.1.200 \_mutex\_test

Tests the mutex component.

### Prototype

```
source\kernel\mutex.c
#include <mutex.h>
_mqx_uint _mutex_test(
    void *mutex_error_ptr)
```

### Parameters

*mutex\_error\_ptr* [OUT] — See description

### Returns

See description

### Traits

Disables and enables interrupts

### See Also

[\\_mutex\\_create\\_component](#)

[\\_mutex\\_init](#)

### Description

The function tests:

- mutex component data
- MQX RTOS queue of mutexes
- each mutex
- waiting queue of each mutex

Return value	Meaning	mutex_error_ptr
<b>MQX_OK</b>	No errors were found	NULL
<b>MQX_CORRUPT_QUEUE</b>	Queue of mutexes is not valid	Pointer to the invalid queue
<b>MQX_EINVAL</b>	One of: <ul style="list-style-type: none"> <li>• a mutex is not valid</li> <li>• a mutex queue is not valid</li> </ul>	Pointer to the mutex with the error
<b>MQX_INVALID_COMPONENT_BASE</b>	Mutex component data is not valid	NULL

## Example

```
void *mutex_ptr;  
...  
if (_mutex_test(&mutex_ptr) != MQX_EOK) {  
    printf("Mutex component failed test. Mutex 0x%lx is not valid.",  
        mutex_ptr);  
    _mqx_exit();  
}
```

## 2.1.201 \_mutex\_try\_lock

Tries to lock the mutex.

### Prototype

```
source\kernel\mutex.c
#include <mutex.h>
_mqx_uint _mutex_try_lock(
    MUTEX_STRUCT_PTR  mutex_ptr)
```

### Parameters

*mutex\_ptr* [IN] — Pointer to the mutex

### Returns

- MQX\_EOK
- Errors

Error	Description
MQX_EBUSY	Mutex is currently locked.
MQX_EDEADLK	Task already has the mutex locked.
MQX_EINVAL	One of the following: <ul style="list-style-type: none"> <li>• mutex_ptr is NULL</li> <li>• mutex has been destroyed</li> </ul>

### See Also

[\\_mutex\\_create\\_component](#)

[\\_mutex\\_init](#)

[\\_mutex\\_lock](#)

[\\_mutex\\_unlock](#)

[\\_mutatr\\_init](#)

### Description

If the mutex is not currently locked, the task locks it. If the mutex is currently locked, the task continues to run; it does not block.

### Example

```
MUTEX_STRUCT mutex;
...
result = _mutex_try_lock(&mutex);
if (result == MQX_EOK) {
    ...
    result = _mutex_unlock(&mutex);
}
```

## 2.1.202 \_\_mutex\_unlock

Unlocks the mutex.

### Prototype

```
source\kernel\mutex.c
#include <mutex.h>
_mqx_uint __mutex_unlock(
    MUTEX_STRUCT_PTR  mutex_ptr)
```

### Parameters

*mutex\_ptr* [IN] — Pointer to the mutex

### Returns

- MQX\_EOK (success)
- MQX\_EINVAL (failure: mutex\_ptr does not point to a valid mutex)

### Traits

Might put a task in the task's ready queue

### See Also

[\\_\\_mutex\\_create\\_component](#)

[\\_\\_mutex\\_init](#)

[\\_\\_mutex\\_lock](#)

[\\_\\_mutex\\_try\\_lock](#)

[\\_\\_mutatr\\_init](#)

### Description

If tasks are waiting for the mutex, MQX RTOS removes the first one from the mutex queue and puts the task in the task's ready queue.

### Example

See `__mutex_lock()`.



## 2.1.203 `_name_add`

Adds the name and its associated number to the names database.

### Prototype

```
source\kernel\name.c
#include <name.h>
_mqx_uint _name_add(
    char      *name,
    _mx_max_type number)
```

### Parameters

*name* [IN] — Name to add  
*number* [IN] — Number to be associated with the name

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_INVALID_COMPONENT_BASE	Name component data is not valid.
MQX_OUT_OF_MEMORY	MQX RTOS cannot allocate memory for the name component.
NAME_EXISTS	Name is already in the names database.
NAME_TABLE_FULL	Names database is full.
NAME_TOO_LONG	Name is longer than NAME_MAX_NAME_SIZE.
NAME_TOO_SHORT	Name is \0.

### Traits

- Creates the name component with default values if it was not previously created
- Cannot be called from an ISR

### See Also

[\\_name\\_create\\_component](#)

[\\_name\\_delete](#)

[\\_name\\_find](#)

### Example

See `_name_create_component()`.

## 2.1.204 `_name_create_component`

Creates the name component.

### Prototype

```
source\kernel\name.c
#include <name.h>
_mqx_uint _name_create_component(
    _mx_uint initial_number,
    _mx_uint grow_number,
    _mx_uint maximum_number)
```

### Parameters

*initial\_number* [IN] — Initial number of names that can be stored

*grow\_number* [IN] — Number of the names to add if the initial number are stored

*maximum\_number* [IN] — If *grow\_number* is not 0; one of the following:

maximum number of names

0 (unlimited number)

### Returns

Error	Description
MQX_OK	Success; one of: <ul style="list-style-type: none"> <li>name component is created</li> <li>name component was already created</li> </ul>
MQX_OUT_OF_MEMORY	Failure: MQX RTOS cannot allocate memory for the name component.

### See Also

[`\_name\_add`](#)

[`\_name\_delete`](#)

[`\_name\_find`](#)

### Description

If an application previously called the function and *maximum\_number* is greater than what was specified, MQX RTOS changes the maximum number of names to *maximum\_number*.

If an application does not explicitly create the name component, MQX RTOS does so with the following default values the first time that a task calls `_name_add()`.

Parameter	Default
<i>initial_number</i>	8

grow_number	8
maximum_number	0 (unlimited)

### Example

```

_mqx_uint result;
...
/* Create name component with initially 5 names allowed, adding
** additional names in groups of 5, and limiting the total to 30:
*/
result = _name_create_component(5, 5, 30);
if (result != MQX_OK) {
    /* An error was found. */
    return result;
}
result = _name_add("TASK_A_Q", (_mqx_max_type)my_qid);
...
result = _name_find("TASK_A_Q", &value);
if (result == MQX_OK) {
    qid = (_queue_id)value;
}
...
result = _name_delete("TASK_A_Q");

```

## 2.1.205 `_name_delete`

Deletes the name and its associated number from the names database.

### Prototype

```
source\kernel\name.c
#include <name.h>
_mqx_uint _name_delete(
    char *name)
```

### Parameters

*name* [IN] — Name to delete

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_COMPONENT_DOES_NOT_EXIST	Name component is not created.
MQX_INVALID_COMPONENT_BASE	Name component data is not valid.
NAME_NOT_FOUND	Name is not in the names database.

### Traits

Cannot be called from an ISR

### See Also

[\\_name\\_add](#)

[\\_name\\_create\\_component](#)

[\\_name\\_find](#)

### Example

See `_name_create_component()`.

## 2.1.206 `_name_find`

Gets the number that is associated with the name in the names database.

### Prototype

```
source\kernel\name.c
#include <name.h>
_mqx_uint _name_find(
    char          *name,
    _mxq_max_type_ptr number_ptr)
```

### Parameters

*name* [IN] — Pointer to the name for which to get the associated number

*number\_ptr* [OUT] — Pointer to the number

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_COMPONENT_DOES_NOT_EXIST	Name component is not created.
MQX_INVALID_COMPONENT_BASE	Name component data is not valid.
NAME_NOT_FOUND	Name is not in the names database.

### See Also

[\\_name\\_add](#)

[\\_name\\_create\\_component](#)

[\\_name\\_delete](#)

### Example

See `_name_create_component()`.

## 2.1.207 `_name_find_by_number`

Gets the name that is associated with the number in the names database.

### Prototype

```
source\kernel\name.c
#include <name.h>
_mqx_uint _name_find_by_number(
    _mqx_max_type number,
    char          *name_ptr)
```

### Parameters

*number* [IN] — Number for which to get the associated name

*name\_ptr* [OUT] — Pointer to the name

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_INVALID_COMPONENT_BASE	Name component data is not valid.
NAME_NOT_FOUND	Number is not in the names database.

### See Also

[\\_name\\_add](#)

[\\_name\\_create\\_component](#)

[\\_name\\_delete](#)

### Description

The function finds the first entry in the database that matches the number and returns its name.

## 2.1.208 `_name_test`

Tests name component.

### Prototype

```
source\kernel\name.c
#include <name.h>
_mqx_uint _name_test(
    void *base_error_ptr,
    void *ext_error_ptr)
```

### Parameters

*base\_error\_ptr* [OUT] — See description

*ext\_error\_ptr* [OUT] — See description

### Returns

- MQX\_OK
- See description

### Traits

Disables and enables interrupts

### See Also

[\\_name\\_add](#)

[\\_name\\_create\\_component](#)

[\\_name\\_delete](#)

### Description

The function tests the data structures that are associated with the name component.

Return	<i>base_error_ptr</i>	<i>ext_error_ptr</i>
<b>MQX_CORRUPT_QUEUE</b> (Task queue that is associated with the name component is incorrect)	NULL	NULL
<b>MQX_INVALID_COMPONENT_BASE</b> (MQX RTOS found an error in a name component data structure)	Pointer to the name table that has an error	Pointer to the name table that has an error

### Example

```
_mqx_uint    result;
void         *table_ptr;
```

```
void          *error_ptr;

result = _name_test(&table_ptr, &error_ptr);
if (result != MQX_OK) {
    /* Name component is not valid. */
}
```



## 2.1.209 `_partition_alloc`, `_partition_alloc_zero`

<code>_partition_alloc()</code>	Allocates a private partition block from the partition.
<code>_partition_alloc_zero()</code>	Allocates a zero-filled private partition block from the partition.

### Prototype

```
source\kernel\partition.c
#include <partition.h>
void *_partition_alloc(
    _partition_id partition_id)

void *_partition_alloc_zero(
    _partition_id partition_id)
```

### Parameters

*partition\_id* [IN] — Partition from which to allocate the partition block

### Returns

- Pointer to the partition block (success)
- NULL (failure)

### Task Error Codes

Task Error Codes	Description
PARTITION_BLOCK_INVALID_CHECKSUM	MQX RTOS found an incorrect checksum in the partition block header.
PARTITION_INVALID	<i>partition_id</i> does not represent a valid partition.
PARTITION_OUT_OF_BLOCKS	All the partition blocks in the partition are allocated (for static partitions only).
Task error code set by <code>_mem_alloc_system()</code>	MQX RTOS cannot allocate memory for the partition block (for dynamic partitions only).

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_partition\\_alloc\\_system](#), [\\_partition\\_alloc\\_system\\_zero](#)

[\\_partition\\_create](#)

[\\_task\\_set\\_error](#)

[\\_mem\\_alloc ...](#)

## Description

The functions allocate a fixed-size memory block, which the task owns.

## Example

Create a dynamic partition, allocate a private partition block, and then free the block.

```
#include <mqx.h>
#include <partition.h>

#define PACKET_SIZE      0x200
#define PACKET_COUNT     100

void part_function(void)
{
    _partition_id  packet_partition;
    void          *packet_ptr;

    /* Create a dynamic partition: */
    packet_partition = _partition_create(PACKET_SIZE, PACKET_COUNT,
        0, 0);
    ...
    /* Allocate a partition block: */
    packet_ptr = _partition_alloc(packet_partition);
    ...
    /* Free the partition block: */
    _partition_free(packet_ptr);
}
```

## 2.1.210 `_partition_alloc_system`, `_partition_alloc_system_zero`

<code>_partition_alloc_system()</code>	Allocates a system partition block from the partition.
<code>_partition_alloc_system_zero()</code>	Allocates a zero-filled system partition block from the partition.

### Prototype

```
source\kernel\partition.c
#include <partition.h>
void *_partition_alloc_system(
    _partition_id partition_id)

void *_partition_alloc_system_zero(
    _partition_id partition_id)
```

### Parameters

*partition\_id* [IN] — Partition from which to allocate the partition block

### Returns

- Pointer to the partition block (success)
- NULL (failure)

### Traits

On failure, calls `_task_set_error()` to set the task error code as described for `_partition_alloc()`

### See Also

[\\_partition\\_alloc](#), [\\_partition\\_alloc\\_zero](#)

[\\_partition\\_create](#)

[\\_task\\_set\\_error](#)

### Description

The functions allocate a fixed-size block of memory that is not owned by any task.

## 2.1.211 `_partition_calculate_blocks`

Calculates the number of partition blocks in a static partition.

### Prototype

```
source\kernel\partition.c
#include <partition.h>
_mqx_uint _partition_calculate_blocks(
    _mem_size partition_size,
    _mem_size block_size)
```

### Parameters

*partition\_size [IN]* — Number of single-addressable units that the partition can occupy

*block\_size [IN]* — Number of single-addressable units in one partition block of the partition

### Returns

Number of partition blocks in the partition

### See Also

[`\_partition\_calculate\_size`](#)

[`\_partition\_create\_at`](#)

### Description

When a task creates a static partition (`_partition_create_at()`), it specifies the size of the partition and the size of partition blocks. The function `_partition_calculate_blocks()` calculates how many blocks MQX RTOS actually created, taking into account internal headers.

## 2.1.212 `_partition_calculate_size`

Calculates the number of single-addressable units in a partition.

### Prototype

```
source\kernel\partition.c
#include <partition.h>
_mem_size _partition_calculate_size(
    _mqx_uint  number_of_blocks,
    _mem_size  block_size)
```

### Parameters

*number\_of\_blocks [IN]* — Number of partition blocks in the partition

*block\_size [IN]* — Number of single-addressable units in one partition block in the partition

### Returns

Number of single-addressable units in the partition

### See Also

[\\_partition\\_calculate\\_blocks](#)

[\\_partition\\_create](#)

[\\_partition\\_create\\_at](#)

### Description

If an application wants to use as much as possible of some memory that is outside the default memory pool, it can use the function to determine the maximum number of blocks that can be created.

For a dynamic partition, the application might want to limit (based on the results of the function) the amount of memory in the default memory pool that it uses to create the partition.

## 2.1.213 `_partition_create`

Creates the partition in the default memory pool (a dynamic partition).

### Prototype

```
source\kernel\partition.c
#include <partition.h>
_partition_id _partition_create(
    _mem_size  block_size,
    _mqx_uint  initial_blocks,
    _mqx_uint  grow_blocks,
    _mqx_uint  maximum_blocks)
```

### Parameters

*block\_size* [IN] — Number of single-addressable units in each partition block

*initial\_blocks* [IN] — Initial number of blocks in the partition

*grow\_blocks* [IN] — Number of blocks by which to grow the partition if all the partition blocks are allocated

*maximum\_blocks* [IN] — If *grow\_blocks* is not 0; one of:  
 maximum number of blocks in the partition  
 0 (unlimited growth)

### Returns

- Partition ID (success)
- PARTITION\_NULL\_ID (failure)

### Task Error Codes

- MQX\_INVALID\_PARAMETER — *block\_size* is 0.
- Task error codes returned by [\\_mem\\_alloc ...](#)

### Traits

- Creates the partition component if it were not previously created
- On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_partition\\_alloc](#), [\\_partition\\_alloc\\_zero](#)

[\\_partition\\_alloc\\_system](#), [\\_partition\\_alloc\\_system\\_zero](#)

[\\_partition\\_calculate\\_size](#)

[\\_partition\\_create\\_at](#)

[\\_partition\\_destroy](#)

[\\_task\\_set\\_error](#)

[\\_mem\\_alloc ...](#)

**Description**

The function creates a partition of fixed-size partition blocks in the default memory pool.

**Example**

See `_partition_alloc()`.

## 2.1.214 `_partition_create_at`

Creates the partition at the specific location outside the default memory pool (a static partition).

### Prototype

```
source\kernel\partition.c
#include <partition.h>
_partition_id _partition_create_at(
    void          *partition_location,
    _mem_size     partition_size,
    _mem_size     block_size)
```

### Parameters

*partition\_location* [IN] — Pointer to the start of the partition  
*partition\_size* [IN] — Number of single-addressable units in the partition  
*block\_size* [IN] — Number of single-addressable units in each partition block in the partition

### Returns

- Partition ID (success)
- PARTITION\_NULL\_ID (failure)

### Task Error Codes

- MAX\_INVALID\_PARAMETER — One of the following:
  - *block\_size* is 0
  - *partition\_size* is too small

### Traits

- Creates the partition component if it were not previously created
- On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_partition\\_alloc, \\_partition\\_alloc\\_zero](#)  
[\\_partition\\_alloc\\_system, \\_partition\\_alloc\\_system\\_zero](#)  
[\\_partition\\_calculate\\_size](#)  
[\\_partition\\_create](#)  
[\\_partition\\_extend](#)  
[\\_task\\_set\\_error](#)

### Example

```
#include <mqx.h>
#include <partition.h>

#define PART_SIZE      0x4000
```



```
#define PART_ADDR1    0x200000
#define PART_ADDR2    0x300000
#define PACKET_SIZE   100

void part_function(void)
{
    _partition_id  packet_partition;
    void           *packet_ptr;

    /* Create a static partition: */
    packet_partition =
        _partition_create_at(PART_ADDR1, PART_SIZE, PACKET_SIZE);
    ...
    /* Allocate a partition block: */
    packet_ptr = _partition_alloc(packet_partition);

    /* Extend the partition: */
    if (packet_ptr == NULL) {
        _partition_extend(packet_partition, PART_ADDR1, PART_SIZE);
        packet_ptr = _partition_alloc(packet_partition);
    }
    ...
    /* Free the partition block: */
    _partition_free(packet_ptr);
}
```

## 2.1.215 `_partition_create_component`

Creates the partition component.

### Prototype

```
source\kernel\partition.c
#include <partition.h>
_mqx_uint _partition_create_component(void)
```

### Parameters

None

### Returns

- MQX\_OK (success)
- Errors (failure)

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_OUT_OF_MEMORY	MQX RTOS is out of memory.

### Traits

- Cannot be called from an ISR
- Might block the calling task

### See Also

[`\_partition\_create`](#)

[`\_partition\_destroy`](#)

## 2.1.216 `_partition_destroy`

Destroys a partition that is in the default memory pool (a dynamic partition).

### Prototype

```
source\kernel\partition.c
_mqx_uint _partition_destroy(
    _partition_id partition)
```

### Parameters

*partition\_id* [IN] — Partition ID of the partition to destroy

### Returns

- MQX\_OK
- Errors

Error	Description
Errors from <code>_mem_free()</code>	
MQX_INVALID_PARAMETER	<i>partition_id</i> is invalid.
PARTITION_ALL_BLOCKS_NOT_FREE	There are allocated partition blocks in the partition.
PARTITION_INVALID_TYPE	Partition is not a dynamic partition.

### See Also

[\\_mem\\_free](#)

[\\_partition\\_create](#)

[\\_partition\\_free](#)

### Description

If all the partition blocks in a dynamic partition are first freed, any task can destroy the partition.

## 2.1.217 `_partition_extend`

Adds partition blocks to the static partition.

### Prototype

```
source\kernel\partition.c
#include <partition.h>
_mqx_uint _partition_extend(
    _partition_id  partition_id,
    void          *partition_location,
    _mem_size      partition_size)
```

### Parameters

*partition\_id* [IN] — Static partition to extend  
*partition\_location* [IN] — Pointer to the beginning of the memory to add  
*partition\_size* [IN] — Number of single-addressable units to add

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_INVALID_PARAMETER	One of the following: <ul style="list-style-type: none"> <li>• <i>partition_size</i> is 0</li> <li>• <i>partition_id</i> does not represent a static partition</li> </ul>
PARTITION_INVALID	<i>partition_id</i> does not represent a valid partition.

### See Also

[\\_partition\\_create\\_at](#)

[\\_partition\\_alloc](#), [\\_partition\\_alloc\\_zero](#)

### Description

The function extends a partition that was created with [\\_partition\\_create\\_at\(\)](#). Based on the size of the partition's partition blocks, the function divides the additional memory into partition blocks and adds them to the partition.

### Example

See [\\_partition\\_create\\_at\(\)](#).

## 2.1.218 `_partition_free`

Frees the partition block and returns it to the partition.

### Prototype

```
source\kernel\partition.c
#include <partition.h>
_mqx_uint _partition_free(
    void *mem_ptr)
```

### Parameters

*mem\_ptr* [IN] — Pointer to the partition block to free

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_NOT_RESOURCE_OWNER	Task is not the one that owns the partition block.
PARTITION_BLOCK_INVALID_CHECKSUM	Checksum in the partition block header is not correct; the integrity of the partition is in question.
PARTITION_INVALID	<i>mem_ptr</i> is part of a partition that is not valid.

### See Also

[\\_partition\\_alloc](#), [\\_partition\\_alloc\\_zero](#)

[\\_partition\\_alloc\\_system](#), [\\_partition\\_alloc\\_system\\_zero](#)

[\\_partition\\_create](#)

### Description

If the partition block was allocated by:	It can be freed by:
<code>_partition_alloc()</code> or <code>_partition_alloc_zero()</code>	Task that allocated it
<code>_partition_alloc_system()</code> or <code>_partition_alloc_system_zero()</code>	Any task

### Example

See `_partition_alloc()`.

## 2.1.219 `_partition_get_block_size`

Gets the size of the partition blocks in the partition.

### Prototype

```
source\kernel\partition.c
#include <partition.h>
_mem_size _partition_get_block_size(
    _partition_id partition_id)
```

### Parameters

*partition\_id* [IN] — Partition about which to get info

### Returns

- Number of single-addressable units in a partition block (success)
- 0 (failure)

### Task Error Codes

- PARTITION\_INVALID — *partition\_id* does not represent a valid partition.

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_partition\\_get\\_free\\_blocks](#)

[\\_partition\\_get\\_max\\_used\\_blocks](#)

[\\_partition\\_get\\_total\\_blocks](#)

[\\_partition\\_get\\_total\\_size](#)

[\\_partition\\_create\\_at](#)

[\\_task\\_set\\_error](#)

### Description

If the processor supports memory alignment, the function might return a value that is larger than what was specified when the partition was created.

### Example

Print the attributes of a partition.

```
#include <mqx.h>
#include <partition.h>

void print_partition_info(_partition_id partition)
{
    printf("\nBlock size  %x",
        _partition_get_block_size(partition));
}
```

```
printf("\nFree blocks %x",
    _partition_get_free_blocks(partition));
printf("\nUsed blocks %x",
    _partition_get_max_used_blocks(partition));
printf("\nTotal blocks %x",
    _partition_get_total_blocks(partition));
printf("\nTotal size %x",
    _partition_get_total_size(partition));
}
```

## 2.1.220 `_partition_get_free_blocks`

Gets the number of free partition blocks in the partition.

### Prototype

```
source\kernel\partition.c
#include <partition.h>
_mqx_uint _partition_get_free_blocks(
    _partition_id partition_id)
```

### Parameters

*partition\_id* [IN] — Partition for which to get info

### Returns

- Number of free partition blocks (success)
- MAX\_MQX\_UINT (failure)

### Task Error Codes

- PARTITION\_INVALID — *partition\_id* does not represent a valid partition.

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_partition\\_get\\_block\\_size](#)

[\\_partition\\_get\\_max\\_used\\_blocks](#)

[\\_partition\\_get\\_total\\_blocks](#)

[\\_partition\\_get\\_total\\_size](#)

[\\_task\\_set\\_error](#)

### Example

See `_partition_get_block_size()`.



## 2.1.221 `_partition_get_max_used_blocks`

Gets the number of allocated partition blocks in the partition.

### Prototype

```
source\kernel\partition.c
#include <partition.h>
_mqx_uint _partition_get_max_used_blocks(
    _partition_id partition_id)
```

### Parameters

*partition\_id* [IN] — Partition for which to get info

### Returns

- Number of allocated partition blocks (success)
- 0 (failure)

### Task Error Codes

- PARTITION\_INVALID — *partition\_id* does not represent a valid partition.

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error code)

### See Also

[\\_partition\\_get\\_block\\_size](#)

[\\_partition\\_get\\_free\\_blocks](#)

[\\_partition\\_get\\_total\\_blocks](#)

[\\_partition\\_get\\_total\\_size](#)

[\\_task\\_set\\_error](#)

### Example

See `_partition_get_block_size()`.

## 2.1.222 `_partition_get_total_blocks`

Gets the total number of partition blocks in the partition.

### Prototype

```
source\kernel\partition.c
#include <partition.h>
_mqx_uint _partition_get_total_blocks(
    _partition_id partition_id)
```

### Parameters

*partition\_id* [IN] — Partition for which to get info

### Returns

- Total number of partition blocks in the partition (success)
- 0 (failure)

### Task Error Codes

- PARTITION\_INVALID — *partition\_id* does not represent a valid partition.

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error code)

### See Also

[\\_partition\\_get\\_block\\_size](#)

[\\_partition\\_get\\_free\\_blocks](#)

[\\_partition\\_get\\_max\\_used\\_blocks](#)

[\\_partition\\_get\\_total\\_size](#)

[\\_task\\_set\\_error](#)

### Description

The function returns the sum of the number of free partition blocks and the number of allocated partition blocks in the partition.

### Example

See `_partition_get_block_size()`.

## 2.1.223 `_partition_get_total_size`

Gets the size of the partition.

### Prototype

```
source\kernel\partition.c
#include <partition.h>
_mem_size _partition_get_total_size(
    _partition_id partition_id)
```

### Parameters

*partition\_id* [IN] — Partition for which to get info

### Returns

- Number of single-addressable units in the partition (success)
- 0 (failure)

### Task Error Codes

- PARTITION\_INVALID — *partition\_id* does not represent a valid partition.

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error code)

### See Also

[\\_partition\\_get\\_block\\_size](#)

[\\_partition\\_get\\_free\\_blocks](#)

[\\_partition\\_get\\_max\\_used\\_blocks](#)

[\\_partition\\_get\\_total\\_blocks](#)

[\\_partition\\_extend](#)

[\\_task\\_set\\_error](#)

### Description

The size of the partition includes extensions and internal overhead.

### Example

See `_partition_get_block_size()`.

## 2.1.224 \_partition\_test

Tests all partitions.

### Prototype

```
source\kernel\partition.c
#include <partition.h>
_mqx_uint _partition_test(
    _partition_id *partpool_in_error,
    void *partpool_block_in_error,
    void *block_in_error)
```

### Parameters

*partpool\_in\_error [OUT]* — Pointer to the partition pool in error (initialized only if an error is found)

*partpool\_block\_in\_error [OUT]* — Pointer to the partition pool block in error (internal to MQX RTOS)

*block\_in\_error [OUT]* — Pointer to the partition block in error (initialized only if an error is found)

### Returns

- MQX\_OK (no partitions had errors)
- Errors

Error	Description
PARTITION_BLOCK_INVALID_CHECKSUM	MQX RTOS found a partition block with an incorrect checksum.
PARTITION_INVALID	MQX RTOS found an invalid partition.

### Traits

Disables and enables interrupts

### See Also

[\\_partition\\_alloc](#), [\\_partition\\_alloc\\_zero](#)

[\\_partition\\_alloc\\_system](#), [\\_partition\\_alloc\\_system\\_zero](#)

[\\_partition\\_create](#)

[\\_partition\\_free](#)

## 2.1.225 `_partition_transfer`

Transfers the ownership of the partition block.

### Prototype

```
source\kernel\partition.c
#include <partition.h>
_mqx_uint _partition_transfer(
    void      *mem_ptr,
    _task_id  new_owner_id)
```

### Parameters

*mem\_ptr* [IN] — Pointer to the partition block to transfer

*new\_owner\_id* [IN] — Task ID of new owner

### Returns

- MQX\_OK
- See errors

Error	Description
PARTITION_BLOCK_INVALID_CHECKSUM	Checksum of the partition block header is not correct, which indicates that <i>mem_ptr</i> might not point to a valid partition block.
PARTITION_INVALID_TASK_ID	<i>task_id</i> is not valid.

### See Also

[\\_partition\\_alloc](#), [\\_partition\\_alloc\\_zero](#)

[\\_partition\\_alloc\\_system](#), [\\_partition\\_alloc\\_system\\_zero](#)

### Description

Any task can transfer the ownership of a private partition block or a system partition block.

If *new\_owner\_id* is the System Task ID, the partition block becomes a system partition block.

If the ownership of a system partition block is transferred to a task, the partition block becomes a resource of the task.

## 2.1.226 `_queue_dequeue`

Removes the first element from the queue.

### Prototype

```
source\kernel\queue.c
QUEUE_ELEMENT_STRUCT_PTR _queue_dequeue(
    QUEUE_STRUCT_PTR q_ptr)
```

### Parameters

*q\_ptr* [IN] — Pointer to the queue from which to remove the first element; initialized with `_queue_init()`

### Returns

- Pointer to removed first queue element
- NULL (Queue is empty)

### See Also

[\\_queue\\_enqueue](#)

[\\_queue\\_init](#)

[QUEUE\\_STRUCT](#)

[QUEUE\\_ELEMENT\\_STRUCT](#)

### CAUTION

If *q\_ptr* is not a pointer to **QUEUE\_STRUCT**, the function might behave unpredictably.

### Example

```
typedef struct my_queue_element_struct
{
    QUEUE_ELEMENT_STRUCT  HEADER;
    _mqx_uint             MY_DATA;
} MY_QUEUE_ELEMENT_STRUCT; * MY_QUEUE_ELEMENT_STRUCT_PTR;

MY_QUEUE_ELEMENT_STRUCT_PTR  element_ptr;
MY_QUEUE_ELEMENT_STRUCT      element1;
MY_QUEUE_ELEMENT_STRUCT      element2;
QUEUE_STRUCT                 my_queue;
_mqx_uint                    i;
_mqx_uint                     result;
...
_queue_init(&my_queue, 0);
result = _queue_enqueue(&my_queue,
    (QUEUE_ELEMENT_STRUCT_PTR)&element1);
result = _queue_enqueue(&my_queue,
    (QUEUE_ELEMENT_STRUCT_PTR)&element2);
```

```
...
/* Empty the queue: */
i = _queue_get_size(&my_queue);
while (i) {
    element_ptr =
        (MY_QUEUE_ELEMENT_STRUCT_PTR)_queue_dequeue(&my_queue);
    i--;
}
```

## 2.1.227 `_queue_enqueue`

Adds the element to the end of the queue.

### Prototype

```
source\kernel\queue.c
bool _queue_enqueue(
    QUEUE_STRUCT_PTR    q_ptr,
    QUEUE_ELEMENT_STRUCT_PTR e_ptr)
```

### Parameters

*q\_ptr* [IN] — Pointer to the queue to which to add the element; initialized with `_queue_init()`  
*e\_ptr* [IN] — Pointer to the element to add

### Returns

- TRUE (success)
- FALSE (failure: the queue is full)

### See also

[\\_queue\\_init](#)

[\\_queue\\_dequeue](#)

[\\_queue\\_init](#)

[QUEUE\\_STRUCT](#)

[QUEUE\\_ELEMENT\\_STRUCT](#)

### CAUTION

The function might behave unpredictably if either:

- *q\_ptr* is not a pointer to **QUEUE\_STRUCT**
- *e\_ptr* is not a pointer to **QUEUE\_ELEMENT\_STRUCT**

### Example

See `_queue_dequeue()`.



## 2.1.228 `_queue_get_size`

Gets the number of elements in the queue.

### Prototype

```
source\kernel\queue.c  
_mqx_uint _queue_get_size(  
    QUEUE_STRUCT_PTR q_ptr)
```

### Parameters

*q\_ptr* [IN] — Pointer to the queue for which to get info; initialized with `_queue_init()`

### Returns

Number of elements in the queue

### See Also

[\\_queue\\_enqueue](#)

[\\_queue\\_init](#)

[QUEUE\\_STRUCT](#)

### CAUTION

If *q\_ptr* is not a pointer to `QUEUE_STRUCT`, the function might behave unpredictably.

### Example

See `_queue_insert()`.

## 2.1.229 `_queue_head`

Gets a pointer to the element at the start of the queue, but do not remove the element.

### Prototype

```
source\kernel\queue.c  
QUEUE_ELEMENT_STRUCT_PTR _queue_head(  
    QUEUE_STRUCT_PTR    q_ptr)
```

### Parameters

*q\_ptr* [IN] — Pointer to the queue to use; initialized with `_queue_init()`

### Returns

- Pointer to the element that is at the start of the queue
- NULL (queue is empty)

### See Also

[\\_queue\\_dequeue](#)

[\\_queue\\_init](#)

[QUEUE\\_STRUCT](#)

[QUEUE\\_ELEMENT\\_STRUCT](#)

### CAUTION

If *q\_ptr* is not a pointer to `QUEUE_STRUCT`, the function might behave unpredictably.

### Example

See `_queue_insert()`.

## 2.1.230 \_queue\_init

Initializes the queue.

### Prototype

```
source\kernel\queue.c  
void _queue_init(  
    QUEUE_STRUCT_PTR q_ptr,  
    uint16_t size)
```

### Parameters

*q\_ptr* [IN] — Pointer to the queue to initialize

*size* [IN] — One of the following:

maximum number of elements that the queue can hold

0 (unlimited number)

### Returns

None

### See Also

[\\_queue\\_enqueue](#)

[\\_queue\\_dequeue](#)

[QUEUE\\_STRUCT](#)

### CAUTION

If *q\_ptr* is not a pointer to **QUEUE\_STRUCT**, the function might behave unpredictably.

### Example

See [\\_queue\\_insert\(\)](#).

## 2.1.231 `_queue_insert`

Inserts the element in the queue.

### Prototype

```
source\kernel\queue.c
bool _queue_insert(
    QUEUE_STRUCT_PTR      q_ptr,
    QUEUE_ELEMENT_STRUCT_PTR qe_ptr,
    QUEUE_ELEMENT_STRUCT_PTR e_ptr)
```

### Parameters

- q\_ptr* [IN] — Pointer to the queue to insert into; initialized with `_queue_init()`
- qe\_ptr* [IN] — One of the following:
- pointer to the element after which to insert the new element
  - NULL (insert the element at the start of the queue)
- e\_ptr* [IN] — Pointer to the element to insert

### Returns

- TRUE (success)
- FALSE (failure: queue is full)

### See Also

[\\_queue\\_init](#)

[QUEUE\\_STRUCT](#)

[QUEUE\\_ELEMENT\\_STRUCT](#)

### CAUTION

The function might behave unpredictably if either:

- *q\_ptr* is not a pointer to **QUEUE\_STRUCT**
- *e\_ptr* is not a pointer to **QUEUE\_ELEMENT\_STRUCT**

### Example

Insert an element into a queue using a particular sorting algorithm.

```
typedef struct my_queue_element_struct
{
    QUEUE_ELEMENT_STRUCT  HEADER;
    _mqx_uint             MY_DATA;
} MY_QUEUE_ELEMENT_STRUCT, * MY_QUEUE_ELEMENT_STRUCT_PTR;

void my_queue_insert(MY_QUEUE_ELEMENT_STRUCT_PTR  connection_ptr)
{
    MY_QUEUE_ELEMENT_STRUCT_PTR  conn2_ptr;
    MY_QUEUE_ELEMENT_STRUCT_PTR  conn_prev_ptr;
```

```

QUEUE_STRUCT          queue;
QUEUE_STRUCT          *queue_ptr;
_mqx_uint             count;

queue_ptr = &queue;
_queue_init(queue_ptr, 0);

/* If the queue is empty, simply enqueue the element: */
if (_queue_is_empty(queue_ptr)) {
    _queue_enqueue(queue_ptr,
        (QUEUE_ELEMENT_STRUCT_PTR)connection_ptr);
    return;
}
/* Search the queue for the particular location to put
   the element: */
conn_prev_ptr =
    (MY_QUEUE_ELEMENT_STRUCT_PTR)_queue_head(queue_ptr);
conn2_ptr      =
    (MY_QUEUE_ELEMENT_STRUCT_PTR)_queue_next(queue_ptr,
        (QUEUE_ELEMENT_STRUCT_PTR)conn_prev_ptr);
count          = _queue_get_size(queue_ptr) + 1;
while (--count) {
    ...
    if (/* found the location, */) {
        break;
    }
    conn_prev_ptr = conn2_ptr;
    conn2_ptr      = _queue_next(queue_ptr,
        (QUEUE_ELEMENT_STRUCT_PTR)conn2_ptr);
}

_queue_insert(queue_ptr,
    (QUEUE_ELEMENT_STRUCT_PTR)conn_prev_ptr,
    (QUEUE_ELEMENT_STRUCT_PTR)connection_ptr);
...
}

```

## 2.1.232 `_queue_is_empty`

Determines whether the queue is empty.

### Prototype

```
source\kernel\queue.c  
bool _queue_is_empty(  
    QUEUE_STRUCT_PTR  q_ptr)
```

### Parameters

*q\_ptr* [IN] — Pointer to the queue for which to get info; initialized with `_queue_init()`

### Returns

- TRUE (queue is empty)
- FALSE (queue is not empty)

### See Also

[\\_queue\\_init](#)

[QUEUE\\_STRUCT](#)

### CAUTION

If *q\_ptr* is not a pointer to `QUEUE_STRUCT`, the function might behave unpredictably.

### Example

See `_queue_insert()`.

### 2.1.233 `_queue_next`

Gets a pointer to the element after this one in the queue, but do not remove the element.

#### Prototype

```
source\kernel\queue.c
QUEUE_ELEMENT_STRUCT_PTR _queue_next(
    QUEUE_STRUCT_PTR      q_ptr,
    QUEUE_ELEMENT_STRUCT_PTR e_ptr)
```

#### Parameters

*q\_ptr* [IN] — Pointer to the queue for which to get info; initialized with `_queue_init()`  
*e\_ptr* [IN] — Get the element after this one

#### Returns

- Pointer to the next queue element (success)
- NULL (failure: see description)

#### See Also

[\\_queue\\_init](#)

[\\_queue\\_dequeue](#)

[QUEUE\\_STRUCT](#)

[QUEUE\\_ELEMENT\\_STRUCT](#)

#### CAUTION

The function might behave unpredictably if either:

- *q\_ptr* is not a pointer to **QUEUE\_STRUCT**
- *e\_ptr* is not a pointer to **QUEUE\_ELEMENT\_STRUCT**

#### Description

The function returns *NULL* if either:

- *e\_ptr* is *NULL*
- *e\_ptr* is a pointer to the last element

#### Example

See `_queue_insert()`.

## 2.1.234 `_queue_test`

Tests the queue.

### Prototype

```
source\kernel\queue.c
_mqx_uint _queue_test(
    QUEUE_STRUCT_PTR q_ptr,
    void *element_in_error_ptr)
```

### Parameters

*q\_ptrm [IN]* — Pointer to the queue to test; initialized with `_queue_init()`

*element\_in\_error\_ptr [OUT]* — Pointer to the first element with an error (initialized only if an error is found)

### Returns

- MQX\_OK (no errors are found)
- MQX\_CORRUPT\_QUEUE (an error is found)

### See Also

[\\_queue\\_init](#)

[QUEUE\\_STRUCT](#)

[QUEUE\\_ELEMENT\\_STRUCT](#)

### Description

The function checks the queue pointers to ensure that they form a circular, doubly linked list, with the same number of elements that the queue header specifies.

### Example

Test a mutex's queue.

```
result = _queue_test(&mutex_ptr->WAITING_TASKS, mutex_error_ptr);
if (result != MQX_OK) {
    /* An error occurred. */
    ...
}
```



## 2.1.235 `_queue_unlink`

Removes the element from the queue.

### Prototype

```
source\kernel\queue.c
void _queue_unlink(
    QUEUE_STRUCT_PTR    q_ptr,
    QUEUE_ELEMENT_STRUCT_PTR e_ptr)
```

### Parameters

*q\_ptr* [IN] — Pointer to the queue from which to remove the element; initialized with `_queue_init()`

*e\_ptr* [IN] — Pointer to the element to remove

### Returns

None

### See Also

[\\_queue\\_init](#)

[\\_queue\\_dequeue](#)

[QUEUE\\_STRUCT](#)

[QUEUE\\_ELEMENT\\_STRUCT](#)

### CAUTION

The function might behave unpredictably if either:

- *q\_ptr* is not a pointer to **QUEUE\_STRUCT**
- *e\_ptr* is not a pointer to **QUEUE\_ELEMENT\_STRUCT**

### Example

Remove an element from its queue if processing for it is finished.

```
typedef struct my_queue_element_struct
{
    QUEUE_ELEMENT_STRUCT  HEADER;
    _mqx_uint             MY_DATA;
    bool                  FINISHED;
} MY_QUEUE_ELEMENT_STRUCT;

MY_QUEUE_ELEMENT_STRUCT element;
QUEUE_STRUCT            my_queue;

...

if (element.FINISHED) {
    _queue_unlink(&my_queue, (QUEUE_ELEMENT_STRUCT_PTR)&element);
}
```

## 2.1.236 `_sched_get_max_priority`

Gets the maximum priority that a task can be.

### Prototype

```
source\kernel\sched.c  
_mqx_uint _sched_get_max_priority(  
    _mqx_uint policy)
```

### Parameters

*policy* — Not used

### Returns

0 (always)

### See Also

[\\_sched\\_get\\_min\\_priority](#)

### Description

POSIX compatibility requires the function and the parameter.

### Example

```
_mqx_uint highest_priority;  
...  
highest_priority = _sched_get_max_priority(MQX_SCHED_RR);
```

## 2.1.237 `_sched_get_min_priority`

Gets the minimum priority that an application task can be.

### Prototype

```
source\kernel\sched.c  
_mqx_uint _sched_get_min_priority(  
    _mqx_uint policy)
```

### Parameters

*policy* — Not used

### Returns

Minimum priority that an application task can be (the numerical value one less than the priority of Idle Task)

### See also

[\\_sched\\_get\\_max\\_priority](#)

### Description

POSIX compatibility requires the function and the parameter.

The minimum priority that a task can be is set when MQX RTOS starts; it is the priority of the lowest-priority task in the task template list.

### Example

```
_mqx_uint minimum_task_priority;  
...  
minimum_task_priority = _sched_get_min_priority(MQX_SCHED_RR);
```

## 2.1.238 `_sched_get_policy`

Gets the scheduling policy.

### Prototype

```
source\kernel\sched.c
_mqx_uint _sched_get_policy(
    _task_id      task_id,
    _mqx_uint      *policy_ptr)
```

### Parameters

*task\_id* [IN] — One of the following:

task on this processor for which to get info

**MQX\_DEFAULT\_TASK\_ID** (get the policy for the processor)

**MQX\_NULL\_TASK\_ID** (get the policy for the calling task)

*policy\_ptr* [OUT] — Pointer to the scheduling policy (see scheduling policies)

### Returns

- MQX\_OK (success)
- MQX\_SCHED\_INVALID\_TASK\_ID (failure: task\_id is not a valid task on this processor)

### See also

[\\_sched\\_set\\_policy](#)

### Scheduling Policies

- MQX\_SCHED\_FIFO — FIFO
- MQX\_SCHED\_RR — Round robin.

### Example

Set the scheduling policy to round robin for the active task and verify the change.

```
_mqx_uint policy;
...
policy = _sched_set_policy(_task_get_id(), MQX_SCHED_RR);
...
result = _sched_get_policy(_task_get_id(), &policy);
```

## 2.1.239 `_sched_get_rr_interval`, `_sched_get_rr_interval_ticks`

Get the time slice in:	
<code>_sched_get_rr_interval()</code>	Milliseconds
<code>_sched_get_rr_interval_ticks()</code>	Tick time

### Prototype

```
source\kernel\sched.c
uint32_t _sched_get_rr_interval(
    _task_id    task_id,
    uint32_t    *ms_ptr)

_mqx_uint _sched_get_rr_interval_ticks(
    _task_id    task_id,
    MQX_TICK_STRUCT_PTR tick_time_ptr)
```

### Parameters

*task\_id* [IN] — One of the following:

task on this processor for which to get info

MQX\_DEFAULT\_TASK\_ID (get the time slice for the processor)

MQX\_NULL\_TASK\_ID (get the time slice for the calling task)

*ms\_ptr* [OUT] — Pointer to the time slice (in milliseconds)

*tick\_time\_ptr* [OUT] — Pointer to the time slice (in tick time)

### Returns

- MQX\_OK (success)
- MAX\_MQX\_UINT (`_sched_get_rr_interval()` failure)
- See task error codes (`_sched_get_rr_interval_ticks()` failure)

### Task Error Codes

- MQX\_SCHED\_INVALID\_PARAMETER\_PTR — *time\_ptr* is *NULL*.
- MQX\_SCHED\_INVALID\_TASK\_ID — *task\_id* is not a valid task on this processor.

### Traits

On failure, calls `_task_set_error()` to set the task error codes (see task error codes)

### See Also

[\\_sched\\_set\\_rr\\_interval](#), [\\_sched\\_set\\_rr\\_interval\\_ticks](#)

[\\_task\\_set\\_error](#)

### Example

```
uint32_t time_slice;
...
result = _sched_get_rr_interval(_task_get_id(), &time_slice);
```

## 2.1.240 `_sched_set_policy`

Sets the scheduling policy.

### Prototype

```
source\kernel\sched.c
_mqx_uint _sched_set_policy(
    _task_id task_id,
    _mqx_uint policy)
```

### Parameters

*task\_id* [IN] — One of the following:

task on this processor for which to set info

MQX\_DEFAULT\_TASK\_ID (set the policy for the processor)

MQX\_NULL\_TASK\_ID (set the policy for the calling task)

*policy* [IN] — New scheduling policy; one of the following:

MQX\_SCHED\_FIFO

MQX\_SCHED\_RR

### Returns

- Previous scheduling policy (success)
- MAX\_MQX\_UINT (failure)

### Task Error Codes

- MQX\_SCHED\_INVALID\_POLICY — *policy* is not one of the allowed policies.
- MQX\_SCHED\_INVALID\_TASK\_ID — *task\_id* is not a valid task on this processor.

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_sched\\_get\\_policy](#)

[\\_task\\_set\\_error](#)

### Example

See `_sched_get_policy()`.

## 2.1.241 `_sched_set_rr_interval, _sched_set_rr_interval_ticks`

Set the time slice in:	
<code>_sched_set_rr_interval()</code>	Milliseconds
<code>_sched_set_rr_interval_ticks()</code>	Tick time

### Prototype

```
source\kernel\sched.c
uint32_t _sched_set_rr_interval(
    _task_id  task_id,
    uint32_t  ms_interval)

uint32_t _sched_set_rr_interval_ticks(
    _task_id      task_id,
    MQX_TICK_STRUCT_PTR new_rr_interval_ptr,
    MQX_TICK_STRUCT_PTR old_rr_interval_ptr)
```

### Parameters

*task\_id* [IN] — One of the following:

task ID for a task on this processor for which to set info

**MQX\_DEFAULT\_TASK\_ID** (set the time slice for the processor)

**MQX\_NULL\_TASK\_ID** (set the time slice for the calling task)

*ms\_interval* [IN] — New time slice (in milliseconds)

*new\_rr\_interval\_ptr* [IN] — Pointer to the new time slice (in tick time)

*old\_rr\_interval\_ptr* [OUT] — Pointer to the previous time slice (in tick time)

### Returns

- Previous time slice (success)
- MAX\_MQX\_UINT (failure)

### Traits

On failure, calls `_task_set_error()` to set the task error code to **MQX\_SCHED\_INVALID\_TASK\_ID**

### See Also

[\\_sched\\_get\\_rr\\_interval, \\_sched\\_get\\_rr\\_interval\\_ticks](#)

[\\_task\\_set\\_error](#)

### Example

Set the time slice to 50 milliseconds for the active task.

```
uint32_t result;
...
result = _sched_set_rr_interval(task_get_id(), 50);
```

## 2.1.242 **\_sched\_yield**

Puts the active task at the end of its ready queue.

### Prototype

```
source\kernel\sched.c  
void _sched_yield(void)
```

### Parameters

None

### Returns

None

### Traits

Might dispatch another task

### Description

The function effectively performs a timeslice. If there are no other tasks in this ready queue, the task continues to be the active task.

### Example

A task timeslices itself after a certain number of counts.

```
_mqx_uint counter = 0;  
...  
if (++counter == TIME_SLICE_COUNT) {  
    counter = 0;  
    _sched_yield();  
}
```



## 2.1.243 `_sem_close`

Closes the connection to the semaphore.

### Prototype

```
source\kernel\sem.c
#include <sem.h>
_mqx_uint _sem_close(
    void *sem_handle)
```

### Parameters

*sem\_handle* [IN] — Semaphore handle from `_sem_open()` or `_sem_open_fast()`

### Returns

- `MQX_OK`
- Errors

Error	Description
Error code from <code>_mem_free()</code>	Task is not the one that opened the connection.
<code>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</code>	Function cannot be called from an ISR.
<code>SEM_INVALID_SEMAPHORE_HANDLE</code>	<ul style="list-style-type: none"> <li>• <code>sem_handle</code> is not a valid semaphore connection</li> <li>• semaphore is no longer valid</li> </ul>

### Traits

- If the semaphore is strict, posts the appropriate number of times to the semaphore for this connection
- Might dispatch tasks that are waiting for the semaphore
- Cannot be called from an ISR

### See Also

[\\_sem\\_destroy](#), [\\_sem\\_destroy\\_fast](#)

[\\_sem\\_open](#), [\\_sem\\_open\\_fast](#)

### Example

See `_sem_open()`

## 2.1.244 `_sem_create`

Creates a named semaphore.

### Prototype

```
source\kernel\sem.c
#include <sem.h>
_mqx_uint _sem_create(
    char      *name,
    _mqx_uint  sem_count,
    _mqx_uint  flags)
```

### Parameters

*name* [IN] — Name by which to identify the semaphore

*sem\_count* [IN] — Number of requests that can concurrently have the semaphore

*flags* [IN] — Bit flags: 0 or as in description

### Returns

- MQX\_OK (success)
- Errors (failure)

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_COMPONENT_DOES_NOT_EXIST	Semaphore component was not created and cannot be created.
MQX_INVALID_COMPONENT_BASE	Semaphore component data is not valid.
MQX_OUT_OF_MEMORY	MQX cannot allocate memory for the semaphore.
SEM_INCORRECT_INITIAL_COUNT	<i>sem_count</i> cannot be 0 if SEM_STRICT is set.
SEM_INVALID_POLICY	SEM_STRICT must be set if SEM_PRIORITY_INHERITANCE is set.
SEM_SEMAPHORE_EXISTS	Semaphore with the name exists.
SEM_SEMAPHORE_TABLE_FULL	Semaphore names database is full and cannot be expanded.

### Traits

- Creates the semaphore component with default values if it were not previously created
- Cannot be called from an ISR
- On failure, calls `_task_set_error()` to set the task error code (see errors)

**See Also**[\\_sem\\_create\\_component](#)[\\_sem\\_destroy](#), [\\_sem\\_destroy\\_fast](#)[\\_sem\\_open](#), [\\_sem\\_open\\_fast](#)[\\_sem\\_close](#)[\\_task\\_set\\_error](#)**Description**

After the semaphore is created, tasks open a connection to it with [\\_sem\\_open\(\)](#) and close the connection with [\\_sem\\_close\(\)](#). A named semaphore is destroyed with [\\_sem\\_destroy\(\)](#).

Bit flag	Set	Not set
<b>SEM_PRIORITY_INHERITANCE</b> ( <b>SEM_STRICT</b> must also be set)	If a task that waits for the semaphore has a higher priority than a task that owns the semaphore, MQX RTOS boosts the priority of one of the owning tasks to the priority of the waiting task. When the boosted task posts its semaphore, MQX RTOS returns its priorities to its original values.	MQX RTOS does not boost priorities
<b>SEM_PRIORITY_QUEUEING</b>	Task that waits for the semaphore is queued according to the task's priority. Within a priority, tasks are in FIFO order.	Task that waits for the semaphore is queued in FIFO order
<b>SEM_STRICT</b>	<ul style="list-style-type: none"> <li>Task must wait for the semaphore before it can post the semaphore</li> </ul>	Task need not wait before posting
	<ul style="list-style-type: none"> <li><i>sem_count</i> must be greater than or equal to 1</li> </ul>	<i>sem_count</i> must be greater than or equal to 0

**Example**

See [\\_sem\\_create\\_component\(\)](#).

## 2.1.245 **\_sem\_create\_component**

Creates the semaphore component.

### Prototype

```
source\kernel\sem.c
#include <sem.h>
_mqx_uint _sem_create_component(
    _mqx_uint  initial_number,
    _mqx_uint  grow_number,
    _mqx_uint  maximum_number)
```

### Parameters

*initial\_number* [IN] — Initial number of semaphores that can be created

*grow\_number* [IN] — Number of semaphores to be added when the initial number have been created

*maximum\_number* [IN] — If *grow\_number* is not 0; one of:  
 maximum number of semaphores that can be created  
 0 (unlimited number)

### Returns

- MQX\_OK (success)
- MQX\_OUT\_OF\_MEMORY (failure: MQX RTOS cannot allocate memory for semaphore component data)

### Traits

On failure, the task error code might be set

### See Also

[\\_sem\\_create](#)

[\\_sem\\_create\\_fast](#)

[\\_sem\\_open, \\_sem\\_open\\_fast](#)

[\\_task\\_set\\_error](#)

### Description

If an application previously called the function and *maximum\_number* is greater than what was specified, MQX RTOS changes the maximum number of semaphores to *maximum\_number*.

If an application does not explicitly create the semaphore component, MQX RTOS does so with the following default values the first time that a task calls [\\_sem\\_create\(\)](#) or [\\_sem\\_create\\_fast\(\)](#).

Parameter	Default
initial_number	8
grow_number	8
maximum_number	0 (unlimited)

### Example

```

_mqx_uint result;
...
/* Create semaphore component: */
result = _sem_create_component(5, 5, 30);

if (result != MQX_OK) {
    /* An error occurred. */
}

/* Create a named semaphore of maximum count 1: */
result = _sem_create(".servo", 1, SEM_PRIORITY_QUEUEING);
if (result != MQX_OK) {
    /* An error occurred. */
}

/* Create a fast semaphore of maximum count 3: */
result = _sem_create_fast(SEM_DODAD, 3, SEM_PRIORITY_QUEUEING);
if (result != MQX_OK) {
    /* An error occurred. */
}

/* Use the semaphores. */

/* Destroy both semaphores: */
result = _sem_destroy("servo", TRUE);
if (result != MQX_OK) {
    /* An error occurred. */
}

result = _sem_destroy_fast(SEM_DODAD, TRUE);
if (result != MQX_OK) {
    /* An error occurred. */
}

```

## 2.1.246 **\_sem\_create\_fast**

Creates the fast semaphore.

### Prototype

```
source\kernel\sem.c
#include <sem.h>
_mqx_uint _sem_create_fast(
    _mqx_uint  sem_index,
    _mqx_uint  initial_count,
    _mqx_uint  flags)
```

### Parameters

*sem\_index [IN]* — Number by which to identify the semaphore  
*initial\_count [IN]* — Number of tasks that can concurrently have the semaphore  
*flags [IN]* — Bit flags, as described for **\_sem\_create()**

### Returns

- MQX\_OK
- Error, as described for **\_sem\_create()**

### Traits

- Creates the semaphore component with default values if it was not previously created
- Cannot be called from an ISR
- On error, the task error code might be set

### See Also

[\\_sem\\_create\\_component](#)

[\\_sem\\_destroy, \\_sem\\_destroy\\_fast](#)

[\\_sem\\_open, \\_sem\\_open\\_fast](#)

[\\_sem\\_close](#)

[\\_sem\\_create](#)

### Description

After the semaphore is created, tasks open a connection to it with **\_sem\_open\_fast()** and close the connection with **\_sem\_close()**. A fast semaphore is destroyed with **\_sem\_destroy\_fast()**.

### Example

See [\\_sem\\_create\\_component\(\)](#).

## 2.1.247 `_sem_destroy`, `_sem_destroy_fast`

`_sem_destroy()` Destroys the named semaphore.  
`_sem_destroy_fast()` Destroys the fast semaphore.

### Prototype

```
source\kernel\sem.c
#include <sem.h>
_mqx_uint _sem_destroy(
    char      *name,
    bool      force_destroy)

_mqx_uint _sem_destroy_fast(
    _mx_uint  index,
    bool      force_destroy)
```

### Parameters

*name* [IN] — Name of the semaphore to destroy, created using `_sem_create()`  
*force\_destroy* [IN] — See description  
*index* [IN] — Number that identifies the semaphore to destroy, created using `_sem_create_fast()`

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_COMPONENT_DOES_NOT_EXIST	Semaphore component is not created.
MQX_INVALID_COMPONENT_BASE	Semaphore component data is not valid.
SEM_INVALID_SEMAPHORE	Semaphore data that is associated with <i>name</i> or <i>index</i> is not valid.
SEM_SEMAPHORE_NOT_FOUND	<i>name</i> or <i>index</i> is not in the semaphore names database.

### Traits

Cannot be called from an ISR

### See Also

[\\_sem\\_close](#)

[\\_sem\\_create](#)

[\\_sem\\_create\\_fast](#)

## Description

<i>force_destroy</i> is <i>TRUE</i>	<i>force_destroy</i> is <i>FALSE</i>
<ul style="list-style-type: none"><li>• Tasks that are waiting for the semaphore are readied.</li><li>• Semaphore is destroyed after all the owners post the semaphore.</li></ul>	<ul style="list-style-type: none"><li>• Semaphore is destroyed after the last waiting task gets and posts the semaphore.</li><li>• This is the action if the semaphore is strict.</li></ul>

## Example

See `_sem_create_component()`.



## 2.1.248 `_sem_get_value`

Gets the value of the semaphore counter; that is, the number of subsequent requests that can get the semaphore without waiting.

### Prototype

```
source\kernel\sem.c
#include <sem.h>
_mqx_uint _sem_get_value(
    void *users_sem_handle)
```

### Parameters

*users\_sem\_handle [IN]* — Semaphore handle from `_sem_open()` or `_sem_open_fast()`

### Returns

- Current value of the semaphore counter (success)
- MAX\_MQX\_UINT (failure)

### Task Error Codes

Task Error Code	Description
SEM_INVALID_SEMAPHORE	<i>sem_ptr</i> does not point to a valid semaphore.
SEM_INVALID_SEMAPHORE_HANDLE	<i>sem_ptr</i> is not a valid semaphore handle.

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_sem\\_open](#), [\\_sem\\_open\\_fast](#)

[\\_sem\\_post](#)

[\\_sem\\_get\\_wait\\_count](#)

[\\_sem\\_wait ...](#)

[\\_task\\_set\\_error](#)

## 2.1.249 **`_sem_get_wait_count`**

Gets the number of tasks that are waiting for the semaphore.

### Prototype

```
source\kernel\sem.c
#include <sem.h>
_mqx_uint _sem_get_wait_count(
    void *sem_handle)
```

### Parameters

*sem\_handle* [IN] — Semaphore handle from **`_sem_open()`** or **`_sem_open_fast()`**

### Returns

- Number of tasks waiting for the semaphore (success)
- MAX\_MQX\_UINT (failure)

### Traits

On failure, calls `_task_set_error()` to set the task error code as for `_sem_get_value()`

### See Also

[`\_sem\_open`](#), [`\_sem\_open\_fast`](#)

[`\_sem\_post`](#)

[`\_sem\_get\_value`](#)

[`\_sem\_wait ...`](#)

[`\_task\_set\_error`](#)

## 2.1.250 `_sem_open`, `_sem_open_fast`

`_sem_open()` Opens a connection to the named semaphore.  
`_sem_open_fast()` Opens a connection to the fast semaphore.

### Prototype

```
source\kernel\sem.c
#include <sem.h>
_mqx_uint _sem_open(
    char *name,
    void *sem_handle)

_mqx_uint _sem_open_fast(
    _mqx_uint index,
    void *sem_handle)
```

### Parameters

*name* [IN] — Name that identifies the semaphore that was created using `_sem_create()`  
*sem\_handle* [OUT] — Pointer to the semaphore handle, which is a connection to the semaphore  
*index* [IN] — Number that identifies the semaphore that was created using `_sem_create_fast()`

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_COMPONENT_DOES_NOT_EXIST	Semaphore component is not created.
MQX_INVALID_COMPONENT_BASE	Semaphore component data is not valid.
MQX_OUT_OF_MEMORY	MQX RTOS cannot allocate memory for the connection.
SEM_INVALID_SEMAPHORE	Data that is associated with the semaphore is not valid.
SEM_SEMAPHORE_DELETED	Semaphore is in the process of being destroyed.
SEM_SEMAPHORE_NOT_FOUND	<i>name</i> is not in the semaphore names database.

### See also

[\\_sem\\_close](#)

[\\_sem\\_create](#)

[\\_sem\\_post](#)

[\\_sem\\_wait ...](#)

### Example

TaskA(void)

```

{
    void *sem_handle;
    _mqx_uint result;

    /* Create a semaphore of maximum count 1: */
    result = _sem_create("phaser", 1, SEM_PRIORITY_QUEUEING);
    if (result == MQX_OK) {
        result = _sem_open("three", &sem_handle);
    }

    while (result != MQX_OK) {
        /* Wait for the semaphore: */
        result = _sem_wait(sem_handle, timeout);
        if (result == MQX_OK) {
            /* Perform work. */
            result = _sem_post(sem_handle);
        }
    }
    /* An error occurred. */
    _sem_close(sem_handle);
}

TaskB(void)
{
    void *sem_handle;
    _mqx_uint result;

    result = _sem_open("three", &sem_handle);
    while (result != MQX_OK) {
        /* Wait for the semaphore: */
        result = _sem_wait(sem_handle, timeout);
        if (result == MQX_OK) {
            /* Perform other work. */
            result = _sem_post(sem_handle);
        }
    }
    /* An error occurred. */
    _sem_close(sem_handle);
}

```

## 2.1.251 `_sem_post`

Posts the semaphore.

### Prototype

```
source\kernel\sem.c
#include <sem.h>
_mqx_uint _sem_post(
    void *sem_handle)
```

### Parameters

*sem\_handle* [IN] — Semaphore handle from `_sem_open()` or `_sem_open_fast()`

### Returns

- `MQX_OK`
- Errors

Error	Description
<code>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</code>	Function cannot be called from an ISR.
<code>SEM_CANNOT_POST</code>	Semaphore is strict and the task has not first waited for the semaphore.
<code>SEM_INVALID_SEMAPHORE</code>	<i>sem_handle</i> represents a semaphore that is no longer valid.
<code>SEM_INVALID_SEMAPHORE_COUNT</code>	Semaphore data is corrupted.
<code>SEM_INVALID_SEMAPHORE_HANDLE</code>	One of the following: <ul style="list-style-type: none"> <li>• <i>sem_handle</i> is not a valid semaphore handle</li> <li>• semaphore is strict and <i>sem_handle</i> was obtained by another task</li> </ul>

### Traits

- Might put a task in its ready queue
- For a strict semaphore, cannot be called from an ISR (ISR can call the function for a non-strict semaphore)

### See Also

[\\_sem\\_open, \\_sem\\_open\\_fast](#)

[\\_sem\\_get\\_wait\\_count](#)

[\\_sem\\_get\\_value](#)

[\\_sem\\_wait ...](#)

### Description

MQX RTOS gives the semaphore to the first waiting task and puts the task in the task's ready queue.

## Example

See [\\_sem\\_open](#), [\\_sem\\_open\\_fast](#).

## 2.1.252 `_sem_test`

Tests the semaphore component.

### Prototype

```
source\kernel\sem.c
#include <sem.h>
_mqx_uint _sem_test(
    void *sem_error_ptr)
```

### Parameters

*sem\_error\_ptr* [OUT] — Pointer to the semaphore that has an error (*NULL* if no errors are found)

### Returns

- MQX\_OK (no errors are found)
- See errors (an error is found)

Error	MQX RTOS found an error in:
MQX_CORRUPT_QUEUE	A semaphore queue
MQX_INVALID_COMPONENT_BASE	Semaphore component data
SEM_INVALID_SEMAPHORE	Semaphore data

### Traits

Disables and enables interrupts

### See Also

[\\_sem\\_close](#)

[\\_sem\\_create](#)

[\\_sem\\_create\\_fast](#)

[\\_sem\\_open, \\_sem\\_open\\_fast](#)

[\\_sem\\_post](#)

[\\_sem\\_wait ...](#)

### Description

The function does the following:

- verifies semaphore component data
- verifies the integrity of the entries in the semaphore names database
- for each semaphore, checks:
  - validity of data (**VALID** field)
  - integrity of the queue of waiting tasks
  - integrity of the queue of tasks that have the semaphore

## 2.1.253 `_sem_wait ...`

Wait for the semaphore:	
<code>_sem_wait()</code>	For the number of milliseconds
<code>_sem_wait_for()</code>	For the number of ticks (in tick time)
<code>_sem_wait_ticks()</code>	For the number of ticks
<code>_sem_wait_until()</code>	Until the specified time (in tick time)

### Prototype

```
source\kernel\sem.c
#include <sem.h>
_mqx_uint _sem_wait(
    void      *sem_handle,
    uint32_t  ms_timeout)

_mqx_uint _sem_wait_for(
    void      *sem_handle,
    MQX_TICK_STRUCT_PTR  tick_time_timeout_ptr)

_mqx_uint _sem_wait_ticks(
    void      *sem_handle,
    _mx_uint  tick_timeout)

_mqx_uint _sem_wait_until(
    void      *sem_handle,
    MQX_TICK_STRUCT_PTR  tick_time_ptr)
```

### Parameters

*sem\_handle* [IN] — Semaphore handle from `_sem_open()` or `_sem_open_fast()`

*ms\_timeout* [IN] — One of the following:

maximum number of milliseconds to wait for the semaphore. After the timeout elapses without the semaphore signalled, the function returns.

0 (unlimited wait)

*tick\_time\_timeout\_ptr* [IN] — One of the following:

pointer to the maximum number of ticks to wait

NULL (unlimited wait)

*tick\_timeout* [IN] — One of the following:

maximum number of ticks to wait

0 (unlimited wait)

*tick\_time\_ptr* [IN] — One of the following:

pointer to the time (in tick time) until which to wait

NULL (unlimited wait)

### Returns



- MQX\_OK
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_EDEADLK	Function was aborted to prevent deadlock: the task has all the semaphore locks and, since the semaphore is strict, the task cannot post to “wake” itself.
SEM_INVALID_SEMAPHORE	<i>sem_handle</i> is for a semaphore that is no longer valid.
SEM_INVALID_SEMAPHORE_HANDLE	One of the following: <ul style="list-style-type: none"> <li>• <i>sem_handle</i> is not a valid semaphore handle</li> <li>• <i>sem_handle</i> was obtained by another task</li> </ul>
SEM_SEMAPHORE_DELETED	MQX RTOS is in the process of destroying the semaphore.
SEM_WAIT_TIMEOUT	Timeout expired before the task can get the semaphore.

### Traits

- Might block the calling task
- Cannot be called from an ISR

### See Also

[\\_sem\\_open](#), [\\_sem\\_open\\_fast](#)

[\\_sem\\_post](#)

[\\_sem\\_get\\_wait\\_count](#)

[\\_sem\\_get\\_value](#)

[\\_sem\\_create](#)

[\\_sem\\_create\\_fast](#)

### MQX\_TICK\_STRUCT

### Description

If the task cannot get the semaphore, MQX RTOS queues the task according to the semaphore’s queuing policy, which is set when the semaphore is created.

### Example

See [\\_sem\\_open](#), [\\_sem\\_open\\_fast](#).

## 2.1.254 `_str_mqx_uint_to_hex_string`

Converts the `_mqx_uint` value to a hexadecimal string.

### Prototype

```
source\string\str_utos.c  
void _str_mqx_uint_to_hex_string(  
    _mqx_uint  number  
    char       *string_ptr)
```

### Parameters

*number* [IN] — Number to convert

*string\_ptr* [OUT] — Pointer to the hexadecimal string equivalent of number

### Returns

None

### See Also

[\\_strlen](#)

## 2.1.255 `_strnlen`

Gets the length of the length-limited string.

### Prototype

```
source\string\strnlen.c
_mqx_uint _strnlen(
    char      *string_ptr
    _mx_uint  max_length)
```

### Parameters

*string\_ptr* [IN] — Pointer to the string

*max\_length* [OUT] — Maximum number characters in the string

### Returns

Number of characters in the string

### See Also

[\\_str\\_mqx\\_uint\\_to\\_hex\\_string](#)

## 2.1.256 `_task_abort`

Makes a task run its task exit handler and then destroys itself.

### Prototype

```
source\kernel\task.c
_mqx_uint _task_abort(
    _task_id task_id)
```

### Parameters

*task\_id* [IN] — One of the following:

task ID of the task to be destroyed

**MQX\_NULL\_TASK\_ID** (abort the calling task)

### Description:

This function can be dangerous if used incorrectly. Use it only to abort tasks which do not call the MQX RTOS synchronization API (mutex, semaphore, message etc.). This warning applies when a task is not calling this API to terminate itself, but to terminate another task.

### Returns

- MQX\_OK (success)
- MQX\_INVALID\_TASK\_ID (failure: task\_id does not represent a valid task)

### See Also

[\\_task\\_destroy](#)

[\\_task\\_get\\_exit\\_handler](#), [\\_task\\_set\\_exit\\_handler](#)

### Example

Task B creates Task A and later aborts it.

```
#include <mqx.h>

void Exit_Handler(void)
{
    printf("Task %x has aborted\n", _task_get_id());
}

void TaskA(uint32_t param)
{
    _task_set_exit_handler(_task_get_id(), Exit_Handler);
    while (TRUE) {
        ...
        _sched_yield();
    }
}

void TaskB(uint32_t param)
```

```
{
    _task_id  taska_id;

    taska_id = _task_create(0, TASKA, 0);
    ...

    _task_abort(taska_id);
}
```

## 2.1.257 `_task_block`

Blocks the active task.

### Prototype

```
source\psp\<<core_family>\core\<<core>\dispatch.S  
void _task_block(void)
```

### Parameters

None

### Returns

None

### Traits

Dispatches another task

### See also

[\\_task\\_ready](#)

[\\_task\\_restart](#)

### Description

The function removes the active task from the task's ready queue and sets the **BLOCKED** bit in the **STATE** field of the task descriptor.

The task does not run again until another task explicitly makes it ready with `_task_ready()`.

### Example

See `_task_ready()`.

## 2.1.258 `_task_check_stack`

Determines whether the stack for the active task is currently out of bounds.

### Prototype

```
source\kernel\task.c  
bool _task_check_stack(void)
```

### Parameters

None

### Returns

- TRUE (stack is out of bounds)
- FALSE (stack is not out of bounds)

### See Also

[\\_task\\_set\\_error](#)

### Description

The function indicates whether the stack is currently past its limit. The function does not indicate whether the stack previously passed its limit.

## 2.1.259 `_task_create`, `_task_create_blocked`, `_task_create_at`, `create_task`

<code>_task_create()</code>	Creates the task and make it ready
<code>_task_create_blocked()</code>	Creates the task, but do not make it ready
<code>_task_create_at()</code>	Creates the task with the stack location specified
<code>create_task()</code>	Creates the task in the universal way

### Prototype

```
source\kernel\task.c
_task_id _task_create(
    _processor_number    processor_number,
    _mqx_uint            template_index,
    uint32_t             parameter)

_task_id _task_create_blocked(
    _processor_number    processor_number,
    _mqx_uint            template_index,
    uint32_t             parameter)

_task_id _task_create_at(
    _processor_number    processor_number,
    _mqx_uint            template_index,
    uint32_t             parameter,
    void                 *stack_ptr,
    _mem_size            stack_size)

task_id_create_task (const taskinit_t * task_description_structure)
```

### Parameters

*processor\_number* [IN] — One of the following:

- processor number of the processor where the task is to be created
- 0 (create on the local processor)

*template\_index* [IN] — One of the following:

- index of the task template in the processor's task template list to use for the child task
- 0 (use the task template that *create\_parameter* defines)

*parameter* [IN]

- template\_index* is not 0 — pointer to the parameter that MQX RTOS passes to the child task
- template\_index* is 0 — pointer to the task template

*stack\_ptr* [IN] — The location where the stack and TD are to be created.

*stack\_size* [IN] — The size of the stack.

*task\_description\_structure*[IN] - The structure describing the task initialization.

### Returns

- Task ID of the child task (success)
- MQX\_NULL\_TASK\_ID (failure)



## Task Error Codes

Task Error Code	Description
MQX_INVALID_PROCESSOR_NUMBER	<i>processor_number</i> is not one of the allowed processor numbers.
MQX_NO_TASK_TEMPLATE	<i>template_index</i> is not in the task template list.
MQX_OUT_OF_MEMORY	MQX RTOS cannot allocate memory for the task data structures.

## Traits

- If the child is on another processor, blocks the creator until the child is created
- On failure, calls `_task_set_error()` to set the task error code (see task error codes)
- For `_task_create()`:
  - If the child is on the same processor, preempts the creator if the child is a higher priority

## See Also

[\\_task\\_abort](#)

[\\_task\\_block](#)

[\\_task\\_destroy](#)

[\\_task\\_get\\_parameter ..., \\_task\\_set\\_parameter ...](#)

[\\_task\\_ready](#)

[\\_task\\_set\\_error](#)

[MQX\\_INITIALIZATION\\_STRUCT](#)  
[TASK\\_TEMPLATE\\_STRUCT](#)

## Example

Create an instance of Receiver task.

```
#define RECEIVER_TEMPLATE (0x100)

result = _task_create(0, RECEIVER_TEMPLATE, 0);

if (result == MQX_NULL_TASK_ID) {
    printf("\nCould not create receiver task.");
} else {
    /* Task with a task ID equal to result was created */
    ...
}
```

## 2.1.260 `_task_destroy`

Destroys the task.

### Prototype

```
source\kernel\task.c
_mqx_uint _task_destroy(
    _task_id task_id)
```

### Parameters

*task\_id* [IN] — One of the following:

task ID of the task to be destroyed

**MQX\_NULL\_TASK\_ID** (destroy the calling task)

### Returns

- MQX\_OK
- MQX\_INVALID\_TASK\_ID

### Traits

- If the task being destroyed is remote, blocks the calling task until the task is destroyed
- If the task being destroyed is local, does not block the calling task
- If the task being destroyed is the active task, blocks it

### See Also

[\\_task\\_create](#), [\\_task\\_create\\_blocked](#), [\\_task\\_create\\_at](#), [create\\_task](#)

[\\_task\\_get\\_creator](#)

[\\_task\\_get\\_id](#)

[\\_task\\_abort](#)

### Description

This function can be dangerous if used incorrectly. Use it only to abort tasks which do not call MQX RTOS synchronization API (mutex, semaphore, message etc.). This warning applies when a task is not calling this API to terminate itself, but to terminate another task. The function does the following for the task being destroyed:

- frees memory resources that the task allocated with functions from the **\_mem** and **\_partition** families
- closes all queues that the task owns and frees all the queue elements
- frees any other component resources that the task owns

### Example

If the second task cannot be created, destroy the first task.

```
_task_id first_born;
_task_id second_born;
```

```
first_born = _task_create(PROCESSOR_ONE, FIRST, CHANNEL_1);
if (first_born == 0) {
    ...
} else if ((second_born = _task_create(PROCESSOR_TWO, SECOND,
    BACKUP_CHANNEL)) == 0) {
    _task_destroy(first_born);
} else {
    ...
}
```

## 2.1.261 `_task_disable_fp`, `_task_enable_fp`

`_task_disable_fp()` Disables floating-point context switching for the active task if the task is a floating-point task.  
`_task_enable_fp()` Enables floating-point context switching for the active task.

### Prototype

```
source\kernel\task.c
void _task_disable_fp(void)

void _task_enable_fp(void)
```

### Traits

Changes context information that MQX RTOS stores

### Description

Function	When MQX RTOS performs a context switch, floating-point registers are saved and restored?
<code>_task_disable_fp()</code>	No
<code>_task_enable_fp()</code>	Yes

### Example

Task is about to do some floating-point work, so change the type of context switch.

```
_task_enable_fp();
/* Start floating-point math. */
...
/* Floating-point math is complete. */
_task_disable_fp();
```

## 2.1.262 `_task_errno`

Gets the task error code for the active task.

### Prototype

```
source\include\mqx.h  
#define _task_errno  (*_task_get_error_ptr())
```

### See Also

[\\_task\\_get\\_error](#), [\\_task\\_get\\_error\\_ptr](#)

[\\_task\\_set\\_error](#)

### Description

MQX RTOS provides the variable for POSIX compatibility.

`_task_errno` gives the same value as `_task_get_error()`.

### Example

Print the task error code of the active task.

```
void *event_ptr;  
_mqx_uint task_wait_count;  
...  
if (_event_open("global", &event_ptr) == MQX_OK) {  
    ...  
    if (_event_get_wait_count(event_ptr) == MAX_MQX_UINT) {  
        printf("\nTask error code is 0x%lx", _task_errno);  
    }  
}
```

## 2.1.263 `_task_get_creator`

Gets the task ID of the task that created the calling task.

### Prototype

```
source\kernel\task.c  
_task_id _task_get_creator(void)
```

### Parameters

None

### Returns

Task ID of the parent task

### See Also

[\\_task\\_get\\_processor](#)

[\\_task\\_get\\_id](#)

## 2.1.264 `_task_get_environment`, `_task_set_environment`

<code>_task_get_environment()</code>	Gets a pointer to the application-specific environment data for the task.
<code>_task_set_environment()</code>	Sets the address of the application-specific environment data for the task.

### Prototype

```
source\kernel\task.c
void *_task_get_environment(
    _task_id task_id)

void *_task_set_environment(
    _task_id task_id,
    void      *environment_ptr)
```

### Parameters

*task\_id* [IN] — Task ID of the task whose environment data is to be set or obtained  
*environment\_ptr* [IN] — Pointer to the environment data

### Returns

- (Get) Environment data (success)
- (Set) Previous environment data (success)
- NULL (failure)

### Traits

On failure, calls `_task_set_error()` to set the task error code to `MQX_INVALID_TASK_ID`

### See Also

[\\_task\\_get\\_parameter ..., \\_task\\_set\\_parameter ...](#)  
[\\_task\\_set\\_error](#)

### Example

Check the environment data for the active task.

```
if (_task_get_environment(_task_get_id())) {
    /* Environment data has been set; don't reset it. */
} else {
    _task_set_environment(_task_get_id(), context_ptr);
}
```

## 2.1.265 `_task_get_error`, `_task_get_error_ptr`

`_task_get_error()` Gets the task error code  
`_task_get_error_ptr()` Gets a pointer to the task error code.

### Prototype

```
source\kernel\task.c
_mqx_uint  _task_get_error(void)

_mqx_uint *_task_get_error_ptr(void)
```

### Parameters

None

### Returns

- `_task_get_error()` — Task error code for the active task
- `_task_get_error_ptr()` — Pointer to the task error code

### See Also

[\\_task\\_set\\_error](#)

[\\_task\\_errno](#)

### Description

#### CAUTION

If a task writes to the pointer that `_task_get_error_ptr()` returns, the task error code is changed to the value, overwriting any previous error code. To avoid overwriting a previous error code, a task should use `_task_set_error()`.

### Example

Get the task error code and reset it if required.

```
if (_task_get_error() == MSGQ_QUEUE_FULL){
    _task_set_error(MQX_OK);
}
```



## 2.1.266 `_task_get_exception_handler`, `_task_set_exception_handler`

`_task_get_exception_handler()` Gets a pointer to the task exception handler.  
`_task_set_exception_handler()` Sets the address of the task exception handler.

### Prototype

```
source\kernel\task.c
TASK_EXCEPTION_FPTR _task_get_exception_handler(
    _task_id task_id)

TASK_EXCEPTION_FPTR _task_set_exception_handler(
    _task_id task_id,
    TASK_EXCEPTION_FPTR handler_address)
```

### Parameters

*task\_id* [IN] — Task ID of the task whose exception handler is to be set or obtained  
*handler\_address* [IN] — Pointer to the task exception handler

### Returns

- `_task_get_exception_handler()` — Pointer to the task exception handler for the task (might be NULL) (success)
- `_task_set_exception_handler()` — Pointer to the previous task exception handler (might be NULL) (success)
- NULL (failure: *task\_id* is not valid)

### Traits

On failure, calls `_task_set_error()` to set the task error code to **MQX\_INVALID\_TASK\_ID**

### See also

[\\_task\\_get\\_exit\\_handler](#), [\\_task\\_set\\_exit\\_handler](#)

[\\_int\\_exception\\_isr](#)

[\\_task\\_set\\_error](#)

## 2.1.267 `_task_get_exit_handler`, `_task_set_exit_handler`

`_task_get_exit_handler()` Gets a pointer to the task exit handler for the task.  
`_task_set_exit_handler()` Sets the address of the task exit handler for the task.

### Prototype

```
source\kernel\task.c
TASK_EXIT_FPTR _task_get_exit_handler(
    _task_id task_id) (void)

TASK_EXIT_FPTR _task_set_exit_handler(
    _task_id task_id,
    TASK_EXIT_FPTR exit_handler_address)
```

### Parameters

*task\_id* [IN] — Task ID of the task whose exit handler is to be set or obtained  
*exit\_handler\_address* [IN] — Pointer to the exit handler for the task

### Returns

- `_task_get_exit_handler()` — Pointer to the exit handler (might be NULL) (success)
- `_task_set_exit_handler()` — Pointer to the previous exit handler (might be NULL) (success)
- NULL (failure: *task\_id* is not valid)

### Traits

On failure, calls `_task_set_error()` to set the task error code to **MQX\_INVALID\_TASK\_ID**

### See Also

[\\_mqx\\_exit](#)  
[\\_task\\_get\\_exception\\_handler](#), [\\_task\\_set\\_exception\\_handler](#)  
[\\_task\\_abort](#)  
[\\_task\\_set\\_error](#)

### Description

MQX RTOS calls a task's task exit handler if either of these conditions is true:

- task is terminated with `_task_abort()`
- task returns from its function body (for example, if it calls `_mqx_exit()`)

### Example

See `_task_abort()`.

## 2.1.268 `_task_get_id`

Gets the task ID of the active task.

### Prototype

```
source\kernel\task.c  
_task_id _task_get_id(void)
```

### Returns

Task ID of the active task

### See also

[\\_task\\_get\\_creator](#)

[\\_task\\_get\\_processor](#)

[\\_task\\_get\\_id\\_from\\_name](#)

### Example

See `_task_ready()`.

## 2.1.269 `_task_get_id_from_name`

Gets the task ID that is associated with the task name.

### Prototype

```
source\kernel\task.c
_task_id _task_get_id_from_name(
    char *name_ptr)
```

### Parameters

*name\_ptr* [IN] — Pointer to the name to find in the task template list

### Returns

- Task ID that is associated with the first match of *name\_ptr* (success)
- MQX\_NULL\_TASK\_ID (failure: name is not in the task template list)

### See Also

[\\_task\\_get\\_creator](#)

[\\_task\\_get\\_processor](#)

[\\_task\\_get\\_id](#)

[TASK\\_TEMPLATE\\_STRUCT](#)

### Example

Check whether a particular task has been created and, if it has not, create it.

```
task_id = _task_get_id_from_name("TestTask");
if (task_id == MQX_NULL_TASK_ID) {
    /* Create the task: */
    _task_create(0, _task_get_template_index("TestTask"), 0);
}
```

## 2.1.270 `_task_get_index_from_id`

Gets the task template index for the task ID.

### Prototype

```
source\kernel\task.c  
_mqx_uint _task_get_index_from_id(  
    _task_id task_id)
```

### Parameters

*task\_id* [IN] — Value to set the task parameter to

### Returns

- task template index (success)
- 0 (failure: task ID was not found)

### See Also

[\\_task\\_get\\_template\\_index](#)

## 2.1.271 `_task_get_parameter` ..., `_task_set_parameter` ...

<code>_task_get_parameter()</code>	Gets the task creation parameter of the active task.
<code>_task_get_parameter_for()</code>	Gets the task creation parameter of the specified task
<code>_task_set_parameter()</code>	Sets the task creation parameter of the active task.
<code>_task_set_parameter_for()</code>	Sets the task creation parameter of the specified task.

### Prototype

```
source\kernel\task.c
uint32_t _task_get_parameter(void)

uint32_t _task_get_parameter_for(
    task_id  task_id)

uint32_t _task_set_parameter(
    uint32_t new_value)

uint32_t _task_set_parameter_for(
    uint32_t new_value,
    task_id  task_id)
```

### Parameters

*new\_value* [IN] — Value to set the task parameter to  
*task\_id* [IN] — Task ID of the task to get or set

### Returns

- `_task_get_parameter()`, `_task_get_parameter_for()` — Creation parameter (might be NULL)
- `_task_set_parameter()`, `_task_set_parameter_for()` — Previous creation parameter (might be NULL)

### See Also

[\\_task\\_create](#), [\\_task\\_create\\_blocked](#), [\\_task\\_create\\_at](#), [create\\_task](#)

### Description

If a deeply nested function needs the task creation parameter, it can get the parameter with `_task_get_parameter()` or `_task_get_parameter_for()` rather than have the task's main body pass the parameter to it.

## 2.1.272 `_task_get_priority`, `_task_set_priority`

`_task_get_priority()` Gets the priority of the task.  
`_task_set_priority()` Sets the priority of the task.

### Prototype

```
source\kernel\task.c
_mqx_uint _task_get_priority(
    _task_id      task_id,
    _mqx_uint      *priority_ptr)

_mqx_uint _task_set_priority(
    _task_id      task_id,
    _mqx_uint      new_priority,
    _mqx_uint      *old_priority_ptr)
```

### Parameters

*task\_id* [IN] — One of the following:  
 task ID of the task for which to set or get info  
**MQX\_NULL\_TASK\_ID** (use the calling task)  
*priority\_ptr* [OUT] — Pointer to the priority  
*new\_priority* [IN] — New priority  
*old\_priority\_ptr* [OUT] — Pointer to the previous priority

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_INVALID_PARAMETER	<i>new_priority</i> is numerically greater than the lowest-allowable priority of an application task. Valid just for <b>_task_set_priority()</b> function.
MQX_INVALID_TASK_ID	<i>task_id</i> does not represent a currently valid task.

### Traits

Might dispatch a task

### See Also

[\\_task\\_get\\_creator](#)  
[\\_task\\_get\\_processor](#)  
[\\_sem\\_create](#)

[\\_sem\\_create\\_fast](#)[\\_sem\\_wait ...](#)[\\_mutatr\\_get\\_sched\\_protocol, \\_mutatr\\_set\\_sched\\_protocol](#)[\\_mutex\\_lock](#)

## Description

MQX RTOS might boost the priority of a task that waits for a semaphore or locks a mutex. If MQX RTOS has boosted the priority of the task that is specified by *task\_id*, **\_task\_set\_priority()** raises but not lower the task's priority.

If the task is in this state:	Priority change takes place:
Blocked	When task is ready
Ready	Immediately

## Example

Raise the priority of the current task.

```

_task_get_priority(_task_get_id(), &priority);
if (priority > 0) {
    priority--;
    if (_task_set_priority(_task_get_id(), priority, &temp) = MQX_OK)
        ...
}
```



## 2.1.273 `_task_get_processor`

Gets the processor number of the task's home processor.

### Prototype

```
source\kernel\task.c
_processor_number _task_get_processor(
    _task_id task_id)
```

### Parameters

*task\_id* [IN] — Task ID of the task for which to get info

### Returns

Processor number of the processor where the task resides

### See Also

[\\_task\\_get\\_id](#)

[MQX\\_INITIALIZATION\\_STRUCT](#)

### Description

The function returns the processor-number portion of *task\_id*. It cannot check the validity of *task\_id* because MQX RTOS on one processor is unaware of which tasks might reside on another processor.

### Example

Determine whether two tasks are on the same processor.

```
_task_id task_a;
_task_id task_b;

if (_task_get_processor(task_a) == _task_get_processor(task_b)) {
    /* Proceed */
    ...
}
```

## 2.1.274 `_task_get_td`

Gets a pointer to the task descriptor for the task ID.

### Prototype

```
source\kernel\task.c  
void *_task_get_td(  
    _task_id task_id)
```

### Parameters

*task\_id* [IN] — One of:  
task ID for a task on this processor  
**MQX\_NULL\_TASK\_ID** (use the current task)

### Returns

- Pointer to the task descriptor for *task\_id* (success)
- NULL (failure: *task\_id* is not valid for this processor)

### See also

[\\_task\\_ready](#)

### Example

See `_task_ready()`.

## 2.1.275 `_task_get_template_index`

Gets the task template index that is associated with the task name.

### Prototype

```
source\kernel\task.c
_mqx_uint _task_get_template_index(
    char *name_ptr)
```

### Parameters

*name\_ptr* [IN] — Pointer to the name to find in the task template list

### Returns

- Task template index that is associated with the first match of *name\_ptr* (success)
- MQX\_NULL\_TASK\_ID (failure: name is not in the task template list)

### See Also

[\\_task\\_get\\_id\\_from\\_name](#)

[\\_task\\_get\\_index\\_from\\_id](#)

[TASK\\_TEMPLATE\\_STRUCT](#)

### Example

See `_task_get_id_from_name()`.

## 2.1.276 `_task_get_template_ptr`

Gets the pointer to the task template for the task ID.

### Prototype

```
source\kernel\task.c  
TASK_TEMPLATE_STRUCT_PTR _task_get_template_ptr(  
    _task_id task_id)
```

### Parameters

*task\_id* [IN] — Task ID for the task for which to get info

### Returns

Pointer to the task's task template. NULL if an invalid task\_id is presented.

### See Also

[\\_task\\_get\\_template\\_index](#)

[\\_task\\_get\\_index\\_from\\_id](#)

## 2.1.277 `_task_ready`

Makes the task ready to run by putting it in its ready queue.

### Prototype

```
source\kernel\task.c
void _task_ready(
    void *td_ptr)
```

### Parameters

*td\_ptr* [IN] — Pointer to the task descriptor of the task (on this processor) to be made ready

### Task error codes

Task Error Code	Description
MQX_INVALID_TASK_ID	<i>task_id</i> is not valid for this processor.
MQX_INVALID_TASK_STATE	Task is already in its ready queue.

### Traits

- If the newly readied task is higher priority than the calling task, MQX RTOS makes the newly readied task active
- Might set the task error code (see task error codes)

### See Also

[\\_task\\_block](#)

[\\_time\\_dequeue](#)

[\\_taskq\\_resume](#)

### Description

The function is the only way to make ready a task that called `_task_block()`.

### Example

The following two functions implement a fast, cooperative scheduling mechanism, which takes the place of task queues.

```
#include mqx_prv.h

#define WAIT_BLOCKED 0xF1

Restart(_task_id tid) {
    TD_STRUCT_PTR td_ptr = _task_get_td(tid);
    _int_disable();
    if ((td_ptr != NULL) && (td_ptr->STATE == WAIT_BLOCKED)) {
        _task_ready(td_ptr);
    }
    _int_enable();
}
```

```
Wait() {  
    TD_STRUCT_PTR td_ptr = _task_get_td(_task_get_id());  
  
    _int_disable();  
    td_ptr->STATE = WAIT_BLOCKED;  
    _task_block();  
    _int_enable();  
}
```

## 2.1.278 \_task\_restart

Restarts the task.

### Prototype

```
source\kernel\task.c
_mqx_uint _task_restart(
    _task_id    task_id,
    uint32_t    *param_ptr,
    bool        blocked)
```

### Parameters

*task\_id* [IN] — Task ID of the task to restart

*param\_ptr* [IN] — One of the following:

pointer to a new task creation parameter

NULL

*blocked* [IN] — Whether to restart the task in the blocked state

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_INVALID_TASK_ID	<i>task_id</i> is invalid.

### Traits

Cannot be called from an ISR

### See Also

[\\_task\\_create](#), [\\_task\\_create\\_blocked](#), [\\_task\\_create\\_at](#), [create\\_task](#)

### Description

The function closes all queues that the task has open, releases all the task's resources, and frees all memory that is associated with the task's resources.

The function restarts the task with the same task descriptor, task ID, and task stack.

## 2.1.279 `_task_set_error`

Sets the task error code.

### Prototype

```
source\kernel\task.c
_mqx_uint _task_set_error(
    _mx_uint error_code)
```

### Parameters

*error\_code* [IN] — Task error code

### Returns

Previous task error code

### See Also

[\\_task\\_check\\_stack](#)

[\\_task\\_get\\_error, \\_task\\_get\\_error\\_ptr](#)

[\\_task\\_errno](#)

### Description

MQX RTOS uses the function to indicate an error. MQX RTOS never sets the task error code to `MQX_OK`; that is, MQX RTOS does not reset the task error code. It is the responsibility of the application to reset the task error code.

As a result, when an application calls `_task_get_error()`, it gets the first error that MQX RTOS detected since the last time the application reset the task error code.

If the current task error code is:	Function changes the task error code:
<code>MQX_OK</code>	To <i>error_code</i>
Not <b><code>MQX_OK</code></b>	To <i>error_code</i> if <i>error_code</i> is <b><code>MQX_OK</code></b>

If the function is called from an ISR, the function sets the interrupt error code.

### Example

#### Reset the task error code and check whether it was set.

```
_mx_uint error;

error = _task_set_error(MQX_OK);
if (error != MQX_OK) {
    /* Handle the error. */
}
```



## 2.1.280 `_task_start_preemption`, `_task_stop_preemption`

`_task_start_preemption()`      Enables preemption of the current task.  
`_task_stop_preemption()`      Disables preemption of the current task.

### Prototype

```
source\kernel\task.c
void _task_start_preemption(void)

void _task_stop_preemption(void)
```

### Parameters

None

### Returns

None

### Traits

- Changes the preemption ability of tasks
- Interrupts are still handled

### See Also

[`\_task\_ready`](#)

[`\_task\_block`](#)

### Description

The `_task_stop_preemption()` function disables interrupt-driven preemption of the calling task unless the task invokes the scheduler explicitly either by a blocking call (`_task_block()`), a non-blocking call (`_lwevent_set()`) or it calls `_task_start_preemption()`. When preemption is stopped, the context switch does not occur upon return from any ISR, even if a higher priority task becomes ready during the ISR execution. This includes the context switch at the end of a timeslice, therefore tasks calling the `_task_stop_preemption()` function may have their timeslice extended.

### Example

Stop a higher-priority task from preempting this task during a critical period, but allow interrupts to be serviced.

```
...
_task_stop_preemption();
/* Perform the critical operation that cannot be preempted. */
...
task_start_preemption();
```

## 2.1.281 `_taskq_create`

Creates a task queue.

### Prototype

```
source\kernel\taskq.c
void *_taskq_create(
    _mqx_uint policy)
```

### Parameters

policy [IN] — Queuing policy; one of the following:

MQX\_TASK\_QUEUE\_BY\_PRIORITY  
MQX\_TASK\_QUEUE\_FIFO

### Returns

- Pointer to the task queue (success)
- NULL (failure)

### Task error codes

Task error code	Description
Error from <code>_mem_alloc_system()</code>	MQX RTOS cannot allocate memory for the task queue.
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_INVALID_PARAMETER	<i>policy</i> is not one of the allowed policies.

### Traits

- Cannot be called from an ISR
- On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_taskq\\_destroy](#)

[\\_taskq\\_resume](#)

[\\_taskq\\_suspend](#)

[\\_task\\_set\\_error](#)

### Description

A task can use the task queue to suspend and resume tasks.

### Example

```
void *task_queue;

void TaskA(void)
{
```

```
task_queue = _taskq_create(MQX_TASK_QUEUE_FIFO);

while (condition) {
    _taskq_suspend(task_queue);
    /* Do some work. */
}

_taskq_destroy(task_queue);
}
```

## 2.1.282 `_taskq_destroy`

Destroys the task queue.

### Prototype

```
source\kernel\taskq.c
_mqx_uint _taskq_destroy(
    void *task_queue_ptr)
```

### Parameters

- `task_queue_ptr [IN]` — Pointer to the task queue to destroy; returned by `_taskq_create()`

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_INVALID_PARAMETER	<code>task_queue_ptr</code> is <code>NULL</code> .
MQX_INVALID_TASK_QUEUE	<code>task_queue_ptr</code> does not point to a valid task queue.

### Traits

- Might put tasks in their ready queues
- Cannot be called from an ISR

### See Also

[\\_task\\_create](#), [\\_task\\_create\\_blocked](#), [\\_task\\_create\\_at](#), [create\\_task](#)

[\\_taskq\\_resume](#)

[\\_taskq\\_suspend](#)

### Description

The function removes all tasks from the task queue, puts them in their ready queues, and frees the task queue.

### Example

See `_taskq_create()`.

## 2.1.283 `_taskq_get_value`

Gets the number of tasks that are in the task queue.

### Prototype

```
source\kernel\taskq.c
_mqx_uint _taskq_get_value(
    void *task_queue_ptr)
```

### Parameters

*task\_queue\_ptr* [IN] — Pointer to the task queue; returned by `_taskq_create()`

### Returns

- Number of tasks on the task queue (success)
- `MAX_MQX_UINT` (failure)

### Task Error Codes

MQX_INVALID_PARAMETER	<i>task_queue_ptr</i> is <code>NULL</code> .
MQX_INVALID_TASK_QUEUE	<i>task_queue_ptr</i> does not point to a valid task queue.

### Traits

On failure, calls `_task_set_error()` to set the task error code (see task error codes)

### See Also

[\\_taskq\\_create](#)

[\\_task\\_set\\_error](#)

## 2.1.284 `_taskq_resume`

Restarts the task that is suspended in the task queue.

### Prototype

```
source\kernel\taskq.c
_mqx_uint _taskq_resume(
    void      *task_queue,
    bool      all_tasks)
```

### Parameters

*task\_queue* [IN] — Pointer to the task queue returned by `_taskq_create()`

*all\_tasks* [IN] — One of the following:

FALSE (ready the first task)

TRUE (ready all tasks)

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_INVALID_PARAMETER	<i>task_queue_ptr</i> is not valid.
MQX_INVALID_TASK_QUEUE	<i>task_queue_ptr</i> is <i>NULL</i> .
MQX_TASK_QUEUE_EMPTY	Task queue is empty.

### Traits

Might put tasks in their ready queues

### See Also

[\\_taskq\\_destroy](#)

[\\_taskq\\_create](#)

[\\_taskq\\_suspend](#)

### Description

The function removes the task or tasks from the task queue and puts them in their ready queues. MQX RTOS schedules the tasks based on their priority, regardless of the scheduling policy of the task queue.

### Example

```
extern void *task_queue;
void TaskB(void)
{
    bool condition;
    ...
}
```

```
if (condition) {  
    /* Schedule the first waiting task: */  
    _taskq_resume(task_queue, FALSE);  
}  
...  
}
```

## 2.1.285 `_taskq_suspend`

Suspends the active task and put it in the task queue.

### Prototype

```
source\kernel\taskq.c
_mqx_uint _taskq_suspend(
    void *task_queue)
```

### Parameters

*task\_queue* [IN] — Pointer to the task queue returned by `_taskq_create()`

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_INVALID_PARAMETER	<i>task_queue_ptr</i> is <i>NULL</i> .
MQX_INVALID_TASK_QUEUE	<i>task_queue_ptr</i> does not point to a valid task queue.

### Traits

- Blocks the calling task
- Cannot be called from an ISR

### See Also

[\\_taskq\\_destroy](#)

[\\_taskq\\_create](#)

[\\_taskq\\_resume](#)

[\\_taskq\\_get\\_value](#)

### Description

The function blocks the calling task and puts the task's task descriptor in the task queue.

### Example

See `_taskq_create()`.



## 2.1.286 \_taskq\_suspend\_task

Suspends the ready task in the task queue.

### Prototype

```
source\kernel\taskq.c
_mqx_uint _taskq_suspend_task(
    _task_id task_id,
    void *task_queue_ptr)
```

### Parameters

*task\_id* [IN] — Task ID of the task to suspend

*task\_queue\_ptr* [IN] — Pointer to the task queue; returned by **\_taskq\_create()**

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_INVALID_PARAMETER	<i>task_queue_ptr</i> is <i>NULL</i> .
MQX_INVALID_TASK_ID	<i>task_id</i> is not a valid task descriptor.
MQX_INVALID_TASK_QUEUE	<i>task_queue_ptr</i> does not point to a valid task queue.
MQX_INVALID_TASK_STATE	Task is not in the ready state.

### Traits

- Blocks the specified task
- Cannot be called from an ISR

### See Also

[\\_taskq\\_destroy](#)

[\\_taskq\\_create](#)

[\\_taskq\\_resume](#)

[\\_taskq\\_get\\_value](#)

### Description

The function blocks the specified task and puts the task's task descriptor in the task queue.

**Example**

```
void *task_queue;

void TaskA(void)
{
    task_queue = _taskq_create(0);

    while (condition) {
        _taskq_suspend_task(_task_get_creator(), task_queue);
        /* Do some work. */
    }

    _taskq_destroy(task_queue);
}
```

## 2.1.287 \_taskq\_test

Tests the task queues.

### Prototype

```
source\kernel\taskq.c
_mqx_uint _taskq_test(
    void *task_queue_error_ptr,
    void *td_error_ptr)
```

### Parameters

*task\_queue\_error\_ptr* [OUT] — Pointer to the task queue with an error (*NULL* if no error is found)

*td\_error\_ptr* [OUT] — Pointer to the task descriptor with an error (*NULL* if no error is found)

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.
MQX_CORRUPT_QUEUE	A task on a task queue is not valid.
MQX_INVALID_TASK_QUEUE	A task queue is not valid.

### Traits

- Cannot be called from an ISR
- Disables and enables interrupts

### See Also

[\\_taskq\\_destroy](#)

[\\_taskq\\_create](#)

[\\_taskq\\_resume](#)

[\\_taskq\\_get\\_value](#)

## 2.1.288 `_ticks_to_time`

Converts tick format to second/millisecond format

### Prototype

```
source\kernel\time.c
bool _ticks_to_time(
    MQX_TICK_STRUCT_PTR  tick_time_ptr,
    TIME_STRUCT_PTR      time_ptr)
```

### Parameters

*tick\_time\_ptr* [IN] — Pointer to a time structure

*time\_ptr* [OUT] — Pointer to the corresponding normalized second/millisecond time structure

### Returns

- TRUE (success)
- FALSE (failure: *tick\_time\_ptr* or *time\_ptr* is NULL)

### See Also

[\\_time\\_to\\_ticks](#)

[MQX\\_TICK\\_STRUCT](#)  
[TIME\\_STRUCT](#)

### Description

The function verifies that the fields in the input structure are within the following ranges.

Field	Minimum	Maximum
TICKS	0	(2 <sup>64</sup> ) - 1
HW_TICKS	0	(2 <sup>32</sup> ) - 1

## 2.1.289 \_time\_add ...

Add time in these units to tick time:	
<code>_time_add_day_to_ticks()</code>	Days
<code>_time_add_hour_to_ticks()</code>	Hours
<code>_time_add_min_to_ticks()</code>	Minutes
<code>_time_add_sec_to_ticks()</code>	Seconds
<code>_time_add_msec_to_ticks()</code>	Milliseconds
<code>_time_add_usec_to_ticks()</code>	Microseconds
<code>_time_add_nsec_to_ticks()</code>	Nanoseconds
<code>_time_add_psec_to_ticks()</code>	Picoseconds

### Prototype

```

source\kernel\time.c
MQX_TICK_STRUCT_PTR _time_add_day_to_ticks(
    MQX_TICK_STRUCT_PTR tick_ptr,
    mqx_uint             days)

MQX_TICK_STRUCT_PTR _time_add_hour_to_ticks(
    MQX_TICK_STRUCT_PTR tick_ptr,
    mqx_uint             hours)

MQX_TICK_STRUCT_PTR _time_add_min_to_ticks(
    MQX_TICK_STRUCT_PTR tick_ptr,
    mqx_uint             mins)

MQX_TICK_STRUCT_PTR _time_add_sec_to_ticks(
    MQX_TICK_STRUCT_PTR tick_ptr,
    mqx_uint             secs)

MQX_TICK_STRUCT_PTR _time_add_msec_to_ticks(
    MQX_TICK_STRUCT_PTR tick_ptr,
    mqx_uint             msecs)

MQX_TICK_STRUCT_PTR _time_add_usec_to_ticks(
    MQX_TICK_STRUCT_PTR tick_ptr,
    mqx_uint             usecs)

MQX_TICK_STRUCT_PTR _time_add_nsec_to_ticks(
    MQX_TICK_STRUCT_PTR tick_ptr,
    mqx_uint             nsecs)

MQX_TICK_STRUCT_PTR _time_add_psec_to_ticks(
    MQX_TICK_STRUCT_PTR tick_ptr,
    mqx_uint             psecs)

```

### Parameters

*tick\_ptr* [IN] — Tick time to add to

*days* [IN] — Days to add

*hours* [IN] — Hours to add  
*mins* [IN] — Minutes to add  
*secs* [IN] — Seconds to add  
*msecs* [IN] — Milliseconds to add  
*usecs* [IN] — Microseconds to add  
*nsecs* [IN] — Nanoseconds to add  
*psecs* [IN] — Picoseconds to add

## Returns

Tick time

## See Also

[\\_mqx\\_zero\\_tick\\_struct](#)

## Description

The functions can also be used in conjunction with the global constant `_mqx_zero_tick_struct` to convert units to tick time.

## Example

Convert 265 days to ticks.

```
_mqx_uint      days;  
MQX_TICK_STRUCT ticks;  
  
...  
  
days = 365;  
ticks = _mqx_zero_tick_struct;  
_time_add_day_to_ticks(&ticks, days);
```

## 2.1.290 `_time_delay` ...

Suspend the active task:	
<code>_time_delay()</code>	For the number of milliseconds
<code>_time_delay_for()</code>	For the number of ticks (in tick time)
<code>_time_delay_ticks()</code>	For the number of ticks
<code>_time_delay_until()</code>	Until the specified time (in tick time)

### Prototype

```
source\kernel\time.c
void _time_delay(
    uint32_t milliseconds)

void _time_delay_for(
    MQX_TICK_STRUCT_PTR tick_time_delay_ptr)

void _time_delay_ticks(
    _mqx_uint tick_delay)

void _time_delay_until(
    MQX_TICK_STRUCT_PTR tick_time_ptr)
```

### Parameters

*milliseconds* [IN] — Minimum number of milliseconds to suspend the task

*tick\_time\_delay\_ptr* [IN] — Pointer to the minimum number of ticks to suspend the task

*tick\_delay* [IN] — Minimum number of ticks to suspend the task

*tick\_time\_ptr* [IN] — Pointer to the time (in tick time) until which to suspend the task

### Returns

None

### Traits

Blocks the calling task

If the requested delay equals zero, then only `_sched_yield()` function is called.

### See Also

[\\_time\\_dequeue](#)

### Description

The functions put the active task in the timeout queue for the specified time.

Before the time expires, any task can remove the task from the timeout queue by calling `_time_dequeue()`.

### Example

See `_time_dequeue()`.

## 2.1.291 `_time_dequeue`

Removes the task (specified by task ID) from the timeout queue.

### Prototype

```
source\kernel\time.c
void _time_dequeue(
    _task_id  tid)
```

### Parameters

*tid* [IN] — Task ID of the task to be removed from the timeout queue

### Returns

None

### Traits

Removes the task from the timeout queue, but does not put it in the task's ready queue

### See Also

[\\_task\\_ready](#)

[\\_time\\_delay ...](#)

[\\_time\\_dequeue\\_td](#)

### Description

The function removes from the timeout queue a task that has put itself there for a period of time (`_time_delay()`).

If *tid* is invalid or represents a task that is on another processor, the function does nothing.

A task that calls the function must subsequently put the task in the task's ready queue with `_task_ready()`.

### Example

Task A creates Task B and then waits for Task B to remove it from the timeout queue and ready it using its task descriptor. Task A then creates Task C and waits for Task C to remove it from the timeout queue and ready it using its task ID.

```
void taskB(uint32_t parameter)
{
    void *td_ptr;
    td_ptr = (void*)parameter;
    ...
    _time_dequeue_td(td_ptr);
    _task_ready(td_ptr);
    ...
}
void taskC(uint32_t parameter)
{
    ...
```



```
_time_dequeue((_task_id)parameter);
_task_ready(_task_get_td((_task_id)parameter);
...
}

void taskA(uint32_t parameter)
{
    ...
    _task_create(0, TASKB, (uint32_t)_task_get_td(_task_get_id()));
    _time_delay(100);
    ...
    _task_create(0, TASKC, (uint32_t)_task_get_id());
    _time_delay(100);
    ...
}
```

## 2.1.292 `_time_dequeue_td`

Removes the task (specified by task descriptor) from the timeout queue.

### Prototype

```
source\kernel\time.c  
void _time_dequeue_td(  
    void *td)
```

### Parameters

*td* [IN] — Pointer to the task descriptor of the task to be removed from the timeout queue

### Returns

None

### Traits

Removes the task from the timeout queue; does not put it in the task's ready queue

### See Also

[\\_task\\_ready](#)

[\\_time\\_delay ...](#)

[\\_time\\_dequeue](#)

### Description

See `_time_dequeue()`.

### Example

See `_time_dequeue()`.

## 2.1.293 `_time_diff`, `_time_diff_ticks`

For `_time_diff_units` functions, see `_time_diff_ ...`

Get the difference between two:	
<code>_time_diff()</code>	Second/millisecond times
<code>_time_diff_ticks()</code>	Tick times

### Prototype

```
source\kernel\time.c
void _time_diff(
    TIME_STRUCT_PTR start_time_ptr,
    TIME_STRUCT_PTR end_time_ptr,
    TIME_STRUCT_PTR diff_time_ptr)

_mqx_uint _time_diff_ticks(
    MQX_TICK_STRUCT_PTR tick_end_time_ptr,
    MQX_TICK_STRUCT_PTR tick_start_time_ptr,
    MQX_TICK_STRUCT_PTR tick_diff_time_ptr)
```

### Parameters

- *start\_time\_ptr* [IN] — Pointer to the normalized start time in second/millisecond time
- *end\_time\_ptr* [IN] — Pointer to the normalized end time, which must be greater than the start time
- *diff\_time\_ptr* [OUT] — Pointer to the time difference (the time is normalized)
- *tick\_start\_time\_ptr* [IN] — Pointer to the normalized start time in tick time
- *tick\_end\_time\_ptr* [IN] — Pointer to the normalized end time, which must be greater than the start time
- *tick\_diff\_time\_ptr* [OUT] — Pointer to the time difference (the time is normalized)

### Returns

For `_time_diff_ticks()`:

- `MQX_OK`
- `MQX_INVALID_PARAMETER` (one or more pointers are NULL)

### See Also

Other functions in the `_time_diff_ ...` family

[\\_time\\_get, \\_time\\_get\\_ticks](#)

[\\_time\\_set, \\_time\\_set\\_ticks](#)

[MQX\\_TICK\\_STRUCT](#)

[TIME\\_STRUCT](#)

## Example

Determine how long it takes to send 100 messages.

```
TIME_STRUCT    start_time, end_time, diff_time;
...
_time_get(&start_time);

/* Send 100 messages. */

_time_get(&end_time);
_time_diff(&start_time, &end_time, &diff_time);

printf("Time to send 100 messages: %ld sec %ld millisec\n",
       diff_time.SECONDS, diff_time.MILLISECONDS);
```

## 2.1.294 `_time_diff_` ...

Get the difference in this unit between two tick times:	
<code>_time_diff_days()</code>	Days
<code>_time_diff_hours()</code>	Hours
<code>_time_diff_minutes()</code>	Minutes
<code>_time_diff_seconds()</code>	Seconds
<code>_time_diff_milliseconds()</code>	Milliseconds
<code>_time_diff_microseconds()</code>	Microseconds
<code>_time_diff_nanoseconds()</code>	Nanoseconds
<code>_time_diff_picoseconds()</code>	Picoseconds
<code>_time_diff_ticks()</code>	See <code>_time_diff()</code> , <code>_time_diff_ticks()</code>

### Prototype

```

source\kernel\time.c
int32_t _time_diff_days(
    MQX_TICK_STRUCT_PTR end_tick_ptr,
    MQX_TICK_STRUCT_PTR start_tick_ptr,
    bool *overflow_ptr)

int32_t _time_diff_hours(
    MQX_TICK_STRUCT_PTR end_tick_ptr,
    MQX_TICK_STRUCT_PTR start_tick_ptr,
    bool *overflow_ptr)

int32_t _time_diff_minutes(
    MQX_TICK_STRUCT_PTR end_tick_ptr,
    MQX_TICK_STRUCT_PTR start_tick_ptr,
    bool *overflow_ptr)

int32_t _time_diff_seconds(
    MQX_TICK_STRUCT_PTR end_tick_ptr,
    MQX_TICK_STRUCT_PTR start_tick_ptr,
    bool *overflow_ptr)

int32_t _time_diff_milliseconds(
    MQX_TICK_STRUCT_PTR end_tick_ptr,
    MQX_TICK_STRUCT_PTR start_tick_ptr,
    bool *overflow_ptr)

int32_t _time_diff_microseconds(
    MQX_TICK_STRUCT_PTR end_tick_ptr,
    MQX_TICK_STRUCT_PTR start_tick_ptr,
    bool *overflow_ptr)

int32_t _time_diff_nanoseconds(
    MQX_TICK_STRUCT_PTR end_tick_ptr,
    MQX_TICK_STRUCT_PTR start_tick_ptr,
    bool *overflow_ptr)

```

```
int32_t _time_diff_picoseconds(
    MQX_TICK_STRUCT_PTR  end_tick_ptr,
    MQX_TICK_STRUCT_PTR  start_tick_ptr,
    bool                  *overflow_ptr)
```

### Parameters

*end\_tick\_ptr* [IN] — Pointer to the ending tick time, which must be greater than the starting tick time

*start\_tick\_ptr* [IN] — Pointer to the starting tick time

*overflow\_ptr* [OUT] — *TRUE* if overflow occurs (see description)

### Returns

Difference in days, hours, minutes, seconds, or so on

### See Also

[\\_mqx\\_zero\\_tick\\_struct](#)

[\\_time\\_diff, \\_time\\_diff\\_ticks](#)

[\\_time\\_get, \\_time\\_get\\_ticks](#)

[\\_time\\_set, \\_time\\_set\\_ticks](#)

[MQX\\_TICK\\_STRUCT](#)

### Description

If the calculation overflows **int32\_t**, the function sets the boolean at *overflow\_ptr* to *TRUE*. If this happens, use the **\_time\_diff** function for a larger unit. For example, if **\_time\_diff\_hours()** sets the overflow, use **\_time\_diff\_days()**.

The functions can also be used in conjunction with the global constant *\_mqx\_zero\_tick\_struct* to convert tick time to units.

## Example

```
bool    overflow = FALSE;
int32_t nsecs;
MQX_TICK_STRUCT ticks;

...

nsecs = _time_diff_nanoseconds(&ticks, &_mqx_zero_tick_struct,
    &overflow);
```

## 2.1.295 `_time_from_date`

Gets second/millisecond time format from date format.

### Prototype

```
source\kernel\time.c
bool _time_from_date(
    DATE_STRUCT_PTR  date_ptr,
    TIME_STRUCT_PTR  ms_time_ptr)
```

### Parameters

*date\_ptr* [IN] — Pointer to a date structure

*ms\_time\_ptr* [OUT] — Pointer to a normalized second/millisecond time structure

### Returns

- TRUE (success)
- FALSE (failure: see description)

### See Also

[\\_time\\_get, \\_time\\_get\\_ticks](#)

[\\_time\\_get\\_elapsed, \\_time\\_get\\_elapsed\\_ticks](#)

[\\_time\\_set, \\_time\\_set\\_ticks](#)

[\\_time\\_to\\_date](#)

[DATE\\_STRUCT](#)

[TIME\\_STRUCT](#)

### Description

The function verifies that the fields in the input structure are within the following ranges.

Field	Minimum	Maximum
YEAR	1970	2099
MONTH	1	12
DAY	1	31 (depending on the month)
HOUR	0	23 (since midnight)
MINUTE	0	59
SECOND	0	59
MILLISEC	0	999

The function converts the fields in the input structure to the fields in the output structure, taking into account leap years.

The time is since 0:00:00.00, January 1, 1970.

The function returns *FALSE* if either:

- *date\_ptr* or *time\_ptr* are *NULL*



- fields in *date\_ptr* are out of range

### Example

Change the time to 10:00:00.00, February 8, 1999.

```
DATE_STRUCT  date;
TIME_STRUCT  time;
...
date.YEAR    = 1999;
date.MONTH   = 2;
date.DAY     = 8;
date.HOUR    = 10;
date.SECOND  = 0;
date.MILLISEC = 0;

_time_from_date(&date, &time);
_time_set(&time);
```

## 2.1.296 `_time_get`, `_time_get_ticks`

Get the absolute time in:	
<code>_time_get()</code>	Second/millisecond time
<code>_time_get_ticks()</code>	Tick time

### Prototype

```
source\kernel\time.c
void _time_get(
    TIME_STRUCT_PTR  ms_time_ptr)

void _time_get_ticks(
    MQX_TICK_STRUCT_PTR  tick_time_ptr)
```

### Parameters

*ms\_time\_ptr* [OUT] — Where to store the normalized absolute time in second/millisecond time  
*tick\_time\_ptr* [OUT] — Where to store the absolute time in tick time

### Returns

None

### See Also

[\\_time\\_get\\_elapsed](#), [\\_time\\_get\\_elapsed\\_ticks](#)

[\\_time\\_set](#), [\\_time\\_set\\_ticks](#)

[MQX\\_TICK\\_STRUCT](#)

[TIME\\_STRUCT](#)

### Description

If the application changed the absolute time with `_time_set()` (or `_time_set_ticks()`), `_time_get()` (or `_time_get_ticks()`) returns the time that was set plus the number of seconds and milliseconds (or ticks) since the time was set.

If the application has not changed the absolute time with `_time_set()` (or `_time_set_ticks()`), `_time_get()` (or `_time_get_ticks()`) returns the same as `_time_get_elapsed()` (or `_time_get_elapsed_ticks()`), which is the number of seconds and milliseconds (or ticks) since MQX RTOS started.

### Example

See `_time_diff()`.

## 2.1.297 `_time_get_elapsed`, `_time_get_elapsed_ticks`

	Get the time in this format since MQX RTOS started:
<code>_time_get_elapsed()</code>	Second/millisecond time
<code>_time_get_elapsed_ticks()</code>	Tick time

### Prototype

```
source\kernel\time.c
void _time_get_elapsed(
    TIME_STRUCT_PTR  ms_time_ptr)

void _time_get_elapsed_ticks(
    MQX_TICK_STRUCT_PTR  tick_time_ptr)
```

### Parameters

*ms\_time\_ptr* [OUT] — Where to store the elapsed normalized second/millisecond  
*timetick\_time\_ptr* [OUT] — Where to store the elapsed tick time

### Returns

None

### See Also

[\\_time\\_get](#), [\\_time\\_get\\_ticks](#)

[\\_time\\_set](#), [\\_time\\_set\\_ticks](#)

[TIME\\_STRUCT](#)

[MQX\\_TICK\\_STRUCT](#)

### Description

The function always returns elapsed time; it is not affected by `_time_set()` or `_time_set_ticks()`.

## 2.1.298 `_time_get_hwticks`

Gets the number of hardware ticks since the last tick.

### Prototype

```
source\kernel\time.c  
uint32_t _time_get_hwticks(void)
```

### Parameters

None

### Returns

Number of hardware ticks since the last tick

### See Also

[`\_time\_get\_hwticks\_per\_tick`](#), [`\_time\_set\_hwticks\_per\_tick`](#)

## 2.1.299 `_time_get_hwticks_per_tick`, `_time_set_hwticks_per_tick`

`_time_get_hwticks_per_tick()` Gets the number of hardware ticks per tick.  
`_time_set_hwticks_per_tick()` Sets the number of hardware ticks per tick.

### Prototype

```
source\kernel\time.c  
uint32_t _time_get_hwticks_per_tick(void)  
  
void _time_set_hwticks_per_tick(  
    uint32_t new_ticks)
```

### Parameters

*new\_ticks [OUT]* — New number of hardware ticks per tick

### Returns

`_time_get_hwticks()`: Number of hardware ticks per tick

### See Also

[\\_time\\_get\\_hwticks](#)

## 2.1.300 `_time_get_microseconds`

Gets the calculated number of microseconds since the last periodic timer interrupt.

### Prototype

```
source\bsp\platform\get_usec.c  
uint16_t _time_get_microseconds(void)
```

### Parameters

None

### Returns

- Number of microseconds since the last periodic timer interrupt
- 0 (BSP does not support the feature)

### Traits

Resolution depends on the periodic timer device

### See Also

[\\_time\\_get\\_elapsed, \\_time\\_get\\_elapsed\\_ticks](#)

[\\_time\\_get, \\_time\\_get\\_ticks](#)

[\\_time\\_set, \\_time\\_set\\_ticks](#)

## 2.1.301 `_time_get_nanoseconds`

Gets the calculated number of nanoseconds since the last periodic timer interrupt.

### Prototype

```
source\bsp\platform\get_nsec.c  
uint32_t _time_get_nanoseconds(void)
```

### Parameters

None

### Returns

- Number of nanoseconds since the last periodic timer interrupt
- 0 (BSP does not support the feature)

### Traits

Resolution depends on the periodic timer device

### See Also

[\\_time\\_get\\_elapsed, \\_time\\_get\\_elapsed\\_ticks](#)

[\\_time\\_get, \\_time\\_get\\_ticks](#)

[\\_time\\_set, \\_time\\_set\\_ticks](#)

## 2.1.302 `_time_get_resolution`, `_time_set_resolution`

`_time_get_resolution()` Gets the resolution of the periodic timer interrupt.  
`_time_set_resolution()` Sets the resolution of the periodic timer interrupt.

### Prototype

```
source\kernel\time.c
_mqx_uint _time_get_resolution(void)

_mqx_uint _time_set_resolution(
    _mx_uint resolution)
```

### Parameters

*resolution [IN]* — Periodic timer resolution (in milliseconds) that MQX RTOS is to use

### Returns

- `_time_get_resolution()`:  
Resolution of the periodic timer interrupt in milliseconds
- `_time_set_resolution()`:
  - `MQX_OK`
  - `MQX_INVALID_PARAMETER` (input resolution is equal to 0 or greater than 1000 milliseconds)

### See Also

[\\_time\\_get\\_elapsed, \\_time\\_get\\_elapsed\\_ticks](#)

[\\_time\\_get, \\_time\\_get\\_ticks](#)

[\\_time\\_set, \\_time\\_set\\_ticks](#)

[TIME\\_STRUCT](#)

### Description

On each clock interrupt, MQX RTOS increments time by the resolution.

### CAUTION

If the resolution does not agree with the interrupt period that was programmed at the hardware level, some timing functions gives incorrect results.



## 2.1.303 `_time_get_ticks_per_sec`, `_time_set_ticks_per_sec`

- `_time_get_ticks_per_sec()` Gets the timer frequency (in ticks per second) that MQX RTOS uses.
- `_time_set_ticks_per_sec()` Sets the timer frequency (in ticks per second) that MQX RTOS uses.

### Prototype

```
source\kernel\time.c
_mqx_uint _time_get_ticks_per_sec(void)

void _time_set_ticks_per_sec(
    _mx_uint ticks_per_sec)
```

### Parameters

*ticks\_per\_sec [IN]* — New timer frequency in ticks per second

### Returns

- `_time_get_ticks_per_sec()`:  
Period of clock interrupt in ticks per second
- `_time_set_ticks_per_sec()`:  
None

### CAUTION

If the timer frequency does not agree with the interrupt period that was programmed at the hardware level, some timing functions gives incorrect results.

## 2.1.304 `_time_init_ticks`

Initializes a tick time structure with the number of ticks.

### Prototype

```
source\kernel\time.c
_mqx_uint _time_init_ticks(
    MQX_TICK_STRUCT_PTR tick_time_ptr,
    _mx_uint             ticks)
```

### Parameters

*tick\_time\_ptr* [OUT] — Pointer to the tick time structure to initialize  
*ticks* [IN] — Number of ticks with which to initialize the structure

### Returns

- TRUE (success)
- FALSE (failure: input year is earlier than 1970 or output year is later than 2481)

### See Also

[\\_time\\_set, \\_time\\_set\\_ticks](#)

[MQX\\_TICK\\_STRUCT](#)

## 2.1.305 `_time_notify_kernel`

The BSP periodic timer ISR calls the function when a periodic timer interrupt occurs.

### Prototype

```
source\kernel\time.c  
void _time_notify_kernel(void)
```

### Parameters

None

### Returns

None

### Traits

See description

### See Also

[\\_time\\_get\\_elapsed, \\_time\\_get\\_elapsed\\_ticks](#)

[\\_time\\_get, \\_time\\_get\\_ticks](#)

[\\_time\\_set, \\_time\\_set\\_ticks](#)

### TIME\_STRUCT

### Description

The BSP installs an ISR for the periodic timer interrupt. The ISR calls `_time_notify_kernel()`, which does the following:

- increments kernel time
- if the active task is a time slice task whose time slice has expired, puts it at the end of the task's ready queue
- if the timeout has expired for tasks on the timeout queue, puts them in their ready queues

If the BSP does not have periodic timer interrupts, MQX RTOS components that use time does not operate.

## 2.1.306 `_time_set`, `_time_set_ticks`

Set the absolute time in:	
<code>_time_set()</code>	Second/millisecond time
<code>_time_set_ticks()</code>	Tick time

### Prototype

```
source\kernel\time.c
void _time_set(
    TIME_STRUCT_PTR  ms_time_ptr)

void _time_set_ticks(
    MQX_TICK_STRUCT_PTR  tick_time_ptr)
```

### Parameters

*ms\_time\_ptr* [IN] — Pointer to a structure that contains the new normalized time in second/millisecond time

*tick\_time\_ptr* [IN] — Pointer to the structure that contains the new time in tick time

### Returns

None

### See Also

[\\_time\\_get](#), [\\_time\\_get\\_ticks](#)

[\\_time\\_get\\_elapsed](#), [\\_time\\_get\\_elapsed\\_ticks](#)

[\\_time\\_to\\_date](#)

[\\_time\\_init\\_ticks](#)

[\\_time\\_to\\_ticks](#)

[\\_time\\_from\\_date](#)

**TIME\_STRUCT**

**MQX\_TICK\_TIME**Prototype

### Description

The function affects `_time_get()` (and `_time_get_ticks()`), but does not affect time `_time_get_elapsed()` (or `_time_get_elapsed_ticks()`).

### Example

See `_time_from_date()`.

## 2.1.307 `_time_set_timer_vector`

Sets the periodic timer interrupt vector number that MQX RTOS uses.

### Prototype

```
source\kernel\time.c  
void _time_set_timer_vector(  
    _mqx_uint  vector)
```

### Parameters

*vector [IN]* — Periodic timer interrupt vector to use

### Returns

None

### See Also

[\\_time\\_get, \\_time\\_get\\_ticks](#)

[\\_time\\_get\\_resolution, \\_time\\_set\\_resolution](#)

### Description

The BSP should call the function during initialization.

## 2.1.308 `_time_to_date`

Converts time format to date format.

### Prototype

```
source\kernel\time.c
bool _time_to_date(
    TIME_STRUCT_PTR  time_ptr,
    DATE_STRUCT_PTR  date_ptr)
```

### Parameters

*time\_ptr* [IN] — Pointer to a normalized second/millisecond time structure  
*date\_ptr* [OUT] — Pointer to the corresponding date structure

### Returns

- TRUE (success)
- FALSE (failure: see description)

### See Also

[\\_time\\_get, \\_time\\_get\\_ticks](#)

[\\_time\\_get\\_elapsed, \\_time\\_get\\_elapsed\\_ticks](#)

[\\_time\\_set, \\_time\\_set\\_ticks](#)

[\\_time\\_from\\_date](#)

[DATE\\_STRUCT](#)

[TIME\\_STRUCT](#)

### Description

The function verifies that the fields in the input structure are within the following ranges.

Field	Minimum	Maximum
<b>SECONDS</b>	0	<b>MAXIMUM_SECONDS_IN_TIME</b> (4,102,444,800)
<b>MILLISECONDS</b>	0	999

The function converts the fields in the input structure to the fields in the output structure, taking into account leap years.

The time is since 0:00:00.00, January 1, 1970.

The function returns *FALSE* if either:

- *date\_ptr* or *time\_ptr* is *NULL*
- fields in *time\_ptr* are out of range

## 2.1.309 `_time_to_ticks`

Converts second/millisecond time format to tick time format.

### Prototype

```
source\kernel\time.c
bool _time_to_ticks(
    TIME_STRUCT_PTR    time_ptr,
    MQX_TICK_STRUCT_PTR tick_time_ptr)
```

### Parameters

*time\_ptr* [IN] — Pointer to a normalized second/millisecond time structure

*tick\_time\_ptr* [OUT] — Pointer to the corresponding tick time structure

### Returns

- TRUE (success)
- FALSE (failure: *time\_ptr* or *tick\_time\_ptr* is NULL)

### See Also

[\\_ticks\\_to\\_time](#)

[MQX\\_TICK\\_STRUCT](#)

[TIME\\_STRUCT](#)

### Description

The function verifies that the fields in the input structure are within the following ranges.

Field	Minimum	Maximum
<b>SECONDS</b>	0	<b>MAXIMUM_SECONDS_IN_TIME</b>
		(4,102,444,800)
<b>MILLISECONDS</b>	0	999

The function converts the fields in the input structure to the fields in the output structure, taking into account leap years.

## 2.1.310 mktime

Converts tm struct time format to second time format

### Prototype

```
source\kernel\time.c  
time_t mktime(struct tm* tm_ptr)
```

### Parameters

*tm\_ptr*[IN] — Pointer to broken-down time format (tm struct).

### Returns

- time value converted from tm struct (SUCCESS)
- 0 (FAILURE: tm\_ptr is NULL or overflow occurs)

### See Also

[gmtime\\_r](#)

[timegm](#)

[localtime\\_r](#)

[time\\_t](#)

### TM STRUCT

### Description

The mktime() function converts a broken-down time structure, expressed as local time, to calendar time representation. The function could normalize tm\_wday and tm\_yday fields in case these members are outside their valid value.



## 2.1.311 gmtime\_r

Converts calendar time format to broken-down time representation

### Prototype

```
source\kernel\time.c
struct tm *gmtime_r(
    const time_t *timep,
    struct tm *result)
```

### Parameters

*timep*[IN] — Pointer to a calendar time.

*result*[OUT] — Pointer to the corresponding tm structure format.

### Returns

- Pointer to user-supplied struct (SUCCESS)
- NULL (FAIL)

### See Also

[mktime](#)

[timegm](#)

[localtime\\_r](#)

[time\\_t](#)

### TM STRUCT

### Description

The `gmtime_r()` function converts the calendar time `timep` to broken-down time representation, expressed in UTC. The function returns NULL in case the input parameter is overflow. The result is stored in user-supplied struct.

## 2.1.312 timegm

Converts the broken-down time format to calendar time representation.

### Prototype

```
source\kernel\time.c
time_t timegm(struct tm *tm_ptr)Parameters
tm_ptr[IN] – Pointer to a broken-down time format (tm structure).
```

### Returns

- The calendar time value (SUCCESS)
- 0 (FAIL)

### See Also

[mktime](#)

[gmtime\\_r](#)

[localtime\\_r](#)

[time\\_t](#)

### TM STRUCT

### Description

The timegm() function is the inverse of gmtime. It converts the broken-down time representation, expressed in UTC, to a calendar time format.

## 2.1.313 localtime\_r

Converts the calendar time representation to broken-down time format.

### Prototype

```
source\kernel\time.c
struct tm *localtime_r(
    const time_t *timep,
    struct tm *result)
```

### Parameters

*timep*[IN] — Pointer to a calendar time.

*Result*[OUT] — Pointer to a broken-down time (tm structure).

### Returns

- pointer to user-supplied struct (SUCCESS)
- NULL (FAIL)

### See Also

[mktime](#)

[timegm](#)

[localtime\\_r](#)

[time\\_t](#)

[TM STRUCT](#)

### Description

The `localtime_r()` function converts the calendar time format to broken-down time representation, expressed relative to the user's specified timezone. The result is stored in a user-supplied structure.

## 2.1.314 `_timer_cancel`

Cancels an outstanding timer request.

### Prototype

```
source\kernel\timer.c
#include <timer.h>
_mqx_uint _timer_cancel(
_timer_id id)
```

### Parameters

*id* [IN] — ID of the timer to be cancelled, from calling a function from the **`_timer_start`** family of functions

### Returns

- MQX\_OK
- Errors

Error	Description
MQX_COMPONENT_DOES_NOT_EXIST	Timer component is not created.
MQX_INVALID_COMPONENT_BASE	Timer component data is no longer valid.
MQX_INVALID_PARAMETER	<i>id</i> is not valid.
MQX_CANNOT_CALL_FUNCTION_FROM_ISR	Function cannot be called from an ISR.

### See Also

[\\_timer\\_start\\_oneshot\\_after ...](#)

[\\_timer\\_start\\_oneshot\\_at ...](#)

[\\_timer\\_start\\_periodic\\_at ...](#)

[\\_timer\\_start\\_periodic\\_every ...](#)

### Example

See `_timer_create_component()`.

## 2.1.315 `_timer_create_component`

Creates the timer component.

### Prototype

```
source\kernel\timer.c
#include <timer.h>
_mqx_uint  _timer_create_component(
_mqx_uint  timer_task_priority,
_mqx_uint  timer_task_stack_size)
```

### Parameters

*timer\_task\_priority* [IN] — Priority of Timer Task

*timer\_task\_stack\_size* [IN] — Stack size (in single-addressable units) for Timer Task

### Returns

- `MQX_OK` (success: see description)
- `MQX_OUT_OF_MEMORY` — MQX RTOS cannot allocate memory for Timer Task or for timer component data.
- `MQX_CANNOT_CALL_FUNCTION_FROM_ISR` — Function cannot be called from an ISR.

### Traits

Creates Timer Task

### See Also

[\\_timer\\_start\\_oneshot\\_after ...](#)

[\\_timer\\_start\\_oneshot\\_at ...](#)

[\\_timer\\_start\\_periodic\\_at ...](#)

[\\_timer\\_start\\_periodic\\_every ...](#)

[\\_timer\\_cancel](#)

### Description

If the timer component is not explicitly created, MQX RTOS creates it with default values the first time that a task calls one of the functions from the `_timer_start` family.

The default values are:

- `TIMER_DEFAULT_TASK_PRIORITY`
- `TIMER_DEFAULT_STACK_SIZE`

The function returns `MQX_OK` if either:

- timer component is created
- timer component was previously created and the configuration is not changed

## Example

Create the timer component, start a periodic timer that sets an event every 20 milliseconds, and later cancel the timer.

```
void timer_set_event
(
    _timer_id    timer_id,
    void          *event_ptr,
    uint32_t      seconds,
    uint32_t      milliseconds
)

{

    if (_event_set(event_ptr, 0x01) != MQX_OK) {
        printf("\nSet Event failed");
        _mqx_exit(1);
    }
}

Void TaskA(uint32_t parameter)
{
    _timer_id    timer;
    ...
    if (_timer_create_component(TIMER_TASK_PRIORITY,
        TIMER_TASK_STACK_SIZE)
        != MQX_OK){
        _mqx_exit(1);
    }

    if (_event_create("timer") == MQX_OK) {
        if (_event_open("timer", &event_ptr) == MQX_OK) {
            timer = _timer_start_periodic_every(timer_set_event,
                event_ptr,
                TIMER_KERNEL_TIME_MODE, 20L);
            if (timer == TIMER_NULL_ID) {
                printf("\n_timer_start_periodic_every() failed.");
                _mqx_exit(1L);
            }
            for (i = 0; i < 10; i++) {
                if (_event_wait_all(event_ptr, 0x01L, 0L) == MQX_OK) {
                    printf("\nEvent 0x01 was set");
                    if (_event_clear(event_ptr, 0x01L) != MQX_OK) {
                        _mqx_exit(1L);
                    }
                } else {
                    _mqx_exit(1L);
                }
            }
        }
    }
}
```

```
    }  
  }  
  _timer_cancel(timer);  
  ...  
}
```

## 2.1.316 \_timer\_start\_oneshot\_after ...

Start a timer that expires after the number of:	
<code>_timer_start_oneshot_after()</code>	Milliseconds
<code>_timer_start_oneshot_after_ticks()</code>	Ticks (in tick time)

### Prototype

```
source\kernel\timer.c
#include <timer.h>
_timer_id _timer_start_oneshot_after(
    TIMER_NOTIFICATION_TIME_FPTR notification_function,
    void *notification_data_ptr,
    _mqx_uint mode,
    uint32_t milliseconds)

_timer_id _timer_start_oneshot_after_ticks(
    TIMER_NOTIFICATION_TICK_FPTR notification_function),
    void *notification_data_ptr,
    _mqx_uint mode,
    MQX_TICK_STRUCT_PTR tick_time_ptr)
```

### Parameters

*notification\_function* [IN] — Notification function that MQX RTOS calls when the timer expires

*notification\_data\_ptr* [IN] — Data that MQX RTOS passes to the notification function

*mode* [IN] — Time to use when calculating the time to expire; one of the following:

**TIMER\_ELAPSED\_TIME\_**

**MODE** (use `_time_get_elapsed()` or `_time_get_elapsed_ticks()`, which are not affected by `_time_set()` or `_time_set_ticks()`)

**TIMER\_KERNEL\_TIME\_**

**MODE** (use `_time_get()` or `_time_get_ticks()`)

*milliseconds* [IN] — Milliseconds to wait before MQX RTOS calls the notification function and cancels the timer

*tick\_time\_ptr* [IN] — Ticks (in tick time) to wait before MQX RTOS calls the notification function and cancels the timer

### Returns

- Timer ID (success)
- **TIMER\_NULL\_ID** (failure)



## Task Error Codes

Task Error Code	Description
MQX_INVALID_COMPONENT_BASE	Timer component data is no longer valid.
MQX_INVALID_PARAMETER	One of the following: <ul style="list-style-type: none"> <li>• mode is not one of the allowed modes</li> <li>• notification_function is NULL</li> <li>• milliseconds is 0</li> <li>• tick_time_ptr is NULL</li> </ul>
MQX_OUT_OF_MEMORY	MQX RTOS cannot allocate memory for the timer data.

## Traits

- Creates the timer component with default values if it was not previously created
- On failure, calls `_task_set_error()` to set the task error code (see task error codes)

## See Also

[\\_task\\_set\\_error](#)

[\\_time\\_get\\_elapsed, \\_time\\_get\\_elapsed\\_ticks](#)

[\\_time\\_get, \\_time\\_get\\_ticks](#)

[\\_time\\_set, \\_time\\_set\\_ticks](#)

[\\_timer\\_cancel](#)

[\\_timer\\_start\\_oneshot\\_at ...](#)

[\\_timer\\_start\\_periodic\\_at ...](#)

[\\_timer\\_start\\_periodic\\_every ...](#)

[\\_timer\\_create\\_component](#)

## Description

The function calculates the expiry time based on *milliseconds* or (*tick\_time\_ptr*) and *mode*.

You might need to increase the Timer Task stack size to accommodate the notification function (see `_timer_create_component()`).

## 2.1.317 \_timer\_start\_oneshot\_at ...

Start a timer that expires once at the specified time in:	
<code>_timer_start_oneshot_at()</code>	Second/millisecond time
<code>_timer_start_oneshot_at_ticks()</code>	Tick time

### Prototype

```
source\kernel\timer.c
#include <timer.h>
_timer_id _timer_start_oneshot_at(
    TIMER_NOTIFICATION_TIME_FPTR notification_function,
    void *notification_data_ptr,
    _mqx_uint mode,
    TIME_STRUCT_PTR ms_time_ptr)

#include <timer.h>
_timer_id _timer_start_oneshot_at_ticks(
    TIMER_NOTIFICATION_TICK_FPTR notification_function,
    void *notification_data_ptr,
    _mqx_uint mode,
    MQX_TICK_STRUCT_PTR tick_time_ptr)
```

### Parameters

*notification\_function* [IN] — Pointer to the notification function that MQX RTOS calls when the timer expires

*notification\_data\_ptr* [IN] — Pointer to the data that MQX RTOS passes to the notification function

*mode* [IN] — Time to use when calculating the time to expire; one of the following:

**TIMER\_ELAPSED\_TIME\_**

**MODE** (use `_time_get_elapsed()` or `_time_get_elapsed_ticks()`, which are not affected by `_time_set()` or `_time_set_ticks()`)

**TIMER\_KERNEL\_TIME\_**

**MODE** (use `_time_get()` or `_time_get_ticks()`)

*ms\_time\_ptr* [IN] — Pointer to the normalized second/millisecond time at which MQX RTOS calls the notification function and cancels the timer

*tick\_time\_ptr* [IN] — Pointer to the tick time at which MQX RTOS calls the notification function and cancels the timer

### Returns

- Timer ID (success)
- **TIMER\_NULL\_ID** (failure)

### Traits

- Creates the timer component with default values if it was not previously created
- On failure, calls `_task_set_error()` to set the task error code (see task error codes)

#### See Also

[\\_timer\\_cancel](#)

[\\_timer\\_start\\_oneshot\\_after ...](#)

[\\_timer\\_start\\_periodic\\_at ...](#)

[\\_timer\\_start\\_periodic\\_every ...](#)

[\\_task\\_set\\_error](#)

[\\_timer\\_create\\_component](#)

#### Description

When the timer expires, MQX RTOS calls *notification\_function* with *timer\_id*, *notification\_data\_ptr*, and the current time.

You might need to increase the Timer Task stack size to accommodate the notification function (see `_timer_create_component()`).

#### Task error codes

Task Error Code	Description
MQX_INVALID_COMPONENT_BASE	Timer component data is no longer valid.
MQX_INVALID_PARAMETER	One of the following: <ul style="list-style-type: none"> <li>• mode is not one of the allowed modes</li> <li>• notification_function is NULL</li> <li>• time_ptr is NULL</li> </ul>
MQX_OUT_OF_MEMORY	MQX RTOS cannot allocate memory for the timer data.

## 2.1.318 \_timer\_start\_periodic\_at ...

	Start a periodic timer at the specified time in:
<code>_timer_start_periodic_at()</code>	Second/millisecond time
<code>_timer_start_periodic_at_ticks()</code>	Tick time

### Prototype

```
source\kernel\timer.c
#include <timer.h>
_timer_id _timer_start_periodic_at(
    TIMER_NOTIFICATION_TIME_FPTR notification_function,
    void *notification_data_ptr,
    _mqx_uint mode,
    TIME_STRUCT_PTR ms_time_start_ptr,
    uint32_t ms_wait)

#include <timer.h>
_timer_id _timer_start_periodic_at_ticks(
    TIMER_NOTIFICATION_TICK_FPTR notification_function,
    void *notification_data_ptr,
    _mqx_uint mode,
    MQX_TICK_STRUCT_PTR tick_time_start_ptr,
    MQX_TICK_STRUCT_PTR tick_time_wait_ptr)
```

### Parameters

*notification\_function* [IN] — Pointer to the notification function that MQX RTOS calls when the timer expires

*notification\_data\_ptr* [IN] — Pointer to the data that MQX RTOS passes to the notification function

*mode* [IN] — Time to use when calculating the time to expire; one of the following:

**TIMER\_ELAPSED\_TIME\_MODE** (use `_time_get_elapsed()` or `_time_get_elapsed_ticks()`, which are not affected by `_time_set()` or `_time_set_ticks()`)

**TIMER\_KERNEL\_TIME\_MODE** (use `_time_get()` or `_time_get_ticks()`)

*ms\_time\_start\_ptr* [IN] — Pointer to the normalized second/millisecond time at which MQX RTOS starts calling the notification function

*ms\_wait* [IN] — Milliseconds that MQX RTOS waits between subsequent calls to the notification function

*tick\_time\_start\_ptr* [IN] — Pointer to the tick time at which MQX RTOS starts calling the notification function

*tick\_time\_wait\_ptr* [IN] — Ticks (in tick time) that MQX RTOS waits between subsequent calls to the notification function

### Returns

- Timer ID (success)
- **TIMER\_NULL\_ID** (failure)

**Traits**

- Creates the timer component with default values if it was not previously created
- On failure, calls **\_task\_set\_error()** to set the task error code as described for **\_timer\_start\_oneshot\_at()**

**See Also**

[\\_timer\\_cancel](#)

[\\_timer\\_start\\_oneshot\\_after ...](#)

[\\_timer\\_start\\_oneshot\\_at ...](#)

[\\_timer\\_start\\_periodic\\_every ...](#)

[\\_time\\_get, \\_time\\_get\\_ticks](#)

[\\_time\\_get\\_elapsed, \\_time\\_get\\_elapsed\\_ticks](#)

[\\_task\\_set\\_error](#)

[\\_timer\\_create\\_component](#)

**Description**

You might need to increase the Timer Task stack size to accommodate the notification function (see **\_timer\_create\_component()**).

## 2.1.319 \_timer\_start\_periodic\_every ...

	Start a periodic timer every number of:
<code>_timer_start_periodic_every()</code>	Milliseconds
<code>_timer_start_periodic_every_ticks()</code>	Ticks (in tick time)

### Prototype

```
source\kernel\timer.c
#include <timer.h>
_timer_id _timer_start_periodic_every(
    TIMER_NOTIFICATION_TIME_FPTR notification_function,
    void *notification_data_ptr,
    _mqx_uint mode,
    uint32_t ms_wait)

#include <timer.h>
_timer_id _timer_start_periodic_every_ticks(
    TIMER_NOTIFICATION_TICK_FPTR notification_function,
    void *notification_data_ptr,
    _mqx_uint mode,
    MQX_TICK_STRUCT_PTR tick_time_wait_ptr)
```

### Parameters

*notification\_function* [IN] — Pointer to the notification function that MQX RTOS calls when the timer expires

*notification\_data\_ptr* [IN] — Pointer to the data that MQX RTOS passes to the notification function

*mode* [IN] — Time to use when calculating the time to expire; one of the following:

**TIMER\_ELAPSED\_TIME**

**MODE** (use `_time_get_elapsed()` or `_time_get_elapsed_ticks()`, which are not affected by `_time_set()` or `_time_set_ticks()`)

**TIMER\_KERNEL\_TIME**

**MODE** (use `_time_get()` or `_time_get_ticks()`)

*ms\_wait* [IN] — Milliseconds that MQX RTOS waits before it first calls the notification function and between subsequent calls to the notification function

*tick\_time\_wait\_ptr* [IN] — Ticks (in tick time) that MQX RTOS waits before it first calls the notification function and between subsequent calls to the notification function

### Returns

- Timer ID (success)
- **TIMER\_NULL\_ID** (failure)

### Traits

- Creates the timer component with default values if it was not previously created

- On failure, calls `_task_set_error()` to set the task error code as described for `_timer_start_oneshot_after()`

#### See Also

[\\_timer\\_cancel](#)

[\\_timer\\_start\\_oneshot\\_after ...](#)

[\\_timer\\_start\\_oneshot\\_at ...](#)

[\\_timer\\_start\\_periodic\\_at ...](#)

[\\_time\\_get, \\_time\\_get\\_ticks](#)

[\\_time\\_get\\_elapsed, \\_time\\_get\\_elapsed\\_ticks](#)

[\\_task\\_set\\_error](#)

[\\_timer\\_create\\_component](#)

#### Description

When the timer expires, MQX RTOS calls *notification\_function* with *timer\_id*, notifier data, and the current time.

You might need to increase the Timer Task stack size to accommodate the notification function (see `_timer_create_component()`).

#### Example

See `_timer_create_component()`.

## 2.1.320 `_timer_test`

Tests the timer component.

### Prototype

```
source\kernel\timer.c
#include <timer.h>
_mqx_uint _timer_test(
    void *timer_error_ptr)
```

### Parameters

*timer\_error\_ptr* [IN] — Pointer to the first timer entry that has an error

Error	Description
MQX_CORRUPT_QUEUE	Queue of timers is not valid.
MQX_INVALID_COMPONENT_HANDLE	One of the timer entries in the timer queue is not valid ( <i>timer_error_ptr</i> ).

### Returns

- MQX\_OK
- See errors

### See Also

[\\_timer\\_start\\_oneshot\\_after ...](#)

[\\_timer\\_start\\_oneshot\\_at ...](#)

[\\_timer\\_start\\_periodic\\_at ...](#)

[\\_timer\\_start\\_periodic\\_every ...](#)

[\\_timer\\_cancel](#)



## 2.1.321 `_usr_lwevent_clear`

This function is an equivalent to the [\\_lwevent\\_clear](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _usr_lwevent_clear(
    LWEVENT_STRUCT_PTR event_group_ptr,
    _mqx_uint bit_mask)
```

### Parameters

*event\_group\_ptr* [IN, RO] — Pointer to the event group

*bit\_mask* [IN] — Each set bit represents an event bit to clear

### See Also

[\\_usr\\_lwevent\\_create](#)

[\\_usr\\_lwevent\\_destroy](#)

[\\_usr\\_lwevent\\_set, \\_usr\\_lwevent\\_set\\_auto\\_clear](#)

[\\_usr\\_lwevent\\_wait\\_...](#)

[\\_usr\\_lwevent\\_get\\_signalled](#)

[LWEVENT\\_STRUCT](#)

### Description

See [\\_lwevent\\_clear\(\)](#).

## 2.1.322 `_usr_lwevent_create`

This function is an equivalent to the [\\_lwevent\\_create](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _usr_lwevent_create(
    LWEVENT_STRUCT_PTR lwevent_group_ptr,
    _mqx_uint flags)
```

### Parameters

*lwevent\_group\_ptr* [IN, RO] — Pointer to the lightweight event group to initialize

*flags*[IN] — Creation flag; one of the following:

**LWEVENT\_AUTO\_CLEAR** - all bits in the lightweight event group are made autoclearing

**0** - lightweight event bits are not set as autoclearing by default

*note:* the autoclearing bits can be changed any time later by calling

[\\_usr\\_lwevent\\_set\\_auto\\_clear](#).

### See Also

[\\_usr\\_lwevent\\_destroy](#)

[\\_usr\\_lwevent\\_set](#), [\\_usr\\_lwevent\\_set\\_auto\\_clear](#)

[\\_usr\\_lwevent\\_clear](#)

[\\_usr\\_lwevent\\_wait\\_ ...](#)

[\\_usr\\_lwevent\\_get\\_signalled](#)

**LWEVENT\_STRUCT**

### Description:

See [\\_lwevent\\_create\(\)](#).

## 2.1.323 `_usr_lwevent_destroy`

This function is an equivalent to the [\\_lwevent\\_destroy](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _usr_lwevent_destroy(
    LWEVENT_STRUCT_PTR lwevent_group_ptr)
```

### Parameters

*lwevent\_group\_ptr* [IN, RO] — Pointer to the event group to deinitialize

### See Also

[\\_usr\\_lwevent\\_create](#)

[\\_usr\\_lwevent\\_set, \\_usr\\_lwevent\\_set\\_auto\\_clear](#)

[\\_usr\\_lwevent\\_clear](#)

[\\_usr\\_lwevent\\_wait\\_ ...](#)

[\\_usr\\_lwevent\\_get\\_signalled](#)

[LWEVENT\\_STRUCT](#)

### Description

See [\\_lwevent\\_destroy\(\)](#).

## 2.1.324 [\\_usr\\_lwevent\\_get\\_signalled](#)

This function is an equivalent to the [\\_lwevent\\_get\\_signalled](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _usr_lwevent_get_signalled(void)
```

### Parameters

None

### See Also

[\\_usr\\_lwevent\\_create](#)

[\\_usr\\_lwevent\\_destroy](#)

[\\_usr\\_lwevent\\_set](#), [\\_usr\\_lwevent\\_set\\_auto\\_clear](#)

[\\_usr\\_lwevent\\_clear](#)

[\\_usr\\_lwevent\\_wait\\_ ...](#)

[LWEVENT\\_STRUCT](#)

### Description

See [\\_lwevent\\_get\\_signalled\(\)](#).

## 2.1.325 `_usr_lwevent_set`

This function is an equivalent to the `_lwevent_set` API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _usr_lwevent_set(
    LWEVENT_STRUCT_PTR lwevent_group_ptr,
    _mqx_uint flags)
```

### Parameters

*lwevent\_group\_ptr* [IN, RO] — Pointer to the lightweight event group to set bits in

*flags* [IN] — Each bit represents an event bit to be set

### See Also

[\\_usr\\_lwevent\\_create](#)

[\\_usr\\_lwevent\\_destroy](#)

[\\_usr\\_lwevent\\_set\\_auto\\_clear](#)

[\\_usr\\_lwevent\\_clear](#)

[\\_usr\\_lwevent\\_wait\\_ ...](#)

[\\_usr\\_lwevent\\_get\\_signalled](#)

[LWEVENT\\_STRUCT](#)

### Description:

See `_lwevent_set()`.

## 2.1.326 `_usr_lwevent_set_auto_clear`

This function is an equivalent to the [\\_lwevent\\_set\\_auto\\_clear](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _usr_lwevent_set_auto_clear(
    LWEVENT_STRUCT_PTR  lwevent_group_ptr,
    _mqx_uint            auto_mask)
```

### Parameters

*lwevent\_group\_ptr* [IN, RO] — Pointer to the lightweight event group to set bits in  
*auto\_mask* [IN] — Mask of events, which become auto-clear (if corresponding bit of mask is set) or manual-clear (if corresponding bit of mask is clear)

### See Also

[\\_usr\\_lwevent\\_create](#)

[\\_usr\\_lwevent\\_destroy](#)

[\\_usr\\_lwevent\\_set](#)

[\\_usr\\_lwevent\\_clear](#)

[\\_usr\\_lwevent\\_wait\\_ ...](#)

[\\_usr\\_lwevent\\_get\\_signalled](#)

[LWEVENT\\_STRUCT](#)

### Description:

See [\\_lwevent\\_set\\_auto\\_clear\(\)](#).

## 2.1.327 `_usr_lwevent_wait_ ...`

This function is an equivalent to the `_lwevent_wait_ ...` API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

	Wait for the specified lightweight event bits to be set in the lightweight event group:
<code>_usr_lwevent_wait_for()</code>	For the number of ticks (in tick time)
<code>_usr_lwevent_wait_ticks()</code>	For the number of ticks
<code>_usr_lwevent_wait_until()</code>	Until the specified time (in tick time)

### Prototype

```
source\kernel\lwevent.c
#include <lwevent.h>
_mqx_uint _usr_lwevent_wait_for(
    LWEVENT_STRUCT_PTR    event_ptr,
    _mx_uint               bit_mask,
    bool                   all,
    MQX_TICK_STRUCT_PTR    tick_ptr)

_mqx_uint _usr_lwevent_wait_ticks(
    LWEVENT_STRUCT_PTR    event_ptr,
    _mx_uint               bit_mask,
    bool                   all,
    _mx_uint               timeout_in_ticks)

_mqx_uint _usr_lwevent_wait_until(
    LWEVENT_STRUCT_PTR    event_ptr,
    _mx_uint               bit_mask,
    bool                   all,
    MQX_TICK_STRUCT_PTR    tick_ptr)
```

### Parameters

*event\_ptr* [IN, RO] — Pointer to the lightweight event

*bit\_mask* [IN] — Each set bit represents an event bit to wait for

*all* — One of the following:

- TRUE (wait for all bits in *bit\_mask* to be set)
- FALSE (wait for any bit in *bit\_mask* to be set)

*tick\_ptr* [IN] — One of the following:

- pointer to the maximum number of ticks to wait
- NULL (unlimited wait)

*timeout\_in\_ticks* [IN] — One of the following:

- maximum number of ticks to wait

0 (unlimited wait)

**See Also**

[\\_usr\\_lwevent\\_create](#)

[\\_usr\\_lwevent\\_destroy](#)

[\\_usr\\_lwevent\\_set](#), [\\_usr\\_lwevent\\_set\\_auto\\_clear](#)

[\\_usr\\_lwevent\\_clear](#)

[\\_usr\\_lwevent\\_get\\_signalled](#)

[LWEVENT\\_STRUCT](#)

[MQX\\_TICK\\_STRUCT](#)

**Description:**

See [\\_lwevent\\_wait\\_ ...\(\)](#).



## 2.1.328 `_usr_lwmem_alloc`

This function is an equivalent to the [\\_lwmem\\_alloc ...](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwmem.c  
void *_usr_lwmem_alloc(  
    _mem_size size)
```

### Parameter

*size [IN]* — Number of single-addressable units to allocate

### See Also

[\\_usr\\_lwmem\\_alloc\\_from](#)

[\\_usr\\_lwmem\\_create\\_pool](#)

[\\_usr\\_lwmem\\_free](#)

### Description

See [\\_lwmem\\_alloc ...\(\)](#).

## 2.1.329 `_usr_lwmem_alloc_from`

This function is an equivalent to the `_lwmem_alloc_from` API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwmem.c
void *_usr_lwmem_alloc_from(
    _lwmem_pool_id  pool_id
    _mem_size       size)
```

### Parameters

*pool\_id* [IN, RW] — Lightweight-memory pool from which to allocate the lightweight-memory block (pool created with [\\_usr\\_lwmem\\_create\\_pool](#) or ordinary lightweight memory pool for which the user-mode access has been enabled by calling [\\_watchdog\\_create\\_component](#))

*size* [IN] — Number of single-addressable units to allocate

### See Also

[\\_usr\\_lwmem\\_alloc](#)

[\\_usr\\_lwmem\\_create\\_pool](#)

[\\_usr\\_lwmem\\_free](#)

### Description

See [\\_lwmem\\_alloc\\_\\*\\_from\(\)](#).

## 2.1.330 `_usr_lwmem_create_pool`

This function is an equivalent to the [\\_lwmem\\_create\\_pool](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwmem.c
_lwmem_pool_id _usr_lwmem_create_pool(
    LWMEM_POOL_STRUCT_PTR  mem_pool_ptr,
    void                   *start,
    _mem_size              size)
```

### Parameters

*mem\_pool\_ptr* [IN, RW] — Pointer to the definition of the pool

*start* [IN] — Start of the memory for the pool

*size* [IN] — Number of single-addressable units in the pool

### See Also

[\\_usr\\_lwmem\\_alloc](#)

[\\_usr\\_lwmem\\_alloc\\_from](#)

[\\_usr\\_lwmem\\_free](#)

### Description

See [\\_lwmem\\_create\\_pool\(\)](#).

## 2.1.331 [\\_usr\\_lwmem\\_free](#)

This function is an equivalent to the [\\_lwmem\\_free](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwmem.c
_mqx_uint _usr_lwmem_free(
    void *mem_ptr)
```

### Parameters

*mem\_ptr* [IN, RW] — Pointer to the block to free

### See Also

[\\_usr\\_lwmem\\_alloc](#)

[\\_usr\\_lwmem\\_alloc\\_from](#)

[\\_usr\\_lwmem\\_create\\_pool](#)

### Description

See [\\_lwmem\\_free\(\)](#).

### 2.1.332 `_usr_lwmsgq_init`

This function is an equivalent to the [\\_lwmsgq\\_deinit](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

#### Prototype

```
source\kernel\lwmsgq.c
#include <lwmsgq.h>
#include <lwmsgq_prv.h>
_mqx_uint _usr_lwmsgq_init(
void *location,
_mqx_uint num_messages,
_mqx_uint msg_size)
```

#### Parameters

*location [IN]* — Pointer to memory to create a message queue.

*num\_message [IN]* — Number of messages in the queue.

*msg\_size [IN]* — Specifies message size as a multiplier factor of `_mqx_max_type` items.

#### See also

[\\_usr\\_lwmsgq\\_receive](#)

[\\_usr\\_lwmsgq\\_send](#)

#### Description

See [\\_lwmsgq\\_deinit\(\)](#).

## 2.1.333 `_usr_lwmsgq_receive`

This function is an equivalent to the [\\_lwmsgq\\_receive](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwmsgq.c
#include <lwmsgq.h>
#include <lwmsgq_prv.h>
_mqx_uint _usr_lwmsgq_receive(
void *handle,
_mqx_max_type_ptr message,
_mqx_uint flags,
_mqx_uint ticks,
MQX_TICK_STRUCT_PTR tick_ptr)
```

### Parameters

- handle* [IN] — Pointer to the message queue created by `_lwmsgq_init`
- message* [OUT] — Received message
- flags* [IN] — LWMSGQ\_RECEIVE\_BLOCK\_ON\_EMPTY (block the reading task if msgq is empty), LWMSGQ\_TIMEOUT\_UNTIL (perform a timeout using the tick structure as the absolute time), LWMSGQ\_TIMEOUT\_FOR (perform a timeout using the tick structure as the relative time)
- ticks* [IN] — The maximum number of ticks to wait or NULL (unlimited wait).
- tick\_ptr* [IN] — Pointer to the tick structure to use.

### See also

[\\_usr\\_lwmsgq\\_init](#)

[\\_usr\\_lwmsgq\\_send](#)

### Description

See [\\_lwmsgq\\_receive\(\)](#).

## 2.1.334 `_usr_lwmsgq_send`

This function is an equivalent to the [\\_lwmsgq\\_send](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwmsgq.c
#include <lwmsgq.h>
#include <lwmsgq_prv.h>
_mqx_uint _usr_lwmsgq_send(
void *handle,
_mqx_max_type_ptr message,
_mqx_uint flags)
```

### Parameters

*handle* [IN] — Pointer to the message queue created by `_lwmsgq_init`

*message* [IN] — Pointer to the message to send.

*flags* [IN] — LWMSGQ\_SEND\_BLOCK\_ON\_FULL — Block the task if queue is full.

LWMSGQ\_SEND\_BLOCK\_ON\_SEND — Block the task after the message is sent.

### See also

[\\_usr\\_lwmsgq\\_init](#)

[\\_usr\\_lwmsgq\\_receive](#)

### Description

See [\\_lwmsgq\\_send\(\)](#).

## 2.1.335 [\\_usr\\_lwsem\\_create](#)

This function is an equivalent to the [\\_lwsem\\_create](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwmem.c
_mqx_uint _usr_lwsem_create(
    LWSEM_STRUCT_PTR  lwsem_ptr,
    _mqx_int           initial_count)
```

### Parameters

*lwsem\_ptr* [IN, RO] — Pointer to the lightweight semaphore to create

*initial\_count* [IN] — Initial semaphore counter

### See Also

[\\_usr\\_lwsem\\_destroy](#)

[\\_usr\\_lwsem\\_post](#)

[\\_usr\\_lwsem\\_wait ...](#)

### Description

See [\\_lwsem\\_create\(\)](#).



## 2.1.336 [\\_usr\\_lwsem\\_destroy](#)

This function is an equivalent to the [\\_lwsem\\_destroy](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwsem.c
_mqx_uint _usr_lwsem_destroy(
    LWSEM_STRUCT_PTR  lwsem_ptr)
```

### Parameters

*lwsem\_ptr* [IN, RO] — Pointer to the created lightweight semaphore

### See Also

[\\_usr\\_lwsem\\_create](#)

### Description

See [\\_lwsem\\_destroy\(\)](#).

## 2.1.337 [\\_usr\\_lwsem\\_poll](#)

This function is an equivalent to the [\\_lwsem\\_poll](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwsem.c  
bool _usr_lwsem_poll(  
    LWSEM_STRUCT_PTR  lwsem_ptr)
```

### Parameters

*lwsem\_ptr* [IN, RO] — Pointer to the created lightweight semaphore

### See Also

[\\_usr\\_lwsem\\_create](#)

[\\_usr\\_lwsem\\_wait ...](#) family

### Description

See [\\_lwsem\\_poll\(\)](#).

## 2.1.338 `_usr_lwsem_post`

This function is an equivalent to the [\\_lwsem\\_post](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\lwsem.c
_mqx_uint _usr_lwsem_post(
    LWSEM_STRUCT_PTR lwsem_ptr)
```

### Parameters

*lwsem\_ptr* [IN, RO] — Pointer to the created lightweight semaphore

### See Also

[\\_usr\\_lwsem\\_create](#)

[\\_usr\\_lwsem\\_wait ...](#)

### Description

See [\\_lwsem\\_post\(\)](#).

## 2.1.339 \_usr\_lwsem\_wait ...

These functions are equivalents to [\\_lwsem\\_wait ...](#) API calls but they can be executed from within the User task or other code running in the CPU User mode. Parameters passed to these functions by pointer are required to meet the memory protection requirements as described in the parameter list below.

	Wait (in FIFO order) for the lightweight semaphore:
<code>_usr_lwsem_wait()</code>	Until it is available
<code>_usr_lwsem_wait_for()</code>	For the number of ticks (in tick time)
<code>_usr_lwsem_wait_ticks()</code>	For the number of ticks
<code>_usr_lwsem_wait_until()</code>	Until the specified time (in tick time)

### Prototype

```
source\kernel\lwsem.c
#include <lwsem.h>
_mqx_uint _usr_lwsem_wait(
    LWSEM_STRUCT_PTR sem_ptr)

_mqx_uint _usr_lwsem_wait_for(
    LWSEM_STRUCT_PTR sem_ptr,
    MQX_TICK_STRUCT_PTR tick_time_timeout_ptr)

_mqx_uint _usr_lwsem_wait_ticks(
    LWSEM_STRUCT_PTR sem_ptr,
    _mqx_uint tick_timeout)

_mqx_uint _usr_lwsem_wait_until(
    LWSEM_STRUCT_PTR sem_ptr,
    MQX_TICK_STRUCT_PTR tick_time_ptr)
```

### Parameters

*sem\_ptr* [IN, RO] — Pointer to the lightweight semaphore

*tick\_time\_timeout\_ptr* [IN, RW] — One of the following:

pointer to the maximum number of ticks to wait

NULL (unlimited wait)

*tick\_timeout* [IN] — One of the following:

maximum number of ticks to wait

0 (unlimited wait)

*tick\_time\_ptr* [IN, RW] — One of the following:

pointer to the time (in tick time) until which to wait

NULL (unlimited wait)

### See Also

[\\_usr\\_lwsem\\_create](#)

[\\_usr\\_lwsem\\_post](#)

[LWSEM\\_STRUCT](#)

[MQX\\_TICK\\_STRUCT](#)

### Description

See [\\_lwsem\\_wait ...\(\)](#).

## 2.1.340 `_usr_task_abort`

This function is an equivalent to the [\\_task\\_abort](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\task.c
_mqx_uint _usr_task_abort(
    _task_id task_id)
```

### Parameters

*task\_id* [IN] — One of the following:  
task ID of the task to be destroyed  
**MQX\_NULL\_TASK\_ID** (abort the calling task)

### See Also

[\\_usr\\_task\\_destroy](#)

### Description

See [\\_task\\_abort\(\)](#).

## 2.1.341 `_usr_task_create`

This function is an equivalent to the `_task_create` API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\task.c
_task_id _usr_task_create(
    _processor_number processor_number,
    _mqx_uint         template_index,
    uint32_t           parameter)
```

### Parameters

*processor\_number* [IN] — One of the following:

processor number of the processor where the task is to be created  
0 (create on the local processor)

*template\_index* [IN] — One of the following:

index of the task template in the processor's task template list to use for the child task  
0 (use the task template that *create\_parameter* defines)

*parameter* [IN]

*template\_index* is not 0 — pointer to the parameter that MQX RTOS passes to the child task  
*template\_index* is 0 — pointer to the task template

### See Also

[`\_usr\_task\_abort`](#)

[`\_usr\_task\_destroy`](#)

[`\_usr\_task\_ready`](#)

[`\_usr\_task\_set\_error`](#)

[`MQX\_INITIALIZATION\_STRUCT`](#)

[`TASK\_TEMPLATE\_STRUCT`](#)

### Description

See [`\_task\_create`](#), [`\_task\_create\_blocked`](#), [`\_task\_create\_at`](#), [`create\_task`](#).

## 2.1.342 `_usr_task_destroy`

This function is an equivalent to the [\\_task\\_destroy](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\task.c
_mqx_uint _usr_task_destroy(
    _task_id task_id)
```

### Parameters

*task\_id* [IN] — One of the following:

task ID of the task to be destroyed

**MQX\_NULL\_TASK\_ID** (destroy the calling task)

### See Also

[\\_usr\\_task\\_create](#)

[\\_usr\\_task\\_abort](#)

### Description

See [\\_task\\_destroy\(\)](#).



### 2.1.343 `_usr_task_get_td`

This function is an equivalent to the [\\_task\\_get\\_td](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

#### Prototype

```
source\kernel\task.c  
void *_usr_task_get_td(  
    _task_id task_id)
```

#### Parameters

*task\_id* [IN] — One of:  
task ID for a task on this processor  
**MQX\_NULL\_TASK\_ID** (use the current task)

#### See also

[\\_usr\\_task\\_ready](#)

#### Description

See [\\_task\\_get\\_td\(\)](#).

## 2.1.344 `_usr_task_ready`

This function is an equivalent to the `_task_ready` API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\task.c  
void _usr_task_ready(  
    void *td_ptr)
```

### Parameters

*td\_ptr* [IN] — Pointer to the task descriptor of the task (on this processor) to be made ready

### Description

See `_task_ready`.

### 2.1.345 `_usr_task_set_error`

This function is an equivalent to the [\\_task\\_set\\_error](#) API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

#### Prototype

```
source\kernel\task.c
_mqx_uint _usr_task_set_error(
    _mx_uint error_code)
```

#### Parameters

*error\_code* [IN] — Task error code

#### Description

See [\\_task\\_set\\_error\(\)](#).

## 2.1.346 `_usr_time_delay ...`

These functions are equivalents to [\\_time\\_delay ...](#) API calls but they can be executed from within the User task or other code running in the CPU User mode. Parameters passed to these functions by pointer are required to meet the memory protection requirements as described in the parameter list below.

Suspend the active task:	
<code>_usr_time_delay()</code>	For the number of milliseconds
<code>_usr_time_delay_ticks()</code>	For the number of ticks

### Prototype

```
source\kernel\time.c
void _usr_time_delay(
    uint32_t ms_delay)

void _usr_time_delay_ticks(
    _mqx_uint tick_delay)
```

### Parameters

*ms\_delay* [IN] — Minimum number of milliseconds to suspend the task

*tick\_delay* [IN] — Minimum number of ticks to suspend the task

### See Also

[\\_usr\\_time\\_get\\_elapsed\\_ticks](#)

### Description

See [\\_time\\_delay ...\(\)](#).

## 2.1.347 `_usr_time_get_elapsed_ticks`

This function is an equivalent to the `_time_get_elapsed_ticks` API call but it can be executed from within the User task or other code running in the CPU User mode. Parameters passed to this function by pointer are required to meet the memory protection requirements as described in the parameter list below.

### Prototype

```
source\kernel\time.c  
void _usr_time_get_elapsed_ticks(  
    MQX_TICK_STRUCT_PTR tick_time_ptr)
```

### Parameters

*timetick\_time\_ptr* [OUT, RW] — Where to store the elapsed tick time

### See Also

[\\_usr\\_time\\_delay ...](#)

[MQX\\_TICK\\_STRUCT](#)

### Description

See [\\_time\\_get\\_elapsed](#), [\\_time\\_get\\_elapsed\\_ticks\(\)](#).

## 2.1.348 `_watchdog_create_component`

Creates the watchdog component.

### Prototype

```
source\kernel\watchdog.c
#include <watchdog.h>
_mqx_uint _watchdog_create_component(
    _mx_uint timer_interrupt_vector,
    WATCHDOG_ERROR_FPTR expiry_function)
```

### Parameters

*timer\_interrupt\_vector* [IN] — Periodic timer interrupt vector number  
*expiry\_function* [IN] — Function that MQX RTOS calls when a watchdog expires

### Returns

- `MQX_OK` (success: see description)
- Errors (failure)

Errors	Description
<code>MQX_OUT_OF_MEMORY</code>	MQX RTOS cannot allocate memory for watchdog component data.
<code>WATCHDOG_INVALID_ERROR_FUNCTION</code>	<i>expiry_function</i> is <code>NULL</code> .
<code>WATCHDOG_INVALID_INTERRUPT_VECTOR</code>	MQX RTOS cannot install the periodic timer interrupt vector.

### See Also

[\\_watchdog\\_start, \\_watchdog\\_start\\_ticks](#)  
[\\_watchdog\\_stop](#)

### Description

An application must explicitly create the watchdog component before tasks can use watchdogs.

The function returns **MQX\_OK** if either:

- watchdog component is created
- watchdog component was previously created and the configuration is not changed

### Example

```
_mx_uint result;
extern void task_watchdog_error(TD_STRUCT_PTR td_ptr);
...
```

```
/* Create watchdog component. */  
result = _watchdog_create_component(TIMER_INTERRUPT_VECTOR,  
    task_watchdog_error);  
if (result != MQX_OK) {  
    /* An error occurred. */  
}
```

## 2.1.349 `_watchdog_start`, `_watchdog_start_ticks`

Starts or restart the watchdog.

### Prototype

```
source\kernel\watchdog.c
#include <watchdog.h>
bool _watchdog_start(
    uint32_t ms_time)

bool _watchdog_start_ticks(
    MQX_TICK_STRUCT_PTR tick_time_ptr)
```

### Parameters

*ms\_time* [IN] — Milliseconds until the watchdog expires

*tick\_time\_ptr* [IN] — Pointer to the number of ticks until the watchdog expires

### Returns

- TRUE (success)
- FALSE (failure: see description)

### See also

[\\_time\\_to\\_ticks](#)

[\\_usr\\_lwevent\\_clear](#)

[\\_watchdog\\_stop](#)

[MQX\\_TICK\\_STRUCT](#)

### Description

The function returns *FALSE* if either of these conditions is true:

- watchdog component was not previously created
- watchdog component data is no longer valid

### Example

```
while (1) {
    _watchdog_stop();
    msg_ptr = _msgq_receive(MSGQ_ANY_QUEUE, 0);
    /* Start the watchdog to expire in 2 seconds, in case we
    ** don't finish in that time.
    */
    _watchdog_start(2000);
    ...
    /* Do the work. */
    ...
}
```



## 2.1.350 `_watchdog_stop`

Stops the watchdog.

### Prototype

```
source\kernel\watchdog.c  
#include <watchdog.h>  
bool _watchdog_stop(void)
```

### Parameters

None

### Returns

- TRUE (success)
- FALSE (failure: see description)

### See also

[\\_usr\\_lwevent\\_clear](#)

[\\_watchdog\\_start](#), [\\_watchdog\\_start\\_ticks](#)

### Description

The function returns *FALSE* if any of these conditions is true:

- watchdog component was not previously created
- watchdog component data is no longer valid
- watchdog was not started

### Example

See `_usr_lwevent_clear()`.

## 2.1.351 `_watchdog_test`

Tests the watchdog component data.

### Prototype

```
source\kernel\watchdog.c
#include <watchdog.h>
_mqx_uint _watchdog_test(
    void *watchdog_error_ptr,
    void *watchdog_table_error_ptr)
```

### Parameters

*watchdog\_error\_ptr* [OUT] — Pointer to the watchdog component base that has an error (*NULL* if no errors are found)

*watchdog\_table\_error\_ptr* [OUT] — Pointer to the watchdog table that has an error (always *NULL*)

### Returns

- `MQX_OK` (see description)
- `MQX_INVALID_COMPONENT_BASE` (an error was found)

### See Also

[\\_usr\\_lwevent\\_clear](#)

[\\_watchdog\\_start, \\_watchdog\\_start\\_ticks](#)

[\\_watchdog\\_stop](#)

### Description

The function returns **MQX\_OK** if either:

- it did not find an error in watchdog component data
- watchdog component was not previously created

### Example

```
void *watchdog_error;
void *watchdog_table_error;
...
if (_watchdog_test(&watchdog_error, &watchdog_table_error) != MQX_OK) {
    /* Watchdog component is corrupted. */
}
```

## 2.1.352 MSG\_MUST\_CONVERT\_DATA\_ENDIAN

Determines whether the data portion of the message needs to be converted to the other endian format.

### Prototype

```
source\include\message.h  
bool MSG_MUST_CONVERT_DATA_ENDIAN(  
    unsigned char  endian_format)
```

### Parameters

*endian\_format* [IN] — Endian format of the message

### Returns

- TRUE
- FALSE

### See Also

[\\_mem\\_swap\\_endian](#)

[\\_msg\\_swap\\_endian\\_data](#)

[MSG\\_MUST\\_CONVERT\\_HDR\\_ENDIAN](#)

[MESSAGE\\_HEADER\\_STRUCT](#)

### Example

See `_msg_swap_endian_data()`.

### 2.1.353 MSG\_MUST\_CONVERT\_HDR\_ENDIAN

Determines whether the header portion of the message needs to be converted to the other endian format.

#### Prototype

```
source\include\message.h  
bool MSG_MUST_CONVERT_HDR_ENDIAN(  
    unsigned char endian_format)
```

#### Parameters

*endian\_format* [IN] — Endian format of the message

#### Returns

- TRUE
- FALSE

#### See Also

[\\_mem\\_swap\\_endian](#)

[\\_msg\\_swap\\_endian\\_header](#)

[\\_msg\\_swap\\_endian\\_data](#)

[MSG\\_MUST\\_CONVERT\\_DATA\\_ENDIAN](#)

[MESSAGE\\_HEADER\\_STRUCT](#)

#### Example

See `_msg_swap_endian_header()`.

## Chapter 3 MQX Data Types

### 3.1 Data Types Overview

Table 3-1. Data Types for Compiler Portability

Data type	Size	Description
<code>_mqx_int</code>	See note 1	See note 1
<code>_mqx_uint</code>	See note 1	See note 1
<code>_mqx_max_type</code>		Largest type available (e.g., on a 32-bit processor, <code>_mqx_max_type</code> is defined as <code>uint32_t</code> )
<code>_mqx_max_type_ptr</code>	See note 3	Pointer to <code>_mqx_max_type</code>
<code>_mem_size</code>	See note 2	See note 2
<code>_mem_size_ptr</code>	See note 3	Pointer to <code>_mem_size</code>
<code>_psp_code_addr</code>	Large enough to hold the address of a code location	
<code>_psp_code_addr_ptr</code>	See note 3	Pointer to <code>_psp_code_addr</code>
<code>_psp_data_addr</code>	Large enough to hold the address of a data location	
<code>_psp_data_addr_ptr</code>	See note 3	Pointer to <code>_psp_data_addr</code>
<code>_file_size</code>	<code>uint32_t</code>	Number of bytes in a file
<code>_file_offset</code>	<code>int32_t</code>	Maximum offset (in bytes) in a file
<code>ieee_single</code>	32 bits	Single-precision IEEE floating-point number
<code>ieee_double</code>	32 or 64 bits depending on the compiler	Double-precision IEEE floating-point number

<sup>1</sup> `_mqx_int`, `_mqx_uint`: MQX determines the size of `_mqx_int` and `_mqx_uint` from the natural size of the processor. They are defined in `psptypes.h` for the PSP. For example, on a 16-bit processor, `_mqx_uint` (`_mqx_int`) is defined as `uint16_t` (`int16_t`). On a 32-bit processor, `_mqx_uint` (`_mqx_int`) is defined as `uint32_t` (`int32_t`).

- <sup>2</sup> **\_mem\_size**: MQX RTOS equates **\_mem\_size** to the type that can hold the maximum data address for the processor. It is defined in *psptypes.h* for the PSP.
- <sup>3</sup> **\*\_ptr** are large enough to hold a data address (**\_mem\_size**).

**Table 3-2. MQX RTOS Simple Data Types**

Name	Data type	Defined in
—	<b>_CODE_PTR_</b>	<i>psptypes.h</i> for the PSP
<b>_lwmem_pool_id</b>	<b>void*</b>	lwmem.h
<b>_mem_pool_id</b>	<b>void*</b>	mqx.h
<b>_msg_size</b>	<b>uint16_t</b>	message.h
<b>_partition_id</b>	<b>void*</b>	part.h
<b>_pool_id</b>	<b>void*</b>	message.h
<b>_processor_number</b>	<b>uint16_t</b>	mqx.h
<b>_queue_id</b>	<b>uint16_t or uint32_t</b>	message.h
<b>_queue_number</b>	<b>uint16_t or uint32_t</b>	message.h
<b>_task_id</b>	<b>uint32_t</b>	mqx.h
<b>_timer_id</b>	<b>_mqx_uint</b>	timer.h

Starting with MQX RTOS 4.1.0, legacy MQX RTOS custom integer types were replaced by the Standard C99 set (int\_32 -> int32\_t, boolean -> bool, etc). The *psptypes\_legacy.h* header file is provided with the set of backward compatible type definitions to make the transition to the new types easier.

## 3.2 MQX RTOS Complex Data Types in Alphabetical Order

### 3.2.1 DATE\_STRUCT

Date structure for time.

#### Prototype

```
#include <mqx.h>
typedef
{
    int16_t YEAR;
    int16_t MONTH;
    int16_t DAY;
    int16_t HOUR;
    int16_t MINUTE;
    int16_t SECOND;
    int16_t MILLISEC;
    int_16 WDAY;
    int_16 YDAY;
} DATE_STRUCT, * DATE_STRUCT_PTR;
```

#### See Also

[\\_time\\_from\\_date](#)

[\\_time\\_get, \\_time\\_get\\_ticks](#)

[\\_time\\_set, \\_time\\_set\\_ticks](#)

[\\_time\\_to\\_date](#)

#### TIME\_STRUCT

Field	Range	
	From	To
YEAR	1970	2099
MONTH	1	12
DAY	1	28, 29, 30, 31 (depending on the month)
HOUR	0	23
MINUTE	0	59
SECOND	0	59
MILLISEC	0	999
WDAY	0	6
YDAY	0	365

### CAUTION

If you violate the ranges, undefined behavior results.

#### Example

See `_time_from_date()`.



### 3.2.2 IPC\_PCB\_INIT\_STRUCT

Initialization structure for IPCs over PCB devices.

#### Prototype

```
#include <mqx.h>
#include <ipc.h>
#include <ipc_pcb.h>
typedef struct ipc_pcb_init_struct {
    char          *IO_PCB_DEVICE_NAME;
    IPC_PCB_DEVINSTALL_FPTR DEVICE_INSTALL;
    void          *DEVICE_INSTALL_PARAMETER;
    uint16_t      IN_MESSAGES_MAX_SIZE;
    uint16_t      IN_MESSAGES_TO_ALLOCATE;
    uint16_t      IN_MESSAGES_TO_GROW;
    uint16_t      IN_MESSAGES_MAX_ALLOCATE;
    uint16_t      OUT_PCBS_INITIAL;
    uint16_t      OUT_PCBS_TO_GROW;
    uint16_t      OUT_PCBS_MAX;
} IPC_PCB_INIT_STRUCT, * IPC_PCB_INIT_STRUCT_PTR;
```

#### See Also

[\\_ipc\\_pcb\\_init](#)

#### Fields

Field	Description
<b>IO_PCB_DEVICE_NAME</b>	String name of the PCB device driver to be opened by the IPC.
<b>DEVICE_INSTALL</b>	Function to call to install the PCB device (if required)
<b>DEVICE_INSTALL_PARAMETER</b>	Parameter to pass to the installation function.
<b>IN_MESSAGES_MAX_SIZE</b>	Maximum size of all messages arriving at the IPC.
<b>IN_MESSAGES_TO_ALLOCATE</b>	Initial number of input messages to allocate.
<b>IN_MESSAGES_TO_GROW</b>	Number of input messages to add to the pool when messages are all in use.
<b>IN_MESSAGES_MAX_ALLOCATE</b>	Maximum number of messages in the input message pool.
<b>OUT_PCBS_INITIAL</b>	Initial number of PCBs in the output PCB pool.
<b>OUT_PCBS_TO_GROW</b>	Number of PCBs to add to the output PCB pool when all the PCBs are in use.
<b>OUT_PCBS_MAX</b>	Maximum number of PCBs in the output PCB pool.

### 3.2.3 IPC\_PROTOCOL\_INIT\_STRUCT

IPC initialization information.

#### Prototype

```
source\ipc\ipc.h
typedef struct ipc_protocol_init_struct
{
    IPC_INIT_FPTR  IPC_PROTOCOL_INIT;
    void           *IPC_PROTOCOL_INIT_DATA;
    char           *IPC_NAME;
    _queue_number  IPC_OUT_QUEUE;
} IPC_PROTOCOL_INIT_STRUCT, * IPC_PROTOCOL_INIT_STRUCT_PTR;
```

#### See Also

[IPC\\_ROUTING\\_STRUCT](#)

[IPC\\_INIT\\_STRUCT](#)

#### Description

The *\_ipc\_init\_table[]* (an array of entries of type **IPC\_PROTOCOL\_INIT\_STRUCT**) defines the communication paths between processors (IPCs). The table is terminated by a zero-filled entry.

#### Fields

Field	Description
<b>IPC_PROTOCOL_INIT</b>	Function that initializes the IPC. The function depends on the IPC.
<b>IPC_PROTOCOL_INIT_DATA</b>	Pointer to the initialization data that is specific to the IPC protocol. The format of the data depends on the IPC.
<b>IPC_NAME</b>	String name that identifies the IPC.
<b>IPC_OUT_QUEUE</b>	Queue number of the output queue to which MQX RTOS routes messages that are to be sent to the remote processor. The queue number must match a queue number that is in the IPC routing table.

### 3.2.4 IPC\_ROUTING\_STRUCT

Entry in the IPC routing table for interprocessor communication.

#### Prototype

```
source\ipc\ipc.h
typedef struct ipc_routing_struct
{
    _processor_number    MIN_PROC_NUMBER;
    _processor_number    MAX_PROC_NUMBER;
    _queue_number        QUEUE;
} IPC_ROUTING_STRUCT, * IPC_ROUTING_STRUCT_PTR;
```

#### See Also

[IPC\\_PROTOCOL\\_INIT\\_STRUCT](#)

[IPC\\_INIT\\_STRUCT](#)

#### Description

Defines an entry in the table *\_ipc\_routing\_table[]*, which has an entry for each remote processor that the processor communicates with. The table is terminated with a zero-filled entry.

#### Fields

Field	Description
<b>MIN_PROC_NUMBER</b> <b>MAX_PROC_NUMBER</b>	Range of processors that can be accessed from the communication path. In most cases, the values are equal, indicating that the end of the communication is occupied by one processor. In some cases, the processor at the end of the path is connected to other processors, in which case the processor might also act as a gateway.
<b>QUEUE</b>	Queue number of the IPC output queue.

### 3.2.5 IPC\_INIT\_STRUCT

IPC initialization structure that is passed to the `_ipc_task` function as a creation parameter.

#### Prototype

```
source\ipc\ipc.h
typedef struct ipc_init_struct
{
    const IPC_ROUTING_STRUCT *    ROUTING_LIST_PTR;
    const IPC_PROTOCOL_INIT_STRUCT *  PROTOCOL_LIST_PTR;
} IPC_INIT_STRUCT, * IPC_INIT_STRUCT_PTR;
```

#### See Also

[IPC\\_PROTOCOL\\_INIT\\_STRUCT](#)

[IPC\\_ROUTING\\_STRUCT](#)

#### Description

This structure allows both user defined IPC routing table and IPC initialization table to be passed to the `_ipc_task`.

#### Fields

Field	Description
<b>ROUTING_LIST_PTR</b>	Pointer to the IPC routing table.
<b>PROTOCOL_LIST_PTR</b>	Pointer to the IPC initialization table.

### 3.2.6 LOG\_ENTRY\_STRUCT

Header of an entry in a user log.

#### Prototype

```
#include <log.h>
typedef struct log_entry_header_struct
{
    _mqx_uint    SIZE;
    _mqx_uint    SEQUENCE_NUMBER;
    uint32_t     SECONDS;
    uint16_t     MILLISECONDS;
    uint16_t     MICROSECONDS;
} LOG_ENTRY_STRUCT, * LOG_ENTRY_STRUCT_PTR;
```

#### See Also

[\\_log\\_read](#)

[\\_log\\_write](#)

#### Description

The length of the entry depends on the **SIZE** field.

#### Fields

Field	Description
<b>SIZE</b>	Number of long words in the entry.
<b>SEQUENCE_NUMBER</b>	Sequence number for the entry.
<b>SECONDS</b> <b>MILLISECONDS</b> <b>MICROSECONDS</b>	Time at which MQX RTOS wrote the entry.

### 3.2.7 LWEVENT\_STRUCT

Lightweight event group.

#### Prototype

```
#include <lwevent.h>
typedef struct lwevent_struct
{
    QUEUE_ELEMENT_STRUCT  LINK;
    QUEUE_STRUCT          WAITING_TASKS;
    _mqx_uint              VALID;
    _mqx_uint              VALUE;
    _mqx_uint              FLAGS;
    _mqx_uint              AUTO;
} LWEVENT_STRUCT, * LWEVENT_STRUCT_PTR;
```

#### See Also

[\\_lwevent\\_clear](#)

[\\_lwevent\\_create](#)

[\\_lwevent\\_destroy](#)

[\\_lwevent\\_set](#)

[\\_lwevent\\_set\\_auto\\_clear](#)

[\\_lwevent\\_wait\\_ ...](#)

#### Fields

Field	Description
<b>LINK</b>	Queue data structures.
<b>WAITING_TASKS</b>	Queue of tasks waiting for event bits to be set.
<b>VALID</b>	Validation stamp.
<b>VALUE</b>	Current bit value of the lightweight event group.
<b>FLAGS</b>	Flags associated with the lightweight event group; currently only LWEVENT_AUTO_CLEAR.
<b>AUTO</b>	Mask specifying lightweight event bits that are configured as auto-clear.

### 3.2.8 LWLOG\_ENTRY\_STRUCT

Entry in kernel log or a lightweight log.

#### Prototype

```
#include <lwlog.h>
typedef struct lwlog_entry_struct
{
    _mqx_uint      SEQUENCE_NUMBER;
    #if MQX_LWLOG_TIME_STAMP_IN_TICKS == 0
        uint32_t    SECONDS;
        uint32_t    MILLISECONDS;
        uint32_t    MICROSECONDS;
    #else
        MQX_RTOS_TICK_STRUCT TIMESTAMP;
    #endif
    _mqx_max_type   DATA[LWLOG_MAXIMUM_DATA_ENTRIES];
    struct lwlog_entry_struct
        *NEXT_PTR;
} LWLOG_ENTRY_STRUCT, * LWLOG_ENTRY_STRUCT_PTR;
```

#### See Also

[\\_lwlog\\_read](#)

[\\_lwlog\\_write](#)

#### Fields

Field	Description
<b>SEQUENCE_NUMBER</b>	The sequence number for the entry.
<b>SECONDS</b> <b>MILLISECONDS</b> <b>MICROSECONDS</b>	The time at which the entry was written if MQX RTOS is not configured at compile time to timestamp in ticks.
<b>TIMESTAMP</b>	The time in tick time at which the entry was written if MQX RTOS is configured at compile time to timestamp in ticks.
<b>DATA</b>	Data for the entry.
<b>NEXT_PTR</b>	Pointer to the next lightweight-log entry.

### 3.2.9 LWSEM\_STRUCT

Lightweight semaphore.

#### Prototype

```
#include <mqx.h>
typedef struct lwsem_struct
{
    struct lwsem_struct      *NEXT;
    struct lwsem_struct      *PREV;
    QUEUE_STRUCT             TD_QUEUE;
    _mqx_uint                VALID;
    _mqx_int                 VALUE;
} LWSEM_STRUCT, * LWSEM_STRUCT_PTR;
```

#### See Also

[\\_lwsem\\_create](#)

#### Fields

Field	Description
<b>NEXT</b>	Pointer to the next lightweight semaphore in the list of lightweight semaphores.
<b>PREV</b>	Pointer to the previous lightweight semaphore in the list of lightweight semaphores.
<b>TD_QUEUE</b>	Manages the queue of tasks that are waiting for the lightweight semaphore. The NEXT and PREV fields in the task descriptors link the tasks.
<b>VALID</b>	When MQX RTOS creates the lightweight semaphore, it initializes the field. When MQX RTOS destroys the lightweight semaphore, it clears the field.
<b>VALUE</b>	Count of the semaphore. MQX RTOS decrements the field when a task waits for the semaphore. If the field is not 0, the task gets the semaphore. If the field is 0, MQX RTOS puts the task in the lightweight semaphore queue until the count is a non-zero value.



### 3.2.10 LWTIMER\_PERIOD\_STRUCT

Lightweight timer queue.

#### Prototype

```
typedef struct lwtimer_period_struct
{
    QUEUE_ELEMENT_STRUCT  LINK;
    _mqx_uint              PERIOD;
    _mqx_uint              EXPIRY;
    _mqx_uint              WAIT;
    QUEUE_STRUCT           TIMERS;
    LWTIMER_STRUCT_PTR     TIMER_PTR;
    _mqx_uint              VALID;
} LWTIMER_PERIOD_STRUCT, * LWTIMER_PERIOD_STRUCT_PTR;
```

#### See Also

#### [LWTIMER\\_STRUCT](#)

#### Description

The structure controls any number of lightweight timers that expire at the same periodic rate as defined by the structure.

#### Fields

Field	Description
<b>LINK</b>	Queue of lightweight timers.
<b>PERIOD</b>	Period (in ticks) of the timer queue; a multiple of BSP_ALARM_RESOLUTION.
<b>EXPIRY</b>	Number of ticks that have elapsed in the period.
<b>WAIT</b>	Number of ticks to wait before starting to process the queue.
<b>TIMERS</b>	Queue of timers to expire at the periodic rate.
<b>TIMER_PTR</b>	Pointer to the last timer that was processed.
<b>VALID</b>	When the timer queue is created, MQX RTOS initializes the field. When the queue is cancelled, MQX RTOS clears the field.

### 3.2.11 LWTIMER\_STRUCT

Lightweight timer.

#### Prototype

```
typedef struct lwtimer_struct
{
    QUEUE_ELEMENT_STRUCT  LINK;
    _mqx_uint              RELATIVE_TICKS;
    _mqx_uint              VALID;
    LWTIMER_ISR_FPTR      TIMER_FUNCTION;
    void                   *PARAMETER;
    void                   *PERIOD_PTR;
} LWTIMER_STRUCT, * LWTIMER_STRUCT_PTR;
```

#### See Also

#### [LWTIMER\\_PERIOD\\_STRUCT](#)

#### Description

With lightweight timers, a timer function is called at a periodic interval.

#### Fields

Field	Description
<b>LINK</b>	Queue data structures.
<b>RELATIVE_TICKS</b>	Relative number of ticks until the timer is to expire.
<b>VALID</b>	When the timer is added to the timer queue, MQX RTOS initializes the field. When the timer or the timer queue that the timer is in is cancelled, MQX RTOS clears the field.
<b>TIMER_FUNCTION</b>	Function that is called when the timer expires.
<b>PARAMETER</b>	Parameter that is passed to the timer function.
<b>PERIOD_PTR</b>	Pointer to the lightweight timer queue to which the timer is attached.

### 3.2.12 MESSAGE\_HEADER\_STRUCT

Message header.

#### Prototype

```
#include <message.h>
typedef struct message_header_struct
{
    _msg_size      SIZE;
#ifdef MQX_USE_32BIT_MESSAGE_QIDS
    uint16_t       PAD;
#endif
    _queue_id      TARGET_QID;
    _queue_id      SOURCE_QID;
    unsigned char   CONTROL;
#ifdef MQX_USE_32BIT_MESSAGE_QIDS
    unsigned char   RESERVED[3];
#else
    unsigned char   RESERVED;
#endif
} MESSAGE_HEADER_STRUCT, * MESSAGE_HEADER_STRUCT_PTR;
```

#### See Also

[\\_msg\\_alloc](#)

[\\_msg\\_alloc\\_system](#)

[\\_msg\\_free](#)

[\\_msgq\\_poll](#)

[\\_msgq\\_receive ...](#)

[\\_msgq\\_send](#)

#### Description

All messages must start with a message header.

## Fields

Field	Description
<b>SIZE</b>	Number of single-addressable units in the message, including the header. The maximum value is MAX_MESSAGE_SIZE. The application sets the field.
<b>TARGET_QID</b>	Queue ID of the queue to which MQX RTOS is to send the message. The application sets the field.
<b>SOURCE_QID</b>	Queue ID of a message queue that is associated with the sending task. When messages are allocated, this field is initialized to MSGQ_NULL_QUEUE_ID. If the sending task does not have a message queue associated with it, MQX RTOS does not use this field.
<b>CONTROL</b>	Indicates the following for the message: endian format priority urgency
<b>RESERVED</b>	Not used

## Example

See `_msgq_send()`.

### 3.2.13 MQX\_INITIALIZATION\_STRUCT

MQX RTOS initialization structure for each processor.

#### Prototype

```
#include <mqx.h>
typedef struct MQX_initialization_struct
{
    _mqx_uint    PROCESSOR_NUMBER;
    void         *START_OF_KERNEL_MEMORY;
    void         *END_OF_KERNEL_MEMORY;
    _mqx_uint    INTERRUPT_STACK_SIZE
    TASK_TEMPLATE_STRUCT_PTR
                TASK_TEMPLATE_LIST;
    _mqx_uint    MQX_HARDWARE_INTERRUPT_LEVEL_MAX;
    _mqx_uint    MAX_MSGPOOLS;
    _mqx_uint    MQX_MSGQS;
    char         *IO_CHANNEL;
    char         *IO_OPEN_MODE;
    _mqx_uint    RESERVED[2];
} MQX_INITIALIZATION_STRUCT, * MQX_INITIALIZATION_STRUCT_PTR;
```

#### See Also

[\\_mqx](#)

[\\_task\\_create](#), [\\_task\\_create\\_blocked](#), [\\_task\\_create\\_at](#), [create\\_task](#)

[\\_task\\_get\\_processor](#)

### TASK\_TEMPLATE\_STRUCT

#### Description

When an application starts MQX RTOS on each processor, it calls [\\_mqx\(\)](#) with the MQX RTOS initialization structure.

## Fields

Field	Description
<b>PROCESSOR_NUMBER</b>	Application-unique processor number of the processor. Minimum is 1, maximum is 255. (Processor number 0 is reserved and is used by tasks to indicate their local processor.)
<b>START_OF_KERNEL_MEMORY</b>	Lowest address from which MQX RTOS allocates dynamic memory and task stacks.
<b>END_OF_KERNEL_MEMORY</b>	Highest address from which MQX RTOS allocates dynamic memory and task stacks. It is the application's responsibility to allocate enough memory for all tasks.
<b>INTERRUPT_STACK_SIZE</b>	Maximum number of single-addressable units used by all ISR stacks.
<b>TASK_TEMPLATE_LIST</b>	Pointer to the task template list for the processor. The default name for the list is <i>MQX_template_list</i> [].
<b>MQX_HARDWARE_INTERRUPT_LEVEL_MAX</b>	Hardware priority at which MQX RTOS runs (for processors with multiple interrupt priority levels). All tasks and interrupts run at lower priority.
<b>MAX_MSGPOOLS</b>	Maximum number of message pools.
<b>MQX_MSGQS</b>	Maximum number of message queues. Minimum is MSGQ_FIRST_USER_QUEUE, maximum is 255.
<b>IO_CHANNEL</b>	Pointer to the string that indicates which device to use as the default. The function <code>_io_fopen()</code> uses the string for default I/O.
<b>IO_OPEN_MODE</b>	Parameter that MQX RTOS passes to the device initialization function when it opens the device.
<b>RESERVED</b>	Reserved for future enhancements to MQX RTOS; each element of the array must be initialized to 0.

## Example

Typical MQX RTOS initialization structure.

```
MQX_INITIALIZATION_STRUCT MQX_init_struct =
{
    /* PROCESSOR_NUMBER */          /* 1,
    /* START_OF_KERNEL_MEMORY */    /* (void*)(0x40000),
    /* END_OF_KERNEL_MEMORY */      /* (void*)(0x2effff),
    /* INTERRUPT_STACK_SIZE */      /* 500,
    /* TASK_TEMPLATE_LIST */        /* (void*)template_list,
    /* MQX_HARDWARE_INTERRUPT_LEVEL_MAX */ /* 6,
    /* MAX_MSGPOOLS */              /* 60,
    /* MQX_MSGQS */                 /* 255,
    /* IO_CHANNEL */                /* BSP_DEFAULT_IO_CHANNEL,
    /* IO_OPEN_MODE */              /* BSP_DEFAULT_IO_OPEN_MODE
};
```

### 3.2.14 MQX\_TICK\_STRUCT

MQX RTOS internally keeps time in ticks.

#### Prototype

```
typedef struct mqx_tick_struct
{
    _mqx_uint  TICKS[MQX_NUM_TICK_FIELDS];
    uint32_t   HW_TICKS;
} MQX_TICK_STRUCT, * MQX_TICK_STRUCT_PTR;
See also
```

All functions that end with **\_ticks**

#### Fields

Field	Description
<b>TICKS[]</b>	Ticks since MQX RTOS started. The field is a minimum of 64 bits; the exact size depends on the PSP.
<b>HW_TICKS</b>	Hardware ticks (timer counter increments) between ticks. The field increases the accuracy over counting the time simply in ticks.

### 3.2.15 MUTEX\_ATTR\_STRUCT

Mutex attributes, which are used to initialize a mutex.

#### Prototype

```
#include <mutex.h>
typedef struct mutex_attr_struct
{
    _mqx_uint    SCHED_PROTOCOL;
    _mqx_uint    VALID;
    _mqx_uint    PRIORITY_CEILING;
    _mqx_uint    COUNT;
    _mqx_uint    WAIT_PROTOCOL;
} MUTEX_ATTR_STRUCT, * MUTEX_ATTR_STRUCT_PTR;
```

#### See Also

[\\_mutatr\\_destroy](#)

[\\_mutatr\\_init](#)

#### Fields

Field	Description
<b>SCHED_PROTOCOL</b>	Scheduling protocol; one of the following: <ul style="list-style-type: none"> <li>• MUTEX_NO_PRIO_INHERIT</li> <li>• MUTEX_PRIO_INHERIT</li> <li>• MUTEX_PRIO_PROTECT</li> <li>• MUTEX_PRIO_INHERIT   MUTEX_PRIO_PROTECT</li> </ul>
<b>VALID</b>	When a task calls <code>_mutatr_init()</code> , MQX RTOS sets the field to <code>MUTEX_VALID</code> (defined in <i>mutex.h</i> ) and does not change it. If the field changes, MQX RTOS considers the attributes invalid. The function <code>_mutatr_init()</code> sets the field to <code>TRUE</code> ; <code>_mutatr_destroy()</code> sets it to <code>FALSE</code> .
<b>PRIORITY_CEILING</b>	Priority of the mutex; applicable only if the scheduling protocol is priority protect.
<b>COUNT</b>	Number of spins to use if the waiting protocol is limited spin.
<b>WAIT_PROTOCOL</b>	Waiting protocol; one of the following: <ul style="list-style-type: none"> <li>• MUTEX_SPIN_ONLY</li> <li>• MUTEX_LIMITED_SPIN</li> <li>• MUTEX_QUEUEING</li> <li>• MUTEX_PRIORITY_QUEUEING</li> </ul>



### 3.2.16 MUTEX\_STRUCT

A mutex.

#### Prototype

```
#include <mutex.h>
typedef struct mutex_struct
{
    void            *NEXT;
    void            *PREV;
    _mqx_uint       POLICY;
    _mqx_uint       VALID;
    _mqx_uint       PRIORITY;
    _mqx_uint       COUNT;
    uint16_t        DELAYED_DESTROY;
    unsigned char    LOCK;
    unsigned char    FILLER;
    QUEUE_STRUCT    WAITING_TASKS;
    void            *OWNER_TD;
    _mqx_uint        BOOSTED;
} MUTEX_STRUCT;
```

#### See Also

[\\_mutex\\_destroy](#)

[\\_mutex\\_init](#)

#### MUTEX\_ATTR\_STRUCT

#### Fields

Field	Description
<b>NEXT PREV</b>	Queue of mutexes. MQX RTOS stores the start and end of the queue in MUTEXES of the MUTEX_COMPONENT_STRUCT.
<b>PROTOCOLS</b>	Waiting protocol (most significant word) and scheduling protocol (least significant word) for the mutex.
<b>VALID</b>	When a task calls <code>_mutex_init()</code> , MQX RTOS sets the field to <code>MUTEX_VALID</code> (defined in <i>mutex.h</i> ) and does not change it. If the field changes, MQX RTOS considers the mutex invalid.
<b>PRIORITY_CEILING</b>	Priority of the mutex. If the scheduling protocol is priority protect, MQX RTOS grants the mutex only to tasks with at least this priority.
<b>COUNT</b>	Maximum number of spins. The field is used only if the waiting protocol is limited spin.
<b>DELAYED_DESTROY</b>	<i>TRUE</i> if the mutex is being destroyed.
<b>LOCK</b>	Most significant bit is set when the mutex is locked.
<b>FILLER</b>	Not used.

<b>WAITING_TASKS</b>	Queue of tasks that are waiting to lock the mutex. If <b>PRIORITY_INHERITANCE</b> is set, the queue is in priority order; otherwise, it is in FIFO order.
<b>OWNER_TD</b>	Task descriptor of the task that has locked the mutex.
<b>BOOSTED</b>	Number of times that MQX RTOS has boosted the priority of the task that has locked the mutex.

### 3.2.17 QUEUE\_ELEMENT\_STRUCT

Header for a queue element.

#### Prototype

```
#include <mqx.h>
typedef struct queue_element_struct
{
    struct queue_element_struct    *NEXT;
    struct queue_element_struct    *PREV;
} QUEUE_ELEMENT_STRUCT, * QUEUE_ELEMENT_STRUCT_PTR;
```

#### See Also

[\\_queue\\_dequeue](#)

[\\_queue\\_enqueue](#)

[\\_queue\\_init](#)

[QUEUE\\_STRUCT](#)

#### Description

Each element in a queue (**QUEUE\_STRUCT**) must start with the structure.

#### Fields

Field	Description
<b>NEXT</b>	Pointer to the next element in the queue.
<b>PREV</b>	Pointer to the previous element in the queue.

### 3.2.18 QUEUE\_STRUCT

Queue of any type of element that has a header of type **QUEUE\_ELEMENT\_STRUCT**.

#### Prototype

```
#include <mqx.h>
typedef struct queue_struct
{
    struct queue_element_struct    *NEXT;
    struct queue_element_struct    *PREV;
    uint16_t                       SIZE;
    uint16_t                       MAX;
} QUEUE_STRUCT, * QUEUE_STRUCT_PTR;
```

#### See Also

[\\_queue\\_init](#)

### QUEUE\_ELEMENT\_STRUCT

#### Fields

Field	Description
<b>NEXT</b>	Pointer to the next element in the queue. If there are no elements in the queue, the field is a pointer to the structure itself.
<b>PREV</b>	Pointer to the last element in the queue. If there are no elements in the queue, the field is a pointer to the structure itself.
<b>SIZE</b>	Number of elements in the queue.
<b>MAX</b>	Maximum number of elements that the queue can hold. If the field is 0, the number is unlimited.

### 3.2.19 TASK\_TEMPLATE\_STRUCT

Task template that MQX RTOS uses to create instances of a task.

#### Prototype

```
#include <mqx.h>
typedef struct task_template_struct
{
    _mqx_uint    TASK_TEMPLATE_INDEX;
    TASK_FPTR    TASK_ADDRESS;
    _mem_size    TASK_STACKSIZE;
    _mqx_uint    TASK_PRIORITY;
    char _PTR    TASK_NAME;
    _mqx_uint    TASK_ATTRIBUTES;
    uint32_t     CREATION_PARAMETER;
    _mqx_uint    DEFAULT_TIME_SLICE;
} TASK_TEMPLATE_STRUCT, * TASK_TEMPLATE_STRUCT_PTR;
```

#### See Also

[\\_mqx](#)

[\\_task\\_create](#), [\\_task\\_create\\_blocked](#), [\\_task\\_create\\_at](#), [create\\_task](#)

### MQX\_INITIALIZATION\_STRUCT

#### Description

The task template list is an array of these structures, terminated by a zero-filled element. The MQX RTOS initialization structure contains a pointer to the list.

## Fields

Field	Description
<b>TASK_TEMPLATE_INDEX</b>	Application-unique number that identifies the task template. The minimum value is 1, maximum is MAX_MQX_UINT. The field is ignored if you call <code>_task_create()</code> or <code>_task_create_blocked()</code> or <code>_task_create_at()</code> with a template index equal to 0 and a creation parameter set to a pointer to a task template.
<b>TASK_ADDRESS</b>	Pointer to the root function for the task. When MQX RTOS creates the task, the task begins running at this address.
<b>TASK_STACKSIZE</b>	Number of single-addressable units of stack space that the task needs.
<b>TASK_PRIORITY</b>	Software priority of the task. Priorities start at 0, which is the highest priority; 1, 2, 3, and so on, are progressively lower priorities.
<b>TASK_NAME</b>	Pointer to a name for tasks that MQX RTOS creates from the template.
<b>TASK_ATTRIBUTES</b>	Attributes of tasks that MQX RTOS creates from the template; any combination of:
<b>NULL</b>	When MQX RTOS starts, it does not create an instance of the task. MQX RTOS uses FIFO scheduling for the task. MQX RTOS does not save floating-point registers as part of the task's context.
<b>MQX_AUTO_START_TASK</b>	When MQX RTOS starts, it creates one instance of the task.
<b>MQX_DSP_TASK</b>	MQX RTOS saves the DSP coprocessor registers as part of the task's context. If the DSP registers are separate from the normal registers, MQX RTOS manages their context independently during task switching. MQX RTOS saves or restores the registers only when a new DSP task is scheduled to run.
<b>MQX_FLOATING_POINT_TASK</b>	MQX RTOS saves floating-point registers as part of the task's context.
<b>MQX_TIME_SLICE_TASK</b>	MQX RTOS uses round robin scheduling for the task (the default is FIFO scheduling).
<b>CREATION_PARAMETER</b>	Passed to tasks that MQX RTOS creates from the template.
<b>DEFAULT_TIME_SLICE</b>	If the task uses round robin scheduling and the field is non-zero, MQX RTOS uses the value as the task's time slice value. If the task uses round robin scheduling and the field is 0, MQX RTOS uses the default time slice value.

### Example

```
#include<mqx.h>
...
extern void taskA();

TASK_TEMPLATE_STRUCT task_list[] =
{
    {FIRST_TASK, taskA, 0x2000, MAIN_PRIOR,
      "taskA", MQX_AUTO_START_TASK, (uint32_t)MY_QUEUE, 0},
    {0,        0,        0,        0,        0,
     0,        0,        0,        0,        0},
};
```

### 3.2.20 TIME\_STRUCT

Time in millisecond format.

#### Prototype

```
#include <mqx.h>
typedef struct time_struct
{
    uint32_t SECONDS;
    uint32_t MILLISECONDS;
} TIME_STRUCT, * TIME_STRUCT_PTR;
```

#### See also

[\\_time\\_from\\_date](#)

[\\_time\\_get, \\_time\\_get\\_ticks](#)

[\\_time\\_set, \\_time\\_set\\_ticks](#)

[\\_time\\_to\\_date](#)

[DATE\\_STRUCT](#)

#### Fields

Field	Description
SECONDS	Number of seconds.
MILLISECONDS	Number of milliseconds.



### 3.2.21 time\_t

Time in second format, representing the number of seconds elapsed since 00:00:00, Jan 1, 1970 UTC.

#### Prototype

```
#include <mqx.h>
/*-----*/
/* time_t */
/*!
 *
 */
#ifndef __time_t_defined
typedef uint_32    time_t;
#define __time_t_defined
#endif
```

#### See also

mktime

gmtime\_r

localtime\_r

timegm

TM STRUCT

### 3.2.22 TM STRUCT

Time in second format, representing the number of seconds elapsed since 00:00:00, Jan 1, 1970 UTC.

#### Prototype

```
#include <mqx.h>
struct tm {
    int_32    tm_sec;
    int_32    tm_min;
    int_32    tm_hour;
    int_32    tm_mday;
    int_32    tm_mon;
    int_32    tm_year;
    int_32    tm_wday;
    int_32    tm_yday;
    int_32    tm_isdst;
};
```

#### See also

[mktime](#)

[gmtime\\_r](#)

[localtime\\_r](#)

[timegm](#)

[time\\_t](#)

#### Fields

Field	Description
tm_sec	The number of seconds after the minute, normally in the range 0 to 59, but can be up to 60 to allow for leap seconds.
tm_min	The number of minutes after the hour, in the range 0 to 59.
tm_hour	The number of hours past midnight, in the range 0 to 23.
tm_mday	The day of the month, in the range 1 to 31.
tm_mon	The number of months since January, in the range 0 to 11.
tm_year	The number of years since 1900.
tm_wday	The number of days since Sunday, in the range 0 to 6.
tm_yday	The number of days since January 1, in the range 0 to 365.
tm_isdst	A flag that indicates whether daylight saving time is in effect at the time described. The value is positive if daylight saving time is in effect, zero if it is not, and negative if the information is not available.



