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.

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 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.

<table>
<thead>
<tr>
<th>Revision Number</th>
<th>Revision Date</th>
<th>Description of Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rev. 0</td>
<td>01/2009</td>
<td>Initial Release coming with MQX RTOS version 3.0</td>
</tr>
<tr>
<td>Rev. 1</td>
<td>05/2009</td>
<td>Minor formatting updates.</td>
</tr>
<tr>
<td>Rev. 3</td>
<td>01/2010</td>
<td>Update coming with MQX RTOS version 3.5. <em>lwevent_wait</em> description updated.</td>
</tr>
<tr>
<td>Rev. 4</td>
<td>08/2010</td>
<td>Update of _lwevent_xxx &amp; _time_diff_xxx sections.</td>
</tr>
<tr>
<td>Rev. 5</td>
<td>11/2010</td>
<td>Update of the following sections: _time_diff_ticks, _task_create_xxx, _task_get_template_ptr, _mem_alloc, _lwmem_alloc, _lwevent_get_signalled (chapter added)</td>
</tr>
<tr>
<td>Rev. 6</td>
<td>04/2011</td>
<td>Update of _time_get_ticks_per_sec, _lwmqgq_init and _sem_open and _time_delay sections.</td>
</tr>
<tr>
<td>Rev. 8</td>
<td>06/2012</td>
<td>_lwtimer_add_timer_to_queue(), _msgq_send_queue and _task_stop_preemption() function descriptions updated.</td>
</tr>
<tr>
<td>Rev. 9</td>
<td>12/2012</td>
<td>Update reflecting changes in the MQX RTOS code and the source tree structure (paths, prototypes, file names, etc.)</td>
</tr>
<tr>
<td>Revision Number</td>
<td>Revision Date</td>
<td>Description of Changes</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Rev. 10</td>
<td>04/2013</td>
<td>_lwmsgq_send and _lwmsgq_receive function descriptions updated. Cautions added to all _DCACHE_xxx sections. Task Error Codes of _msgq_send_broadcast, _lwmem_alloc, <em>lwmem_alloc</em><em>_from and _mem_alloc sections updated. Error codes of _mutex_destroy, _lwmem_free and _mem_test functions updated. Return values of the _mutatr_init, _mem_extend_pool and _mem_extend functions updated. _lwmem_alloc and <em>lwmem_alloc</em></em>_from sections updated.</td>
</tr>
<tr>
<td>Rev. 11</td>
<td>05/2013</td>
<td>Added description of following functions: _mem_alloc_system_align _mem_alloc_system_align_from _lwmem_alloc_system_align _lwmem_alloc_system_align_from</td>
</tr>
<tr>
<td>Rev. 12</td>
<td>06/2013</td>
<td>Made corrections to various <em>mmu</em> functions.</td>
</tr>
<tr>
<td>Rev. 13</td>
<td>08/2013</td>
<td>Added _psp_push_fp_context and _psp_pop_fp_context.</td>
</tr>
<tr>
<td>Rev. 14</td>
<td>10/2013</td>
<td>Updated <em>lwmem_alloc</em>*_from, _mem_alloc ..., and _time_get_resolution, _time_set_resolution</td>
</tr>
<tr>
<td>Rev. 15</td>
<td>01/2014</td>
<td>Added C99 types.</td>
</tr>
<tr>
<td>Rev. 16</td>
<td>02/2014</td>
<td>4.1.0 updates.</td>
</tr>
<tr>
<td>Rev. 17</td>
<td>07/2014</td>
<td>Kinetis SDK-related updates</td>
</tr>
<tr>
<td>Rev. 18</td>
<td>12/2014</td>
<td>Updates for KSDK 1.1.0</td>
</tr>
<tr>
<td>Rev. 19</td>
<td>04/2015</td>
<td>Updated for KSDK 1.2.0</td>
</tr>
</tbody>
</table>

© 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
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_int_exception_isr</td>
<td>61</td>
</tr>
<tr>
<td>_int_get_default_isr</td>
<td>62</td>
</tr>
<tr>
<td>_int_get_exception_handler</td>
<td>63</td>
</tr>
<tr>
<td>_int_get_isr</td>
<td>64</td>
</tr>
<tr>
<td>_int_get_isr_data</td>
<td>65</td>
</tr>
<tr>
<td>_int_get_isr_depth</td>
<td>66</td>
</tr>
<tr>
<td>_int_get_kernel_isr</td>
<td>67</td>
</tr>
<tr>
<td>_int_get_previous_vector_table</td>
<td>68</td>
</tr>
<tr>
<td>_int_get_vector_table</td>
<td>69</td>
</tr>
<tr>
<td>_int_install_default_isr</td>
<td>70</td>
</tr>
<tr>
<td>_int_install_exception_isr</td>
<td>71</td>
</tr>
<tr>
<td>_int_install_isr</td>
<td>72</td>
</tr>
<tr>
<td>_int_install_kernel_isr</td>
<td>74</td>
</tr>
<tr>
<td>_int_install_unexpected_isr</td>
<td>75</td>
</tr>
<tr>
<td>_int_kernel_isr</td>
<td>76</td>
</tr>
<tr>
<td>_int_set_exception_handler</td>
<td>77</td>
</tr>
<tr>
<td>_int_set_isr_data</td>
<td>78</td>
</tr>
<tr>
<td>_int_set_vector_table</td>
<td>79</td>
</tr>
<tr>
<td>_int_unexpected_isr</td>
<td>80</td>
</tr>
<tr>
<td>_psp_push_fp_context</td>
<td>81</td>
</tr>
<tr>
<td>_psp_pop_fp_context</td>
<td>82</td>
</tr>
<tr>
<td>_ipc_add_io_ipc_handler</td>
<td>83</td>
</tr>
<tr>
<td>_ipc_add_ipc_handler</td>
<td>84</td>
</tr>
<tr>
<td>_ipc_msg_processor_route_exists</td>
<td>85</td>
</tr>
<tr>
<td>_ipc_msg_route_add</td>
<td>86</td>
</tr>
<tr>
<td>_ipc_msg_route_remove</td>
<td>87</td>
</tr>
<tr>
<td>_ipc_pcb_init</td>
<td>88</td>
</tr>
<tr>
<td>_ipc_task</td>
<td>89</td>
</tr>
<tr>
<td>_klog_control</td>
<td>90</td>
</tr>
<tr>
<td>_klog_create, _klog_create_at</td>
<td>91</td>
</tr>
<tr>
<td>_klog_disable_logging_task, _klog_enable_logging_task</td>
<td>92</td>
</tr>
<tr>
<td>_klog_display</td>
<td>96</td>
</tr>
<tr>
<td>_klog_get_interrupt_stack_usage</td>
<td>97</td>
</tr>
<tr>
<td>_klog_get_task_stack_usage</td>
<td>99</td>
</tr>
<tr>
<td>_klog_show_stack_usage</td>
<td>100</td>
</tr>
<tr>
<td>_log_create</td>
<td>101</td>
</tr>
<tr>
<td>_log_create_component</td>
<td>103</td>
</tr>
<tr>
<td>_log_destroy</td>
<td>104</td>
</tr>
<tr>
<td>_log_enable</td>
<td>105</td>
</tr>
<tr>
<td>_log_read</td>
<td>106</td>
</tr>
<tr>
<td>_log_reset</td>
<td>108</td>
</tr>
<tr>
<td>_log_test</td>
<td>109</td>
</tr>
<tr>
<td>_log_write</td>
<td>110</td>
</tr>
<tr>
<td>_lwevent_clear</td>
<td>111</td>
</tr>
<tr>
<td>_lwevent_create</td>
<td>112</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2.1.209_partition_alloc, _partition_alloc_zero</td>
<td>289</td>
</tr>
<tr>
<td>2.1.210_partition_alloc_system, _partition_alloc_system_zero</td>
<td>291</td>
</tr>
<tr>
<td>2.1.211_partition_calculate_blocks</td>
<td>292</td>
</tr>
<tr>
<td>2.1.212_partition_calculate_size</td>
<td>293</td>
</tr>
<tr>
<td>2.1.213_partition_create</td>
<td>294</td>
</tr>
<tr>
<td>2.1.214_partition_create_at</td>
<td>296</td>
</tr>
<tr>
<td>2.1.215_partition_create_component</td>
<td>298</td>
</tr>
<tr>
<td>2.1.216_partition_destroy</td>
<td>299</td>
</tr>
<tr>
<td>2.1.217_partition_extend</td>
<td>300</td>
</tr>
<tr>
<td>2.1.218_partition_free</td>
<td>301</td>
</tr>
<tr>
<td>2.1.219_partition_get_block_size</td>
<td>302</td>
</tr>
<tr>
<td>2.1.220_partition_get_free_blocks</td>
<td>304</td>
</tr>
<tr>
<td>2.1.221_partition_get_max_used_blocks</td>
<td>305</td>
</tr>
<tr>
<td>2.1.222_partition_get_total_blocks</td>
<td>306</td>
</tr>
<tr>
<td>2.1.223_partition_get_total_size</td>
<td>307</td>
</tr>
<tr>
<td>2.1.224_partition_test</td>
<td>308</td>
</tr>
<tr>
<td>2.1.225_partition_transfer</td>
<td>309</td>
</tr>
<tr>
<td>2.1.226_queue_dequeue</td>
<td>310</td>
</tr>
<tr>
<td>2.1.227_queue_enqueue</td>
<td>312</td>
</tr>
<tr>
<td>2.1.228_queue_get_size</td>
<td>313</td>
</tr>
<tr>
<td>2.1.229_queue_head</td>
<td>314</td>
</tr>
<tr>
<td>2.1.230_queue_init</td>
<td>315</td>
</tr>
<tr>
<td>2.1.231_queue_insert</td>
<td>316</td>
</tr>
<tr>
<td>2.1.232_queue_is_empty</td>
<td>318</td>
</tr>
<tr>
<td>2.1.233_queue_next</td>
<td>319</td>
</tr>
<tr>
<td>2.1.234_queue_test</td>
<td>320</td>
</tr>
<tr>
<td>2.1.235_queue_unlink</td>
<td>321</td>
</tr>
<tr>
<td>2.1.236_sched_get_max_priority</td>
<td>322</td>
</tr>
<tr>
<td>2.1.237_sched_get_min_priority</td>
<td>323</td>
</tr>
<tr>
<td>2.1.238_sched_get_policy</td>
<td>324</td>
</tr>
<tr>
<td>2.1.239_sched_get_rr_interval, _sched_get_rr_interval_ticks</td>
<td>325</td>
</tr>
<tr>
<td>2.1.240_sched_set_policy</td>
<td>326</td>
</tr>
<tr>
<td>2.1.241_sched_set_rr_interval, _sched_set_rr_interval_ticks</td>
<td>327</td>
</tr>
<tr>
<td>2.1.242_sched_yield</td>
<td>328</td>
</tr>
<tr>
<td>2.1.243_sem_close</td>
<td>329</td>
</tr>
<tr>
<td>2.1.244_sem_create</td>
<td>330</td>
</tr>
<tr>
<td>2.1.245_sem_create_component</td>
<td>332</td>
</tr>
<tr>
<td>2.1.246_sem_create_fast</td>
<td>334</td>
</tr>
<tr>
<td>2.1.247_sem_destroy, _sem_destroy_fast</td>
<td>335</td>
</tr>
<tr>
<td>2.1.248_sem_get_value</td>
<td>337</td>
</tr>
<tr>
<td>2.1.249_sem_get_wait_count</td>
<td>338</td>
</tr>
<tr>
<td>2.1.250_sem_open, _sem_open_fast</td>
<td>339</td>
</tr>
<tr>
<td>2.1.251_sem_post</td>
<td>341</td>
</tr>
<tr>
<td>2.1.252_sem_test</td>
<td>343</td>
</tr>
<tr>
<td>2.1.253_sem_wait</td>
<td>344</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>str_mqx_uint_to_hex_string</td>
<td>346</td>
</tr>
<tr>
<td>strnlen</td>
<td>347</td>
</tr>
<tr>
<td>task_abort</td>
<td>348</td>
</tr>
<tr>
<td>task_block</td>
<td>350</td>
</tr>
<tr>
<td>task_check_stack</td>
<td>351</td>
</tr>
<tr>
<td>task_create, _task_create_blocked, _task_create_at, create_task</td>
<td>352</td>
</tr>
<tr>
<td>task_destroy</td>
<td>354</td>
</tr>
<tr>
<td>task_disable_fp, _task_enable_fp</td>
<td>356</td>
</tr>
<tr>
<td>errno</td>
<td>357</td>
</tr>
<tr>
<td>task_get_creator</td>
<td>358</td>
</tr>
<tr>
<td>task_get_environment, _task_set_environment</td>
<td>359</td>
</tr>
<tr>
<td>task_get_error, _task_get_error_ptr</td>
<td>360</td>
</tr>
<tr>
<td>task_get_exception_handler, _task_set_exception_handler</td>
<td>361</td>
</tr>
<tr>
<td>task_get_exit_handler, _task_set_exit_handler</td>
<td>362</td>
</tr>
<tr>
<td>task_get_id</td>
<td>363</td>
</tr>
<tr>
<td>task_get_id_from_name</td>
<td>364</td>
</tr>
<tr>
<td>task_get_index_from_id</td>
<td>365</td>
</tr>
<tr>
<td>task_get_parameter ... _task_set_parameter ...</td>
<td>366</td>
</tr>
<tr>
<td>task_get_priority</td>
<td>367</td>
</tr>
<tr>
<td>task_get_processor</td>
<td>369</td>
</tr>
<tr>
<td>task_get_td</td>
<td>370</td>
</tr>
<tr>
<td>task_get_template_index</td>
<td>371</td>
</tr>
<tr>
<td>task_get_template_ptr</td>
<td>372</td>
</tr>
<tr>
<td>task_ready</td>
<td>373</td>
</tr>
<tr>
<td>task_restart</td>
<td>375</td>
</tr>
<tr>
<td>task_set_error</td>
<td>376</td>
</tr>
<tr>
<td>task_start_preemption, _task_stop_preemption</td>
<td>377</td>
</tr>
<tr>
<td>taskq_create</td>
<td>378</td>
</tr>
<tr>
<td>taskq_destroy</td>
<td>380</td>
</tr>
<tr>
<td>taskq_get_value</td>
<td>381</td>
</tr>
<tr>
<td>taskq_resume</td>
<td>382</td>
</tr>
<tr>
<td>taskq_suspend</td>
<td>384</td>
</tr>
<tr>
<td>taskq_suspend_task</td>
<td>385</td>
</tr>
<tr>
<td>taskq_test</td>
<td>387</td>
</tr>
<tr>
<td>ticks_to_time</td>
<td>388</td>
</tr>
<tr>
<td>time_add</td>
<td>389</td>
</tr>
<tr>
<td>time_delay</td>
<td>391</td>
</tr>
<tr>
<td>time_dequeue</td>
<td>392</td>
</tr>
<tr>
<td>time_dequeue_td</td>
<td>394</td>
</tr>
<tr>
<td>time_diff, _time_diff_ticks</td>
<td>395</td>
</tr>
<tr>
<td>time_diff_</td>
<td>397</td>
</tr>
<tr>
<td>time_from_date</td>
<td>400</td>
</tr>
<tr>
<td>time_get, _time_get_ticks</td>
<td>402</td>
</tr>
<tr>
<td>time_get_elapsed, _time_get_elapsed_ticks</td>
<td>403</td>
</tr>
<tr>
<td>time_get_hwticks</td>
<td>404</td>
</tr>
</tbody>
</table>
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
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.

```plaintext
function_name()
```

A short description of what function `function_name()` does.

Prototype

Provides a prototype for the function `function_name()`.

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

Parameters

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

<table>
<thead>
<tr>
<th>Component</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache-control macros for data cache</td>
<td><em>DCACHE</em></td>
</tr>
<tr>
<td>Cache-control macros for instruction cache</td>
<td><em>ICACHE</em></td>
</tr>
<tr>
<td>Endian conversion macros</td>
<td>MSG_</td>
</tr>
<tr>
<td>Events</td>
<td><em>event</em></td>
</tr>
<tr>
<td>Inter-processor communication</td>
<td><em>ipc</em></td>
</tr>
<tr>
<td>Interrupt handling</td>
<td><em>int</em></td>
</tr>
<tr>
<td>Kernel log</td>
<td><em>klog</em></td>
</tr>
<tr>
<td>Lightweight events</td>
<td><em>lwevent</em></td>
</tr>
<tr>
<td>Lightweight logs</td>
<td><em>lwlog</em></td>
</tr>
<tr>
<td>Lightweight memory with variable-size blocks</td>
<td><em>lwmem</em></td>
</tr>
<tr>
<td>Lightweight semaphores</td>
<td><em>lwsem</em></td>
</tr>
<tr>
<td>Logs (user logs)</td>
<td><em>log</em></td>
</tr>
<tr>
<td>Memory with fixed-size blocks (partitions)</td>
<td><em>partition</em></td>
</tr>
<tr>
<td>Memory with variable-size blocks</td>
<td><em>mem</em></td>
</tr>
<tr>
<td>Messages</td>
<td><em>msg</em></td>
</tr>
<tr>
<td></td>
<td><em>msgpool</em></td>
</tr>
<tr>
<td></td>
<td><em>msgq</em></td>
</tr>
<tr>
<td>Miscellaneous</td>
<td><em>mqx</em></td>
</tr>
<tr>
<td>MMU and virtual memory control</td>
<td><em>mmu</em></td>
</tr>
<tr>
<td>Mutexes</td>
<td><em>mutatr</em></td>
</tr>
<tr>
<td></td>
<td><em>mutex</em></td>
</tr>
<tr>
<td>Names</td>
<td><em>name</em></td>
</tr>
<tr>
<td>Partitions</td>
<td><em>partition</em></td>
</tr>
<tr>
<td>Queues</td>
<td><em>queue</em></td>
</tr>
<tr>
<td>Scheduling</td>
<td><em>sched</em></td>
</tr>
<tr>
<td>Semaphores</td>
<td><em>sem</em></td>
</tr>
<tr>
<td>String functions</td>
<td><em>str</em></td>
</tr>
<tr>
<td>Task management</td>
<td><em>task</em></td>
</tr>
<tr>
<td>Task queues</td>
<td><em>taskq</em></td>
</tr>
<tr>
<td>Timers</td>
<td><em>timer</em></td>
</tr>
<tr>
<td>Timing</td>
<td><em>time</em></td>
</tr>
</tbody>
</table>
## Table 2-1. MQX RTOS Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual memory control</td>
<td><em>mmu</em></td>
</tr>
<tr>
<td>Watchdogs</td>
<td>_watchdog</td>
</tr>
</tbody>
</table>
2.1.1 _DCACHE_DISABLE

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

Prototype

```
#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

```c
#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

```c
#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

```c
#include <psp.h>
void _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.

```c
... 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

```c
#include <psp.h>
void _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.

```c
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

#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

```c
#include <psp.h>

_VOID void _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.

```c
... 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

```c
#include <psp.h>

#define _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.

```c
... 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

```
#include <event.h>
_mqx_uint _event_clear(
    void *event_group_ptr,
    _mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_INVALID_EVENT</td>
<td>Event group is not valid.</td>
</tr>
<tr>
<td>EVENT_INVALID_EVENT_HANDLE</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>- <code>_event_open()</code> or <code>_event_open_fast()</code> did not get the event</td>
</tr>
<tr>
<td></td>
<td>group handle</td>
</tr>
<tr>
<td></td>
<td>- <code>_event_create()</code> did not create the event group</td>
</tr>
</tbody>
</table>

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.

```c
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**

```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.

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_INVALID_EVENT_HANDLE</td>
<td>Event group connection is not valid.</td>
</tr>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_event_create()</td>
<td>Creates the named event group.</td>
</tr>
<tr>
<td>_event_create_auto_clear()</td>
<td>Creates the named event group with autclearing event bits.</td>
</tr>
</tbody>
</table>

Prototype

_function_create()

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

_function_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

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

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

```c
#include <event.h>
_mqx_uint _event_create_component(
    _mqx_uint initial_number,
    _mqx_uint grow_number,
    _mqx_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 that 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial_number</td>
<td>8</td>
</tr>
<tr>
<td>grow_number</td>
<td>8</td>
</tr>
<tr>
<td>maximum_number</td>
<td>0 (unlimited)</td>
</tr>
</tbody>
</table>
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.

```c
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_event_create_fast()</td>
<td>Creates the fast event group.</td>
</tr>
<tr>
<td>_event_create_fast_auto_clear()</td>
<td>Creates the fast event group with autoclearing event bits.</td>
</tr>
</tbody>
</table>

Prototype

```
_event_create_fast()

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

_event_create_fast_auto_clear()

source\kernel\event.c
#include <event.h>
_mqx_uint _event_create_fast_auto_clear(
  _mqx_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

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

Parameters

- `name [IN]` — Name of the event group

Returns

- MQX_OK
- Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_INVALID_EVENT</td>
<td>Event group is no longer valid.</td>
</tr>
<tr>
<td>EVENT_NOT_FOUND</td>
<td>Event group is not in the table.</td>
</tr>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Event component was not created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Event component data is not valid.</td>
</tr>
</tbody>
</table>

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(
  _mqx_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

```c
#include <event.h>
_mqx_uint _event_get_value(
    void        *event_group_ptr,
    _mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_INVALID_EVENT</td>
<td>Event group is no longer valid.</td>
</tr>
<tr>
<td>EVENT_INVALID_EVENT_HANDLE</td>
<td>Event group handle is not valid.</td>
</tr>
</tbody>
</table>

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.

```c
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);
            }
        }
    }
}
```
…
}
}

MQX RTOS Functions and Macros
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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_INVALID_EVENT</td>
<td>Event group data is no longer valid.</td>
</tr>
<tr>
<td>EVENT_NOT_FOUND</td>
<td>Named event group is not in the name table.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Event component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Event component data is not valid.</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>MQX RTOS could not allocate memory for the event connection data.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>This string:</th>
<th>Opens this named event group:</th>
<th>On this processor:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;2:Fred&quot;</td>
<td>&quot;Fred&quot;</td>
<td>2</td>
</tr>
<tr>
<td>&quot;0:Sue&quot;</td>
<td>&quot;Sue&quot;</td>
<td>Local processor</td>
</tr>
</tbody>
</table>

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

```c
#include <event.h>
_mqx_uint _event_open_fast(
    _mqx_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

```c
#include <event.h>
_mqx_uint _event_set(
    void    *event_group_ptr,
    _mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_INVALID_EVENT</td>
<td>Event group is no longer valid.</td>
</tr>
<tr>
<td>EVENT_INVALID_EVENT_HANDLE</td>
<td>Event group handle is not a valid event connection.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Event component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Event component data is no longer valid.</td>
</tr>
</tbody>
</table>

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.

```c
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

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

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 ...

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_event_wait_all()</td>
<td>Wait for all the specified event bits to be set in the event group: For the number of milliseconds</td>
</tr>
<tr>
<td>_event_wait_all_for()</td>
<td>For the number of ticks (in tick time)</td>
</tr>
<tr>
<td>_event_wait_all_ticks()</td>
<td>For the number of ticks</td>
</tr>
<tr>
<td>_event_wait_all_until()</td>
<td>Until the specified time (in tick time)</td>
</tr>
</tbody>
</table>

Prototype

```c
#include <event.h>

_mqx_uint _event_wait_all(
    void *event_group_ptr,
    _mqx_uint bit_mask,
    uint32_t ms_timeout)
```

```c
#include <event.h>

_mqx_uint _event_wait_all_for(
    void *event_group_ptr,
    _mqx_uint bit_mask,
    MQX_TICK_STRUCT_PTR tick_time_timeout_ptr)
```

```c
#include <event.h>

_mqx_uint _event_wait_all_ticks(
    void *event_group_ptr,
    _mqx_uint bit_mask,
    uint32_t tick_timeout)
```

```c
#include <event.h>

_mqx_uint _event_wait_all_until(
    void *event_group_ptr,
    _mqx_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

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

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 ...

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_event_wait_any()</td>
<td>Wait for any of the specified event bits to be set in the event group:</td>
</tr>
<tr>
<td>_event_wait_any_for()</td>
<td>For the number of milliseconds</td>
</tr>
<tr>
<td>_event_wait_any_ticks()</td>
<td>For the number of ticks (in tick time)</td>
</tr>
<tr>
<td>_event_wait_any_until()</td>
<td>Until the specified time (in tick time)</td>
</tr>
</tbody>
</table>

Prototype

```c
#include <event.h>

_mqx_uint _event_wait_any(
    void *event_group_ptr,
    _mqx_uint bit_mask,
    uint32_t ms_timeout)

#include <event.h>

_mqx_uint _event_wait_any_for(
    void *event_group_ptr,
    _mqx_uint bit_mask,
    MQX_TICK_STRUCT_PTR tick_time_timeout_ptr)

#include <event.h>

_mqx_uint _event_wait_any_ticks(
    void *event_group_ptr,
    _mqx_uint bit_mask,
    _mqx_uint tick_timeout)

#include <event.h>

_mqx_uint _event_wait_any_until(
    void *event_group_ptr,
    _mqx_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

```c
#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**

```c
#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

```
#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.

```c
...
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

```c
#include <psp.h>

void _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.

```c
... 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_int_disable()</td>
<td>Disable hardware interrupts.</td>
</tr>
<tr>
<td>_int_enable()</td>
<td>Enable hardware interrupts.</td>
</tr>
</tbody>
</table>

Prototype

```c
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

```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

```c
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

```c
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

```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)
• >= 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

```c
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.

```c
#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

```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

```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**

```c
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

```c
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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Interrupt component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_VECTORED_INTERRUPT</td>
<td>Vector is outside the valid range of interrupt numbers.</td>
</tr>
</tbody>
</table>

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()`.

```c
_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

```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\<core_family>\core\<core>\dispatch.S
void _int_kernel_isr(void)

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

\begin{verbatim}
source\kernel\int.c
INT_EXCEPTION_FPTR \_int_set_exception\_handler
(_mqx_uint vector,
 INT_EXCEPTION_FPTR error\_handler\_address)
\end{verbatim}

Parameters

- \textit{vector [IN]} — Interrupt vector that this exception handler is for
- \textit{error\_handler\_address [IN]} — Pointer to the exception handler

Returns

- Pointer to the exception handler before the function was called (success)
- \textit{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

```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

```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 _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 _psp_pop_fp_context(void)

Parameters
None

Returns
None

See Also
_psp_push_fp_context

Description
This function must be used in each interrupt handler which uses FPU extension. Call it at the end, when every operation with the floatpoints finished.
2.1.52 _ipc_add_io_ipc_handler

Add an IPC handler for the I/O component.

Prototype

```c
#include <ipc.h>
_mqx_uint _ipc_add_io_ipc_handler(
  IFC_HANDLER_FPTR handler,
  _mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_IPC_SERVICE_NOT_AVAILBLE</td>
<td>IPC server has not been started.</td>
</tr>
</tbody>
</table>

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_SDLC_COMPONENT
- IO_SNMP_COMPONENT
- IO_SUBSYSTEM_COMPONENT
2.1.53 _ipc_add_ipc_handler

Adds an IPC handler for the MQX RTOS component.

Prototype

```c
#include <ipc.h>
_mqx_uint _ipc_add_ipc_handler(
  IPC_HANDLER_FPTR handler,
  _mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_IPC_SERVICE_NOT_AVAILABLE</td>
<td>IPC server has not been started.</td>
</tr>
</tbody>
</table>

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

```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`
MQX RTOS Functions and Macros

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

```c
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_PCIE_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_PCIE_INIT_STRUCT`.

Example

Initialize an IPC for the PCB.

```c
IPC_PCIE_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,
};
```
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

```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.

```c
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 };
};
```
2.1.59 _klog_control

Controls logging in kernel log.

Prototype

```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.

<table>
<thead>
<tr>
<th>If this bit is set:</th>
<th>MQX RTOS:</th>
</tr>
</thead>
<tbody>
<tr>
<td>KLOG_ENABLED (log MQX RTOS services)</td>
<td>Logs to kernel log</td>
</tr>
</tbody>
</table>
### Example

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

```c
_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()) { }
```

<table>
<thead>
<tr>
<th>If combinations of these bits are set:</th>
<th>Select combinations from:</th>
</tr>
</thead>
</table>
| **KLOG_FUNCTIONS_ENABLED** (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 |
| **KLOG_TASK_QUALIFIED** (log specific tasks only)       | For each task to log, call one of:
_klog_disable_logging_task() |
_klog_enable_logging_task() |
| **KLOG_INTERRUPTS_ENABLED** (log interrupts)            | — |
| **KLOG_SYSTEM_CLOCK_INT_ENABLED** (log periodic timer interrupts) | — |
| **KLOG_CONTEXT_ENABLED** (log context switches)        | — |
2.1.60  _klog_create,  _klog_create_at

| _klog_create() | Creates kernel log. |
| _klog_create_at() | Creates kernel log at the specific location |

Prototype

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

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

Parameters

- **max_size [IN]** — Maximum size (in **mqx_max_type**) 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

<table>
<thead>
<tr>
<th>Errors from _lwlog_create()</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_EXISTS</td>
<td>Kernel log already exists.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Log component data is not valid.</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>MQX RTOS cannot allocate memory for kernel log.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_klog_disable_logging_task()</td>
<td>Disables kernel logging for the task.</td>
</tr>
<tr>
<td>_klog_enable_logging_task()</td>
<td>Enables kernel logging for the task.</td>
</tr>
</tbody>
</table>

**Prototype**

```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**

```c
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**

```c
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.

```c
_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)

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_INVALID_CONFIGURATION</td>
<td>Compile-time configuration option MQX_MONITOR_STACK is not set.</td>
</tr>
<tr>
<td>MQX_INVALID_TASK_ID</td>
<td>task_id is not valid.</td>
</tr>
</tbody>
</table>

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

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

Parameters

- `log_number [IN]` — Log number to create (0 through 15)
- `max_size [IN]` — Maximum number of _mqx_uints 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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_EXISTS</td>
<td>Log log_number has already been created.</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>MQX RTOS is out of memory.</td>
</tr>
</tbody>
</table>

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_OLEDEST_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

```c
#include <log.h>
_mqx_uint  _log_destroy(
    _mqx_uint  log_number)
```

Parameters

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

Returns

- MQX_OK
- Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_DOES_NOT_EXIST</td>
<td><code>log_number</code> was not previously created.</td>
</tr>
<tr>
<td>LOG_INVALID</td>
<td><code>log_number</code> is out of range.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Log component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_HANDLE</td>
<td>Log component data is not valid.</td>
</tr>
</tbody>
</table>

See Also

- _log_create
- _log_create_component

Example

See _log_create().
2.1.69  _log_disable, _log_enable

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_log_disable()</td>
<td>Stops logging to the log.</td>
</tr>
<tr>
<td>_log_enable()</td>
<td>Starts logging to the log.</td>
</tr>
</tbody>
</table>

Prototype

```
#include <log.h>

_mqx_uint  _log_disable(
    _mqx_uint  log_number)

_mqx_uint  _log_enable(
    _mqx_uint  log_number)
```

Parameters

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

Returns

- MQX_OK
- Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_DOES_NOT_EXIST</td>
<td><code>log_number</code> was not created.</td>
</tr>
<tr>
<td>LOG_INVALID</td>
<td><code>log_number</code> is out of range.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Log component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_HANDLE</td>
<td>Log component data is not valid.</td>
</tr>
</tbody>
</table>

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

```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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG DOES NOT_EXIST</td>
<td><code>log_number</code> was not created.</td>
</tr>
<tr>
<td>LOG_ENTRY NOT AVAILABLE</td>
<td>Log entry is not available.</td>
</tr>
<tr>
<td>LOG INVALID</td>
<td><code>log_number</code> is out of range.</td>
</tr>
<tr>
<td>LOGINVALID_READ_TYPE</td>
<td><code>read_type</code> is not valid.</td>
</tr>
<tr>
<td>MQX_COMPONENT DOES NOT_EXIST</td>
<td>Log component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_HANDLE</td>
<td>Log component data is not valid.</td>
</tr>
<tr>
<td>MQX_INVALID_POINTER</td>
<td><code>entry_ptr</code> is NULL.</td>
</tr>
</tbody>
</table>

See Also

- _log_create
- _log_write
- LOG_STRUCT
- LOG_ENTRY_STRUCT
Description

<table>
<thead>
<tr>
<th>read_type</th>
<th>Returns this entry in the log:</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_READ_NEWEST</td>
<td>Newest</td>
</tr>
<tr>
<td>LOG_READ_NEXT</td>
<td>Next one after the previous one read (must be used with LOG_READ_OLEST)</td>
</tr>
<tr>
<td>LOG_READ_OLEST</td>
<td>Oldest</td>
</tr>
<tr>
<td>LOG_READ_OLEST_AND_DELETE</td>
<td>Oldest and deletes it</td>
</tr>
</tbody>
</table>

Example

See _log_create().
2.1.71 _log_reset

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

Prototype

```c
#include <log.h>
_mqx_uint _log_reset(
    _mqx_uint log_number)
```

Parameters

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

Returns

- MQX_OK
- Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_DOES_NOT_EXIST</td>
<td><code>log_number</code> was not created.</td>
</tr>
<tr>
<td>LOG_INVALID</td>
<td><code>log_number</code> is out of range.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Log component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_HANDLE</td>
<td>Log component data is not valid.</td>
</tr>
</tbody>
</table>

See Also

`_log_disable, _log_enable`

Example

```c
_mqx_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

\begin{verbatim}
source\kernel\log.c
#include <log.h>
_mqx_uint _log_test(
    _mqx_uint *log\_error\_ptr)
\end{verbatim}

Parameters

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

Returns

See description

Traits

Disables and enables interrupts

See Also

\_log\_create\_component
\_log\_create

Description

<table>
<thead>
<tr>
<th>Return value</th>
<th>\textit{_log_error_ptr}</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_INVALID</td>
<td>Log number of the first invalid log</td>
<td>Information for a specific log is not valid</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>0</td>
<td>Log component data is not valid</td>
</tr>
<tr>
<td>MQX_OK</td>
<td>0</td>
<td>Log component data is valid</td>
</tr>
</tbody>
</table>

Example

\begin{verbatim}
_mqx_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;
}
...
\end{verbatim}
**2.1.73  _log_write**

Writes to the log.

**Prototype**

```c
#include <log.h>
_mqx_uint _log_write(
    _mqx_uint log_number,
    _mqx_uint num_of_parameters,
    _mqx_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.

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

**Example**

See `_log_create()`. 
2.1.74 _lwevent_clear

Clears the specified event bits in the lightweight event group.

Prototype

```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

```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**

```c
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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_LWEVENT_INVALID</td>
<td>Lightweight event group was not valid.</td>
</tr>
</tbody>
</table>

**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

```c
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

```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**

```c
#include <lwevent.h>
_mqx_uint lwevent_set_auto_clear(
    LWEVENT_STRUCT_PTR \_lwevent_group_ptr,
    _mqx_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

```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 (\texttt{NULL} if no error is found)

_td_error_ptr [OUT] — TD on the lightweight event in error (\texttt{NULL} if no error is found)

Returns

- MQX_OK
- Errors

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

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_ ...

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_lwevent_wait_for()</td>
<td>Wait for the specified lightweight event bits to be set in the lightweight event group: For the number of ticks (in tick time)</td>
</tr>
<tr>
<td>_lwevent_wait_ticks()</td>
<td>For the number of ticks</td>
</tr>
<tr>
<td>_lwevent_wait_until()</td>
<td>Until the specified time (in tick time)</td>
</tr>
</tbody>
</table>

### Prototype

```c
#include <lwevent.h>

_mqx_uint _lwevent_wait_for(
    LWEVENT_STRUCT_PTR event_ptr,
    _mqx_uint bit_mask,
    bool all,
    MQX_TICK_STRUCT_PTR tick_ptr)
```

```c
#include <lwevent.h>

_mqx_uint _lwevent_wait_ticks(
    LWEVENT_STRUCT_PTR event_ptr,
    _mqx_uint bit_mask,
    bool all,
    _mqx_uint timeout_in_ticks)
```

```c
#include <lwevent.h>

_mqx_uint _lwevent_wait_until(
    LWEVENT_STRUCT_PTR event_ptr,
    _mqx_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
MQX RTOS Functions and Macros

0 (unlimited wait)

Returns

- MQX_OK
- LWEVENT_WAIT_TIMEOUT (the time elapsed before an event signalled)
- Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_LWEVENT_INVALID</td>
<td>Lightweight event group is no longer valid or was never valid.</td>
</tr>
</tbody>
</table>

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

```c
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_lwlog_create()</td>
<td>Creates the lightweight log.</td>
</tr>
<tr>
<td>_lwlog_create_at()</td>
<td>Creates the lightweight log at the specific location.</td>
</tr>
</tbody>
</table>

Prototype

source\kernel\lwlog.c

_mqx_uint _lwlog_create(
    _mqx_uint log_number,
    _mqx_uint max_size,
    _mqx_uint flags)

source\kernel\lwlog.c

_mqx_uint _lwlog_create_at(
    _mqx_uint log_number,
    _mqx_uint max_size,
    _mqx_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 _lwlog_create_component()

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_EXISTS</td>
<td>Lightweight log with log number log_number exists.</td>
</tr>
<tr>
<td>LOG_INVALID</td>
<td>log_number is out of range.</td>
</tr>
<tr>
<td>LOG_INVALID_SIZE</td>
<td>max_size is 0.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_T_BASE</td>
<td>Data for the lightweight log component is not valid.</td>
</tr>
<tr>
<td>MQX_INVALID_POINTER</td>
<td>where is NULL.</td>
</tr>
</tbody>
</table>
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

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

Parameters

None

Returns

- MQX_OK
- Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>MQX RTOS is out of memory.</td>
</tr>
</tbody>
</table>

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(
    _mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_DOES_NOT_EXIST</td>
<td>$log_number$ was not previously created.</td>
</tr>
<tr>
<td>LOG_INVALID</td>
<td>$log_number$ is out of range.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Lightweight log component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID/component_HANDLE</td>
<td>Lightweight log component data is not valid.</td>
</tr>
</tbody>
</table>

Traits

Disables and enables interrupts

See Also

- _lwlog_create, _lwlog_create_at
- _lwlog_create_component
2.1.86  _lwlog_disable,  _lwlog_enable

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_lwlog_disable()</td>
<td>Stops logging to the lightweight log.</td>
</tr>
<tr>
<td>_lwlog_enable()</td>
<td>Starts logging to the lightweight log.</td>
</tr>
</tbody>
</table>

Prototype

```
#include <lwlog.h>
_mqx_uint  _lwlog_disable(
    _mqx_uint  log_number)

_mqx_uint  _lwlog_enable(
    _mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG DOES NOT_EXIST</td>
<td><code>log_number</code> was not created.</td>
</tr>
<tr>
<td>LOG INVALID</td>
<td><code>log_number</code> is out of range.</td>
</tr>
<tr>
<td>MQX_COMPONENT DOES NOT_EXIST</td>
<td>Lightweight log component is not created.</td>
</tr>
<tr>
<td>MQX INVALID COMPONENT HANDLE</td>
<td>Lightweight log component data is not valid.</td>
</tr>
</tbody>
</table>

See Also

- _lwlog_read
- _lwlog_reset
- _lwlog_write
2.1.87  _lwlog_read

Reads the information in the lightweight log.

Prototype

```c
#include <lwlog.h>
_mqx_uint _lwlog_read(
    _mqx_uint log_number,
    _mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_DOES_NOT_EXIST</td>
<td><code>log_number</code> was not created.</td>
</tr>
<tr>
<td>LOG_ENTRY_NOT_AVAILABLE</td>
<td>Log entry is not available.</td>
</tr>
<tr>
<td>LOG_INVALID</td>
<td><code>log_number</code> is out of range.</td>
</tr>
<tr>
<td>LOG_INVALID_READ_TYPE</td>
<td><code>read_type</code> is not valid.</td>
</tr>
<tr>
<td>MQX_Component_does_not_exist</td>
<td>Lightweight log component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_HANDLE</td>
<td>Lightweight log component data is not valid.</td>
</tr>
<tr>
<td>MQX_INVALID_POINTER</td>
<td><code>entry_ptr</code> is <code>NULL</code>.</td>
</tr>
</tbody>
</table>

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

```c
#include <lwlog.h>
_mqx_uint _lwlog_reset(_mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_DOES_NOT_EXIST</td>
<td><code>log_number</code> was not created.</td>
</tr>
<tr>
<td>LOG_INVALID</td>
<td><code>log_number</code> is out of range.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Log component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_HANDLE</td>
<td>Log component data is not valid.</td>
</tr>
</tbody>
</table>

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(
    _mqx_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

<table>
<thead>
<tr>
<th>Return value</th>
<th>*log_error_ptr</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_INVALID</td>
<td>Log number of the first invalid lightweight log</td>
<td>Information for a specific lightweight log is not valid</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>0</td>
<td>Lightweight log component data is not valid</td>
</tr>
<tr>
<td>MQX_OK</td>
<td>0</td>
<td>Lightweight log component data is valid</td>
</tr>
</tbody>
</table>
2.1.90 _lwlog_write

Writes to the lightweight log.

Prototype

```c
#include <lwlog.h>
_mqx_uint _lwlog_write(
    _mqx_uint log_number,
    _mqx_max_type p1,
    _mqx_max_type p2,
    _mqx_max_type p3,
    _mqx_max_type p4,
    _mqx_max_type p5,
    _mqx_max_type p6,
    _mqx_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 >= 10 (0 through 9 are reserved for MQX RTOS), data specified by `p2` through `p7` is written to kernel log.

Returns

- MQX_OK
- Errors

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

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  ...

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_lwmem_alloc()</td>
<td>Allocate this type of lightweight-memory block from the default memory pool</td>
</tr>
<tr>
<td>_lwmem_alloc_system()</td>
<td>System</td>
</tr>
<tr>
<td>_lwmem_alloc_system_zero()</td>
<td>System (zero-filled)</td>
</tr>
<tr>
<td>_lwmem_alloc_zero()</td>
<td>Private (zero-filled)</td>
</tr>
<tr>
<td>_lwmem_alloc_at()</td>
<td>Private (start address defined)</td>
</tr>
<tr>
<td>_lwmem_alloc_align()</td>
<td>Private (aligned)</td>
</tr>
<tr>
<td>_lwmem_alloc_system_align()</td>
<td>System (aligned)</td>
</tr>
</tbody>
</table>

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
MQX RTOS Functions and Macros

- Pointer to the lightweight-memory block (success)
- \textit{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 \texttt{\_task\_set\_error()} to set the task error code (see task error codes)

**See Also**
\texttt{\_lwmem\_create\_pool}
\texttt{\_lwmem\_free}
\texttt{\_lwmem\_get\_size}
\texttt{\_lwmem\_set\_default\_pool}
\texttt{\_lwmem\_transfer}
\texttt{\_lwmem\_alloc\_\_\_from}
\texttt{\_msg\_alloc}
\texttt{\_msg\_alloc\_system}
\texttt{\_task\_set\_error}

**Description**
The application must first set a value for the default lightweight-memory pool by calling
\texttt{\_lwmem\_set\_default\_pool()}.

The \texttt{\_lwmem\_alloc} functions allocate at least \texttt{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 \texttt{\_msg\_alloc()} or
\texttt{\_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:
- \texttt{\_lwmem\_alloc()}
- \texttt{\_lwmem\_alloc\_zero()}
- \texttt{\_lwmem\_alloc\_at()}
- \texttt{\_lwmem\_alloc\_align()}

Any task can free a lightweight-memory block that is allocated with one of the following functions:
- \texttt{\_lwmem\_alloc\_system()}

\textit{Freescale MQXTM RTOS Reference Manual, Rev. 19}
• `_lwmem_alloc_system_zero()
• `_lwmem_alloc_system_align()`
2.1.92 _lwmem_alloc_*_from

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_lwmem_alloc_from()</td>
<td>Allocate this type of lightweight-memory block from the specified lightweight-memory pool:</td>
</tr>
<tr>
<td>_lwmem_alloc_system_from()</td>
<td>System</td>
</tr>
<tr>
<td>_lwmem_alloc_system_zero_from()</td>
<td>System (zero-filled)</td>
</tr>
<tr>
<td>_lwmem_alloc_zero_from()</td>
<td>Private (zero-filled)</td>
</tr>
<tr>
<td>_lwmem_alloc_align_from()</td>
<td>Private (aligned)</td>
</tr>
<tr>
<td>_lwmem_alloc_system_align_from()</td>
<td>System (aligned)</td>
</tr>
</tbody>
</table>

Prototype

source/kernel/lwmem.c

```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

```c
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

```c
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)

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

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

```c
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**

```c
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

```c
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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CORRUPT_STORAGE_POOL</td>
<td>A memory pool pointer is not correct.</td>
</tr>
<tr>
<td>MQX_CORRUPT_STORAGE_POOL_FREE_LIST</td>
<td>Memory pool freelist is corrupted.</td>
</tr>
<tr>
<td>MQX_LWMEM_POOL_INVALID</td>
<td>Lightweight-memory pool is corrupted.</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>Errors/Task Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_INVALID_POINTER</td>
<td>block_ptr is NULL.</td>
</tr>
<tr>
<td>MQX_INVALID_TASK_ID</td>
<td>source or target does not represent a valid task.</td>
</tr>
<tr>
<td>MQX_NOTRESOURCE_OWNER</td>
<td>Block is not a resource of the task represented by source</td>
</tr>
</tbody>
</table>

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

```c
#include <lwmsgq.h>
#include <lwmsgq_prv.h>

_mqx_uint _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.

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(_mqx_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

```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 flags 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_ONEMPTY 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

```c
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

```c
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

```c
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

```c
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

```c
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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_INVALID_LWSEM</td>
<td>Results of _queue_test()</td>
</tr>
</tbody>
</table>

Traits

- Cannot be called from an ISR
- Disables and enables interrupts

See Also

- _lwsem_create
- _lwsem_destroy
- _queue_test

Example

See _lwsem_create().
MQX RTOS Functions and Macros

2.1.108 _lwsem_wait ...

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_lwsem_wait()</td>
<td>Wait (in FIFO order) for the lightweight semaphore: Until it is available</td>
</tr>
<tr>
<td>_lwsem_wait_for()</td>
<td>For the number of ticks (in tick time)</td>
</tr>
<tr>
<td>_lwsem_wait_ticks()</td>
<td>For the number of ticks</td>
</tr>
<tr>
<td>_lwsem_wait_until()</td>
<td>Until the specified time (in tick time)</td>
</tr>
</tbody>
</table>

Prototype

```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_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_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
MQX RTOS Functions and Macros

### Traits
- Might block the calling task
- Cannot be called from an ISR

### See Also
- `_lwsem_create`
- `_lwsem_post`
- `LWSEM_STRUCT`
- `MQX_TICK_STRUCT`

### Error

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_INVALID_LWSEM</td>
<td><code>sem_ptr</code> is for a lightweight semaphore that is not longer valid.</td>
</tr>
<tr>
<td>MQX_LWSEM_WAIT_TIMEOUT</td>
<td>Timeout expired before the task could get the lightweight semaphore.</td>
</tr>
</tbody>
</table>

### 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

```c
#include <lwtimer.h>

_mqx_uint _lwtimer_add_timer_to_queue(
    LWTIMER_PERIOD_STRUCT_PTR period_ptr,
    LWTIMER_STRUCT_PTR timer_ptr,
    _mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_LWTIMER_INVALID</td>
<td>period_ptr points to an invalid periodic queue.</td>
</tr>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td>ticks is greater than or equal to the periodic queue’s period.</td>
</tr>
</tbody>
</table>

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**

```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**

```c
#include <lwtimer.h>

_mqx_uint _lwtimer_create_periodic_queue(
    LWTIMER_PERIOD_STRUCT_PTR period_ptr,
    _mqx_uint period,
    _mqx_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

```c
#include <lwtimer.h>
_mxq_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)

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error from _queue_test()</td>
<td>A periodic queue or its queue was in error.</td>
</tr>
<tr>
<td>MQX_LWTIMER_INVALID</td>
<td>Invalid periodic queue.</td>
</tr>
</tbody>
</table>

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 ...

<table>
<thead>
<tr>
<th>Function</th>
<th>Type Block</th>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mem_alloc()</td>
<td>Private Default Memory Pool</td>
<td>Default Memory Pool</td>
</tr>
<tr>
<td>_mem_alloc_from()</td>
<td>Private Specified Memory Pool</td>
<td>Specified Memory Pool</td>
</tr>
<tr>
<td>_mem_alloc_system()</td>
<td>System Default Memory Pool</td>
<td>Default Memory Pool</td>
</tr>
<tr>
<td>_mem_alloc_system_from()</td>
<td>System Specified Memory Pool</td>
<td>Specified Memory Pool</td>
</tr>
<tr>
<td>_mem_alloc_system_zero()</td>
<td>System (zero-filled) Default Memory Pool</td>
<td>Default Memory Pool</td>
</tr>
<tr>
<td>_mem_alloc_system_zero_from()</td>
<td>System (zero-filled) Specified Memory Pool</td>
<td>Specified Memory Pool</td>
</tr>
<tr>
<td>_mem_alloc_zero()</td>
<td>Private (zero-filled) Default Memory Pool</td>
<td>Default Memory Pool</td>
</tr>
<tr>
<td>_mem_alloc_zero_from()</td>
<td>Private (zero-filled) Specified Memory Pool</td>
<td>Specified Memory Pool</td>
</tr>
<tr>
<td>_mem_alloc_align()</td>
<td>Private (aligned) Default Memory Pool</td>
<td>Default Memory Pool</td>
</tr>
<tr>
<td>_mem_alloc_align_from()</td>
<td>Private (aligned) Specified Memory Pool</td>
<td>Specified Memory Pool</td>
</tr>
<tr>
<td>_mem_alloc_at()</td>
<td>Private (start address defined) Default Memory Pool</td>
<td>Default Memory Pool</td>
</tr>
<tr>
<td>_mem_alloc_system_align()</td>
<td>System (aligned) Default Memory Pool</td>
<td>Default Memory Pool</td>
</tr>
<tr>
<td>_mem_alloc_system_align_from()</td>
<td>System (aligned) Specified Memory Pool</td>
<td>Specified Memory Pool</td>
</tr>
</tbody>
</table>

#### Prototype

```c
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)

<table>
<thead>
<tr>
<th>Task Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CORRUPT_STORAGE_POOL_FREE_LIST</td>
<td>Memory pool freelist is corrupted.</td>
</tr>
<tr>
<td>MQX_INVALID_CHECKSUM</td>
<td>Checksum of the current memory block header is incorrect.</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>MQX RTOS cannot find a block of the requested size.</td>
</tr>
<tr>
<td>MQX_INVALID_CONFIGURATION</td>
<td>User area not aligned on a cache line boundary.</td>
</tr>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td>Requested alignment is not power of 2.</td>
</tr>
</tbody>
</table>
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:
MQX RTOS Functions and Macros

- `_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.

```c
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

```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

```c
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

```c

#include "mem.h"

_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)

<table>
<thead>
<tr>
<th>Task error codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_MEM_POOL_TOO_SMALL</td>
<td>size is less than the minimum allowable message-pool size</td>
</tr>
<tr>
<td>MQX_CORRUPT_MEMORY_SYSTEM</td>
<td>Internal data for the message component is corrupted</td>
</tr>
</tbody>
</table>

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

```c
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.

```c
... _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

```c
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 * MIN_MEMORY_STORAGE_SIZE)`, as defined in `mem_prv.h`. 
2.1.119  _mem_free

Frees the memory block.

Prototype

```c
#include "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)

<table>
<thead>
<tr>
<th>Errors/Task Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_INVALID_CHECKSUM</td>
<td>Block's checksum is not correct, indicating that at least some of the block was overwritten.</td>
</tr>
<tr>
<td>MQX_INVALID_POINTER</td>
<td><code>mem_ptr</code> is NULL, not in the pool, or misaligned.</td>
</tr>
<tr>
<td>MQX_NOT_RESOURCE_OWNER</td>
<td>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.</td>
</tr>
</tbody>
</table>

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()`
MQX RTOS Functions and Macros

- _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

```c
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**

```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

```c
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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CORRUPT_STORAGE_POOL</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• <code>mem_ptr</code> does not point to a block that was allocated with a function</td>
</tr>
<tr>
<td></td>
<td>from the _mem_alloc family</td>
</tr>
<tr>
<td></td>
<td>• Memory is corrupted</td>
</tr>
<tr>
<td>MQX_INVALID_CHECKSUM</td>
<td>Checksum is not correct because part of the memory block header was</td>
</tr>
<tr>
<td></td>
<td>overwritten.</td>
</tr>
<tr>
<td>MQX_INVALID_POINTER</td>
<td><code>mem_ptr</code> is NULL or improperly aligned.</td>
</tr>
</tbody>
</table>

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

```c
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

```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

```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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CORRUPT_STORAGE_POOL</td>
<td>A memory pool pointer is not correct.</td>
</tr>
<tr>
<td>MQX_CORRUPT_STORAGE_POOL_FREE_LIST</td>
<td>Memory pool freelist is corrupted.</td>
</tr>
<tr>
<td>MQX_INVALID_CHECKSUM</td>
<td>Checksum of the current memory block header is incorrect (header is corrupted).</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors from _mem_test()</td>
<td>A memory pool has an error.</td>
</tr>
<tr>
<td>Errors from _queue_test()</td>
<td>Memory-pool queue has an error.</td>
</tr>
</tbody>
</table>

See Also

_mem_test
_mem_test_pool
_queue_test
2.1.131  _mem_test_and_set

Tests and sets a memory location.

**Prototype**

```c
<mqx_KSDK_DIR>/rtos/mqx/mqx/source/psp/cortex_m/core/M4/dispatch.S
_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**

```c
cchar 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
_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)

<table>
<thead>
<tr>
<th>Error / Task Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_INVALID_CHECKSUM</td>
<td>Block's checksum is not correct, indicating that at least some of the block was overwritten.</td>
</tr>
<tr>
<td>MQX_INVALID_POINTER</td>
<td><code>block_ptr</code> is NULL or misaligned.</td>
</tr>
<tr>
<td>MQX_INVALID_TASK_ID</td>
<td><code>source</code> or <code>target</code> does not represent a valid task.</td>
</tr>
<tr>
<td>MQX_NOTRESOURCEOWNER</td>
<td>Memory block is not a resource of the task represented by <code>source</code>.</td>
</tr>
</tbody>
</table>

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

```c
#include <psp.h>
_mqx_uint _mmu_add_vregion(
    void       *paddr,
    void       *vaddr,
    _mem_size  size,
    _mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>_mmu_vinit() was not previously called.</td>
</tr>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td>Incorrect input parameter.</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>Unable to allocate L2 table.</td>
</tr>
</tbody>
</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.

```c
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\vmmu_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

```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

```c
#include <psp.h>
_mqx_uint  _mmu_vinit(
    _mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>_cpu_type_initialize_support() was not previously called (see the PSP release note).</td>
</tr>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td>One or both of the following:</td>
</tr>
<tr>
<td></td>
<td>• L1 table points to NULL</td>
</tr>
<tr>
<td></td>
<td>• Address of L1 table is not aligned</td>
</tr>
<tr>
<td></td>
<td>• Invalid flags</td>
</tr>
</tbody>
</table>

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 Vyrid ARM® Cortex®-A5 processor.

```c
_mqx_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

```
#include <psp.h>
_mqx_uint  _mmu_vtop(
 void  *va,
 void  *pa)
```

Parameters

- `va [IN]` — Virtual address
- `pa [OUT]` — Physical address

Returns

- MQX_OK
- Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_INVALID_POINTER</td>
<td><code>vaddr</code> is invalid.</td>
</tr>
</tbody>
</table>

See Also

:_mmu_vinit:

Example

Get the physical address that corresponds to the virtual address of a DMA device.

```c
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)

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors from _int_install_isr()</td>
<td>MQX RTOS cannot install the interrupt subsystem.</td>
</tr>
<tr>
<td>Errors from _io_init()</td>
<td>MQX RTOS cannot install the I/O subsystem.</td>
</tr>
<tr>
<td>Errors from _mem_alloc_system()</td>
<td>There is not enough memory to allocate either the interrupt stack or the interrupt table.</td>
</tr>
<tr>
<td>Errors from _mem_alloc_zero()</td>
<td>There is not enough memory to allocate the ready queues.</td>
</tr>
<tr>
<td>MQX_KERNEL_MEMORY_TOO_SMALL</td>
<td><code>init_struct_ptr</code> does not specify enough kernel memory.</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>There is not enough memory to allocate either the ready queues, the interrupt stack, or the interrupt table.</td>
</tr>
<tr>
<td>MQX_TIMER_ISR_INSTALL_FAIL</td>
<td>MQX RTOS cannot install the periodic timer ISR.</td>
</tr>
</tbody>
</table>

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.

```c
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

```c
<stdio.h>
const char *_mqx_bsp_revision
```

See Also

_ mqx_copyright
_ mqx_date
_ mqx_generic_revision
_ mqx_version
_ mqx_psp_revision

Example

```c
puts(_mqx_bsp_revision);
```
2.1.142 _mqx_copyright

Pointer to the global MQX RTOS copyright string.

Prototype

```c
const char *_mqx_copyright
```

See Also

- _mqx_bsp_revision
- _mqx_date
- _mqx_generic_revision
- _mqx_version
- _mqx_psp_revision

Example

```c
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

```c
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

```c
#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

\[
\begin{align*}
\text{source/kernel/mqx.c} \\
\text{const char * _mqx_generic_revision}
\end{align*}
\]

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

```c
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

```c
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.

```c
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mqx_get_tad_data()</td>
<td>Gets the TAD_RESERVED field from the task descriptor.</td>
</tr>
<tr>
<td>_mqx_set_tad_data()</td>
<td>Sets the TAD_RESERVED field in the task descriptor.</td>
</tr>
</tbody>
</table>

Prototype

```c
void *mqx_get_tad_data(
  void *td)

void *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.

<table>
<thead>
<tr>
<th>CPU</th>
<th>Number of bits in the counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit</td>
<td>64</td>
</tr>
<tr>
<td>32-bit</td>
<td>128</td>
</tr>
</tbody>
</table>

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

```c
const char *_mqx_psp_revision
```

See Also

- _mqx_bsp_revision
- _mqx_copyright
- _mqx_date
- _mqx_generic_revision
- _mqx_version

Example

```c
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

```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.

```c
/* 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

```c
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

```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

```
#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)

<table>
<thead>
<tr>
<th>Task Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Message component is not created.</td>
</tr>
<tr>
<td>MSGPOOL_INVALID_POOL_ID</td>
<td><code>pool_id</code> is not valid.</td>
</tr>
<tr>
<td>MSGPOOL_OUT_OF_MESSAGES</td>
<td>All the messages in the pool are allocated.</td>
</tr>
<tr>
<td>Task error codes from <code>_mem_alloc_system()</code></td>
<td>(If MQX RTOS needs to grow the pool.)</td>
</tr>
</tbody>
</table>

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

```
#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)

<table>
<thead>
<tr>
<th>Task Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Message component is not created.</td>
</tr>
<tr>
<td>Task error codes from _mem_alloc_system()</td>
<td>(If MQX RTOS needs to grow the pool.)</td>
</tr>
</tbody>
</table>

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

```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

```c
#include <message.h>
_mqx_uint  _msg_create_component(void)
```

Parameters

None

Returns

- MQX_OK (success)
- Errors (failure)

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSGPOOL_POOL_NOT_CREATED</td>
<td>MQX RTOS cannot allocate the data structures for message pools.</td>
</tr>
<tr>
<td>MSGQ_TOO_MANY_QUEUES</td>
<td>MQX RTOS cannot allocate the data structures for message queues.</td>
</tr>
</tbody>
</table>

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:
MQX RTOS Functions and Macros

- `_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

```
#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

```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.

```c
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

Converting the message header to the other endian format.

Prototype

    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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSGPOOL_MESSAGE_SIZE_TOO_SMALL</td>
<td><code>message_size</code> is less than the size of the message header structure</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>MQX RTOS cannot allocate memory to create the message pool</td>
</tr>
<tr>
<td>MSGPOOL_OUT_OF_POOLS</td>
<td>Maximum number of message pools have been created, where the number is</td>
</tr>
<tr>
<td></td>
<td>defined at initialization time in MAX_MSGPOOLS in the MQX RTOS initialization structure</td>
</tr>
<tr>
<td>Task error codes from _mem_alloc_system()</td>
<td>—</td>
</tr>
<tr>
<td>Task error codes from _msg_create_component()</td>
<td>—</td>
</tr>
</tbody>
</table>

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.

```c
_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

```c
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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSGPOOL_ALL_MESSAGES_NOT_FREE</td>
<td>All messages in the message pool have not been freed.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Message component is not created.</td>
</tr>
<tr>
<td>MSGPOOL_INVALID_POOL_ID</td>
<td><code>pool_id</code> does not represent a message pool that was created by _msgpool_create().</td>
</tr>
</tbody>
</table>

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

\begin{verbatim}
source\kernel\msgpool.c
#include <message.h>
_mqx_uint _msgpool_test(
    void *pool_error_ptr,
    void *msg_error_ptr)
\end{verbatim}

Parameters

\emph{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

\emph{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

\begin{table}
<table>
<thead>
<tr>
<th>Errors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Message component is not created.</td>
</tr>
<tr>
<td>MSGQ_INVALID_MESSAGE</td>
<td>At least one message in at least one message pool failed.</td>
</tr>
</tbody>
</table>
\end{table}

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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Message component is not created.</td>
</tr>
<tr>
<td>MSGQ_INVALID_QUEUE_ID</td>
<td><code>queue_id</code> is not valid for this processor.</td>
</tr>
<tr>
<td>MSGQ_NOT_QUEUE_OWNER</td>
<td>Task that got <code>queue_id</code> did so by opening a private message queue (_msgq_open()) and is not the task calling _msgq_close().</td>
</tr>
<tr>
<td>MSGQ_QUEUE_IS_NOT_OPEN</td>
<td><code>queue_id</code> does not represent a queue that is open.</td>
</tr>
<tr>
<td>Task error codes from _msg_free()</td>
<td>(If MQX RTOS cannot free messages that are in the queue.)</td>
</tr>
</tbody>
</table>

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

```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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Message component is not created.</td>
</tr>
<tr>
<td>MSGQ_INVALID_QUEUE_ID</td>
<td><code>queue_id</code> is not valid for this processor.</td>
</tr>
<tr>
<td>MSGQ_QUEUE_IS_NOT_OPEN</td>
<td><code>queue_id</code> does not represent a message queue that is open.</td>
</tr>
</tbody>
</table>

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

```
#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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSGQ_INVALID_QUEUE_ID</td>
<td><code>queue_id</code> does not represent a valid message queue on this processor.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Message component is not created.</td>
</tr>
<tr>
<td>MSGQ_QUEUE_IS_NOT_OPEN</td>
<td><code>queue_id</code> does not represent an open message queue.</td>
</tr>
</tbody>
</table>

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

```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

<table>
<thead>
<tr>
<th>Task Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Message component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_PROCESSOR_NUMBER</td>
<td>Processor number that <code>queue_id</code> specifies is not valid.</td>
</tr>
<tr>
<td>MSGQ_QUEUE_IS_NOT_OPEN</td>
<td>Message queue with queue ID <code>queue_id</code> is not open.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Task Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSGQ_INVALID_QUEUE_NUMBER</td>
<td><code>queue_number</code> is out of range</td>
</tr>
<tr>
<td>MSGQ_QUEUE_IN_USE</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• message queue is already open</td>
</tr>
<tr>
<td></td>
<td>• MQX RTOS cannot get a queue number for an unopened queue</td>
</tr>
</tbody>
</table>

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

```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

<table>
<thead>
<tr>
<th>Task Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Message component is not created.</td>
</tr>
<tr>
<td>MSGQ_MESSAGE NOT AVAILABLE</td>
<td>There are no messages in the message queue.</td>
</tr>
<tr>
<td>MSGQ_NOT_QUEUE_OWNER</td>
<td>Task is not the owner of the private message queue.</td>
</tr>
<tr>
<td>MSGQ_QUEUE_IS_NOT_OPEN</td>
<td>Queue is not open.</td>
</tr>
</tbody>
</table>

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
MQX RTOS Functions and Macros

_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

```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

<table>
<thead>
<tr>
<th>Task Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Message component is not created.</td>
</tr>
<tr>
<td>MSGQ_INVALID_QUEUE_ID</td>
<td>queue_id is not valid.</td>
</tr>
<tr>
<td>MSGQ_MESSAGE_NOT_AVAILABLE</td>
<td>There are no messages in the message queue.</td>
</tr>
<tr>
<td>MSGQ_NOT_QUEUE_OWNER</td>
<td>Task is not the owner of the private message queue.</td>
</tr>
<tr>
<td>MSGQ_QUEUE_IS_NOT_OPEN</td>
<td>Queue is not open.</td>
</tr>
</tbody>
</table>

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

```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

<table>
<thead>
<tr>
<th>Task Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Message component is not created.</td>
</tr>
<tr>
<td>MSGQ_INVALID_QUEUE_ID</td>
<td><code>queue_id</code> is not valid or is not on this processor.</td>
</tr>
<tr>
<td>MSGQ_MESSAGE_NOT_AVAILABLE</td>
<td>There are no messages in the message queue.</td>
</tr>
<tr>
<td>MSGQ_NOT_QUEUE_OWNER</td>
<td>Queue is a private message queue that the task does not own.</td>
</tr>
<tr>
<td>MSGQ_QUEUE_IS_NOT_OPEN</td>
<td>Queue is not open.</td>
</tr>
</tbody>
</table>

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

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

```c
#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  …

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_msgq_receive()</td>
<td>Wait for a message from the private message queue: For the number of milliseconds</td>
</tr>
<tr>
<td>_msgq_receive_for()</td>
<td>For the number of ticks (in tick time)</td>
</tr>
<tr>
<td>_msgq_receive_ticks()</td>
<td>For the number of ticks</td>
</tr>
<tr>
<td>_msgq_receive_until()</td>
<td>Until the specified time (in tick time)</td>
</tr>
</tbody>
</table>

Prototype

```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

<table>
<thead>
<tr>
<th>Task Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Message component is not created.</td>
</tr>
<tr>
<td>MSGQ_INVALID_QUEUE_ID</td>
<td>queue_id is for a specific queue, but the ID is not valid.</td>
</tr>
<tr>
<td>MSGQ_MESSAGE_NOT_AVAILABLE</td>
<td>No messages were in the message queue before the timeout expired.</td>
</tr>
<tr>
<td>MSGQ_NOT_QUEUE_OWNER</td>
<td>Message is not a resource of the task.</td>
</tr>
</tbody>
</table>
| MSGQ_QUEUE_IS_NOT_OPEN                           | One of the following:
    • specific queue is not open
    • queue_id is MSGQ_ANY_QUEUE, but the task has no queues open |

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

```c
#include <message.h>
bool _msgq_send(
    void *msg_ptr)
```

Parameters

- `msg_ptr` — Pointer to the message to be sent

Returns

- TRUE (success: see description)
- FALSE (failure)

Task error codes

<table>
<thead>
<tr>
<th>Task error code</th>
<th>Meaning</th>
<th>Msg. accepted</th>
<th>Msg. freed</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Message component is not created</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
| MSGQ_INVALID_MESSAGE | `msg_ptr` is NULL or points to a message that is one of:  
  - not valid  
  - on a message queue  
  - free | 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 _msg_free() | (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 function returns successfully, the message is no longer a resource of the task.

Example

```c
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);
    }
}
```

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 message is for a message queue on: MQX RTOS sends the message to:

- Local processor: The message queue
- Remote processor: The remote processor
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

<table>
<thead>
<tr>
<th>Task Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td><code>qid_ptr</code> does not point to a valid queue ID.</td>
</tr>
<tr>
<td>MSGPOOL_OUT_OF_MESSAGES</td>
<td>MQX RTOS could not allocate a message from the message pool.</td>
</tr>
<tr>
<td>MSGQ_INVALID_MESSAGE</td>
<td><code>msg_ptr</code> does not point to a message that was allocated as described for <code>_msgq_send()</code>.</td>
</tr>
</tbody>
</table>

- 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

```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.

```c
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.

```c
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

```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

```
#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

<table>
<thead>
<tr>
<th>Return value</th>
<th>Meaning</th>
<th>Notification function installed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to the previous notification function</td>
<td>Success</td>
<td>Yes</td>
</tr>
<tr>
<td>NULL</td>
<td>Success: Previous notification function was NULL</td>
<td>Yes</td>
</tr>
<tr>
<td>NULL</td>
<td>Failure</td>
<td>No</td>
</tr>
</tbody>
</table>

Task Error Codes

<table>
<thead>
<tr>
<th>Task Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_OK</td>
<td>Notification function is installed; the previous function was NULL.</td>
</tr>
<tr>
<td>MSGQ_INVALID_QUEUE_ID</td>
<td><code>qid</code> is not valid.</td>
</tr>
<tr>
<td>MSGQ_QUEUE_IS_NOT_OPEN</td>
<td>Queue is not open.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Message component is not created.</td>
</tr>
</tbody>
</table>

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**

```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.

```c
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

```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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mutatr_get_priority_ceiling()</td>
<td>Gets the priority value of the mutex attributes structure.</td>
</tr>
<tr>
<td>_mutatr_set_priority_ceiling()</td>
<td>Sets the priority value of the mutex attributes structure.</td>
</tr>
</tbody>
</table>

Prototype

```
#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_EINV (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("Priority ceiling is %ld", priority);
    result = _mutex_init(&mutex, &mutex_attributes);
    result = _mutex_destroy(&mutex_attributes);
    if (result != MQX_EOK) {
        /* Could not initialize the mutex. */
    }
}
2.1.190  _mutatr_get_sched_protocol, _mutatr_set_sched_protocol

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mutatr_get_sched_protocol</td>
<td>Gets the scheduling protocol of the mutex attributes structure.</td>
</tr>
<tr>
<td>_mutatr_set_sched_protocol</td>
<td>Sets the scheduling protocol of the mutex attributes structure.</td>
</tr>
</tbody>
</table>

Prototype

```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

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUTEX_PRIO_INHERIT</td>
<td>(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.</td>
</tr>
</tbody>
</table>
Example

See `_mutatr_get_priority_ceiling()`.
2.1.191 _mutatr_get_spin_limit, _mutatr_set_spin_limit

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mutatr_get_spin_limit()</td>
<td>Gets the spin limit of the mutex attributes structure.</td>
</tr>
<tr>
<td>_mutatr_set_spin_limit()</td>
<td>Sets the spin limit of the mutex attributes structure.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mutatr_get_wait_protocol()</td>
<td>Gets the waiting policy of the mutex attributes structure.</td>
</tr>
<tr>
<td>_mutatr_set_wait_protocol()</td>
<td>Sets the waiting policy of the mutex attributes structure.</td>
</tr>
</tbody>
</table>

Prototype

```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

<table>
<thead>
<tr>
<th>Waiting Protocols</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUTEX_SPIN_ONLY</td>
<td>If the mutex is already locked, MQX RTOS timeslices the task until another task unlocks the mutex.</td>
</tr>
<tr>
<td>MUTEX_LIMITED_SPIN</td>
<td>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.</td>
</tr>
<tr>
<td>MUTEX_QUEUEING</td>
<td>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.</td>
</tr>
<tr>
<td>MUTEX_PRIORITY_QUEUEING</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

\textit{attr_ptr} \textbf{[IN]} — Pointer to the mutex attributes structure to initialize

Returns

- MQX_EOK (success)
- MQX_EINVAL (failure: attr_ptr is \texttt{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.

<table>
<thead>
<tr>
<th>Mutex attribute</th>
<th>Field in MUTEX_ATTR_STRUCT</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduling protocol</td>
<td>POLICY</td>
<td>MUTEX_NO_PRIO_INHERIT</td>
</tr>
<tr>
<td>--</td>
<td>VALID</td>
<td>TRUE</td>
</tr>
<tr>
<td>Priority</td>
<td>PRIORITY</td>
<td>0</td>
</tr>
<tr>
<td>Spin limit</td>
<td>COUNT</td>
<td>0</td>
</tr>
<tr>
<td>Waiting protocol</td>
<td>WAITING_POLICY</td>
<td>MUTEX_QUEUEING</td>
</tr>
</tbody>
</table>

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
    _mutex_groups_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

```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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_EINVAL</td>
<td><code>mutex_ptr</code> does not point to a valid mutex (mutex is locked).</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Mutex component data is not valid.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Mutex component is not created.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mutex_get_priority_ceiling()</td>
<td>Gets the priority of the mutex.</td>
</tr>
<tr>
<td>_mutex_set_priority_ceiling()</td>
<td>Sets the priority of the mutex.</td>
</tr>
</tbody>
</table>

Prototype

```
#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("Current 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

```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

```
#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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_EINVAL</td>
<td>One of the following:</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Mutex component data is not valid.</td>
</tr>
<tr>
<td>MQX_INVALID_ATTRIBUTE_BASE</td>
<td>Wild pointer.</td>
</tr>
</tbody>
</table>

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

#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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_EBUSY</td>
<td>Mutex is already locked.</td>
</tr>
<tr>
<td>MQX_EDEADLK</td>
<td>Task already has the mutex locked.</td>
</tr>
</tbody>
</table>
| MQX_EINVAL                     | One of the following:
  • mutex_ptr is NULL
  • mutex was destroyed          |

Traits

• Might block the calling task
• Cannot be called from an ISR

See Also

(mutex_init
_mutex_try_lock
_mutex_unlock
_mutex_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

<table>
<thead>
<tr>
<th>Return value</th>
<th>Meaning</th>
<th>mutex_error_ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_OK</td>
<td>No errors were found</td>
<td>NULL</td>
</tr>
<tr>
<td>MQX_CORRUPT_QUEUE</td>
<td>Queue of mutexes is not valid</td>
<td>Pointer to the invalid queue</td>
</tr>
</tbody>
</table>
| MQX EINVAL           | One of:
  - a mutex is not valid
  - a mutex queue is not valid | Pointer to the mutex with the error |
| MQX_INVALID_COMPONENT_BASE | Mutex component data is not valid | NULL            |
Example

```c
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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_EBUSY</td>
<td>Mutex is currently locked.</td>
</tr>
<tr>
<td>MQX_EDeadlk</td>
<td>Task already has the mutex locked.</td>
</tr>
<tr>
<td>MQX EINVAL</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• mutex_ptr is NULL</td>
</tr>
<tr>
<td></td>
<td>• mutex has been destroyed</td>
</tr>
</tbody>
</table>

See Also

(mutex_create_component
(mutex_init
(mutex_lock
(mutex_unlock
(mutexr_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

```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
- _mutattr_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

```c
#define _mqx_uint  _name_add(  
char           *name,  
_mqx_max_type   number)  

Parameters

name [IN] — Name to add
number [IN] — Number to be associated with the name

Returns

- MQX_OK
- Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Name component data is not valid.</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>MQX RTOS cannot allocate memory for the name</td>
</tr>
<tr>
<td></td>
<td>component.</td>
</tr>
<tr>
<td>NAME_EXISTS</td>
<td>Name is already in the names database.</td>
</tr>
<tr>
<td>NAME_TABLE_FULL</td>
<td>Names database is full.</td>
</tr>
<tr>
<td>NAME_TOO_LONG</td>
<td>Name is longer than NAME_MAX_NAME_SIZE.</td>
</tr>
<tr>
<td>NAME_TOO_SHORT</td>
<td>Name is \0.</td>
</tr>
</tbody>
</table>

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

```c
#include <name.h>
_mqx_uint _name_create_component(
    _mqx_uint initial_number,
    _mqx_uint grow_number,
    _mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_OK</td>
<td>Success; one of:</td>
</tr>
<tr>
<td></td>
<td>• name component is created</td>
</tr>
<tr>
<td></td>
<td>• name component was already created</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>Failure: MQX RTOS cannot allocate memory for the name component.</td>
</tr>
</tbody>
</table>

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().

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial_number</td>
<td>8</td>
</tr>
</tbody>
</table>
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

```
#include <name.h>
_mqx_uint  _name_delete(
    char *name)
```

Parameters

- `name [IN]` — Name to delete

Returns

- MQX_OK
- Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Name component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Name component data is not valid.</td>
</tr>
<tr>
<td>NAME_NOT_FOUND</td>
<td>Name is not in the names database.</td>
</tr>
</tbody>
</table>

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

```c
#include <name.h>
_mqx_uint _name_find(
    char *name,
    _mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Name component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Name component data is not valid.</td>
</tr>
<tr>
<td>NAME_NOT_FOUND</td>
<td>Name is not in the names database.</td>
</tr>
</tbody>
</table>

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

```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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Name component data is not valid.</td>
</tr>
<tr>
<td>NAME_NOT_FOUND</td>
<td>Number is not in the names database.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Return</th>
<th>base_error_ptr</th>
<th>ext_error_ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CORRUPT_QUEUE</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>(Task queue that is associated with the</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name component is incorrect)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Pointer to the</td>
<td></td>
</tr>
<tr>
<td>(MQX RTOS found an error in a name</td>
<td>name table</td>
<td></td>
</tr>
<tr>
<td>component data structure)</td>
<td>that has an</td>
<td></td>
</tr>
<tr>
<td></td>
<td>error</td>
<td></td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_partition_alloc()</td>
<td>Allocates a private partition block from the partition.</td>
</tr>
<tr>
<td>_partition_alloc_zero()</td>
<td>Allocates a zero-filled private partition block from the partition.</td>
</tr>
</tbody>
</table>

**Prototype**

```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**

<table>
<thead>
<tr>
<th>Task Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTITION_BLOCK_INVALID_CHECKSUM</td>
<td>MQX RTOS found an incorrect checksum in the partition block header.</td>
</tr>
<tr>
<td>PARTITION_INVALID partition_id</td>
<td><code>partition_id</code> does not represent a valid partition.</td>
</tr>
<tr>
<td>PARTITION_OUT_OF_BLOCKS</td>
<td>All the partition blocks in the partition are allocated (for static partitions only).</td>
</tr>
<tr>
<td>Task error code set by _mem_alloc_system()</td>
<td>MQX RTOS cannot allocate memory for the partition block (for dynamic partitions only).</td>
</tr>
</tbody>
</table>

**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 ...

---

**Freescale MQX™ RTOS Reference Manual, Rev. 19**

Freescale Semiconductor, Inc.
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.

```c
#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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_partition_alloc_system()</td>
<td>Allocates a system partition block from the partition.</td>
</tr>
<tr>
<td>_partition_alloc_system_zero()</td>
<td>Allocates a zero-filled system partition block from the partition.</td>
</tr>
</tbody>
</table>

Prototype

```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

```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

```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

```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

```c
#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

```c
#include <partition.h>
_mqx_uint  _partition_create_component(void)
```

Parameters

None

Returns

- MQX_OK (success)
- Errors (failure)

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>MQX RTOS is out of memory.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors from _mem_free()</td>
<td></td>
</tr>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td><code>partition_id</code> is invalid.</td>
</tr>
<tr>
<td>PARTITION_ALL_BLOCKS_NOT_FREE</td>
<td>There are allocated partition blocks in the partition.</td>
</tr>
<tr>
<td>PARTITION_INVALID_TYPE</td>
<td>Partition is not a dynamic partition.</td>
</tr>
</tbody>
</table>

**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

```
#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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• partition_size is 0</td>
</tr>
<tr>
<td></td>
<td>• partition_id does not represent a static partition</td>
</tr>
<tr>
<td>PARTITION_INVALID</td>
<td><code>partition_id</code> does not represent a valid partition.</td>
</tr>
</tbody>
</table>

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

```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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_NOTRESOURCE_OWNER</td>
<td>Task is not the one that owns the partition block.</td>
</tr>
<tr>
<td>PARTITION_BLOCK_INVALID_CHECKSUM</td>
<td>Checksum in the partition block header is not correct; the</td>
</tr>
<tr>
<td></td>
<td>integrity of the partition is in question.</td>
</tr>
<tr>
<td>PARTITION_INVALID mem_ptr</td>
<td><code>mem_ptr</code> is part of a partition that is not valid.</td>
</tr>
</tbody>
</table>

See Also

- _partition_alloc, _partition_alloc_zero
- _partition_alloc_system, _partition_alloc_system_zero
- _partition_create

Description

<table>
<thead>
<tr>
<th>If the partition block was allocated by:</th>
<th>It can be freed by:</th>
</tr>
</thead>
<tbody>
<tr>
<td>_partition_alloc() or</td>
<td>Task that allocated it</td>
</tr>
<tr>
<td>_partition_alloc_zero()</td>
<td></td>
</tr>
<tr>
<td>_partition_alloc_system() or</td>
<td>Any task</td>
</tr>
<tr>
<td>_partition_alloc_system_zero()</td>
<td></td>
</tr>
</tbody>
</table>

Example

See _partition_alloc().
2.1.219 _partition_get_block_size

Gets the size of the partition blocks in the partition.

Prototype

```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 that what was specified when the partition was created.

Example

Print the attributes of a partition.

```c
#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**

```
#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

#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

```
#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**

```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_INV AID — `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

```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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTITION_BLOCK_INVALID_CHECKSUM</td>
<td>MQX RTOS found a partition block with an incorrect checksum.</td>
</tr>
<tr>
<td>PARTITION_INVALID</td>
<td>MQX RTOS found an invalid partition.</td>
</tr>
</tbody>
</table>

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

```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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTITION_BLOCK_INVALID_CHECKSUM</td>
<td>Checksum of the partition block header is not correct, which indicates that <code>mem_ptr</code> might not point to a valid partition block.</td>
</tr>
<tr>
<td>PARTITION_INVALID_TASK_ID</td>
<td><code>task_id</code> is not valid.</td>
</tr>
</tbody>
</table>

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

```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

```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

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

```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

```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.

```c
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

```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

```c
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_ptr [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

```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.

```c
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

```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

```c
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Get the time slice in:</th>
</tr>
</thead>
<tbody>
<tr>
<td>_sched_get_rr_interval()</td>
<td>Milliseconds</td>
</tr>
<tr>
<td>_sched_get_rr_interval_ticks()</td>
<td>Tick time</td>
</tr>
</tbody>
</table>

**Prototype**

```c
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**

\textit{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)

\textit{ms\_ptr [OUT]} — Pointer to the time slice (in milliseconds)

\textit{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 — \textit{time\_ptr} is \textit{NULL}.
- MQX\_SCHED\_INVALID\_TASK\_ID — \textit{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**

```c
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Set the time slice in:</th>
</tr>
</thead>
<tbody>
<tr>
<td>_sched_set_rr_interval()</td>
<td>Milliseconds</td>
</tr>
<tr>
<td>_sched_set_rr_interval_ticks()</td>
<td>Tick time</td>
</tr>
</tbody>
</table>

Prototype

```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.

```c
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

```c
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.

```c
_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

```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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error code from _mem_free()</td>
<td>Task is not the one that opened the connection.</td>
</tr>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>SEM_INVALID_SEMAPHORE_HANDLE</td>
<td>• sem_handle is not a valid semaphore connection</td>
</tr>
<tr>
<td></td>
<td>• semaphore is no longer valid</td>
</tr>
</tbody>
</table>

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

```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)

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Semaphore component was not created and cannot be created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Semaphore component data is not valid.</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>MQX cannot allocate memory for the semaphore.</td>
</tr>
<tr>
<td>SEM_INCORRECT_INITIAL_COUNT</td>
<td><code>sem_count</code> cannot be 0 if SEM STRICT is set.</td>
</tr>
<tr>
<td>SEM_INVALID_POLICY</td>
<td>SEM STRICT must be set if SEM_PRIORITY_INHERITANCE is set.</td>
</tr>
<tr>
<td>SEM_SEMAPHORE_EXISTS</td>
<td>Semaphore with the name exists.</td>
</tr>
<tr>
<td>SEM_SEMAPHORE_TABLE_FULL</td>
<td>Semaphore names database is full and cannot be expanded.</td>
</tr>
</tbody>
</table>

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().

<table>
<thead>
<tr>
<th>Bit flag</th>
<th>Set</th>
<th>Not set</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEM_PRIORITY_INHERITANCE</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>(SEM_STRICT must also be set)</td>
<td></td>
<td>MQX RTOS does not boost priorities</td>
</tr>
<tr>
<td>SEM_PRIORITY_QUEUEING</td>
<td>Task that waits for the semaphore is queued according to the task's priority. Within a priority, tasks are in FIFO order.</td>
<td>Task that waits for the semaphore is queued in FIFO order</td>
</tr>
<tr>
<td>SEM_STRICT</td>
<td>• Task must wait for the semaphore before it can post the semaphore</td>
<td>Task need not wait before posting</td>
</tr>
<tr>
<td></td>
<td>• sem_count must be greater than or equal to 1</td>
<td>sem_count must be greater than or equal to 0</td>
</tr>
</tbody>
</table>

Example

See _sem_create_component().
2.1.245 _sem_create_component

Creates the semaphore component.

Prototype

```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 that 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()`.
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. */
}

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial_number</td>
<td>8</td>
</tr>
<tr>
<td>grow_number</td>
<td>8</td>
</tr>
<tr>
<td>maximum_number</td>
<td>0 (unlimited)</td>
</tr>
</tbody>
</table>
2.1.246 _sem_create_fast

Creates the fast semaphore.

Prototype

```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

#### Description

* _sem_destroy() *
  Destroys the named semaphore.

* _sem_destroy_fast() *
  Destroys the fast semaphore.

#### Prototype

```c
#include <sem.h>

_mqx_uint _sem_destroy(
    char *name,
    bool force_destroy)

_mqx_uint _sem_destroy_fast(
    _mqx_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Semaphore component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Semaphore component data is not valid.</td>
</tr>
<tr>
<td>SEM_INVALID_SEMAPHORE</td>
<td>Semaphore data that is associated with <code>name</code> or <code>index</code> is not valid.</td>
</tr>
<tr>
<td>SEM_SEMAPHORE_NOT_FOUND</td>
<td><code>name</code> or <code>index</code> is not in the semaphore names database.</td>
</tr>
</tbody>
</table>

#### Traits

Cannot be called from an ISR

#### See Also

- `_sem_close`
- `_sem_create`
- `_sem_create_fast`

---

**Freescale MQSTM RTOS Reference Manual, Rev. 19**

Freescale Semiconductor, Inc.
Description

<table>
<thead>
<tr>
<th>force_destroy is TRUE</th>
<th>force_destroy is FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Tasks that are waiting for the semaphore are readied.</td>
<td>• Semaphore is destroyed after the last waiting task gets and posts the semaphore.</td>
</tr>
<tr>
<td>• Semaphore is destroyed after all the owners post the semaphore.</td>
<td>• This is the action if the semaphore is strict.</td>
</tr>
</tbody>
</table>

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

```
#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

<table>
<thead>
<tr>
<th>Task Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEM_INVALID_SEMAPHORE</td>
<td><code>sem_ptr</code> does not point to a valid semaphore.</td>
</tr>
<tr>
<td>SEM_INVALID_SEMAPHORE_HANDLE</td>
<td><code>sem_ptr</code> is not a valid semaphore handle.</td>
</tr>
</tbody>
</table>

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

```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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Semaphore component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Semaphore component data is not valid.</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>MQX RTOS cannot allocate memory for the connection.</td>
</tr>
<tr>
<td>SEM_INVALID_SEMAPHORE</td>
<td>Data that is associated with the semaphore is not valid.</td>
</tr>
<tr>
<td>SEM_SEMAPHORE_DELETED</td>
<td>Semaphore is in the process of being destroyed.</td>
</tr>
<tr>
<td>SEM_SEMAPHORE_NOT_FOUND</td>
<td>name is not in the semaphore names database.</td>
</tr>
</tbody>
</table>

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**

```
#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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>SEM_CANNOT_POST</td>
<td>Semaphore is strict and the task has not first waited for the semaphore.</td>
</tr>
<tr>
<td>SEM_INVALID_SEMAPHORE</td>
<td><em>sem_handle</em> represents a semaphore that is no longer valid.</td>
</tr>
<tr>
<td>SEM_INVALID_SEMAPHORE_COUNT</td>
<td>Semaphore data is corrupted.</td>
</tr>
<tr>
<td>SEM_INVALID_SEMAPHORE_HANDLE</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>- sem_handle is not a valid semaphore handle</td>
</tr>
<tr>
<td></td>
<td>- semaphore is strict and sem_handle was obtained by another task</td>
</tr>
</tbody>
</table>

**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)

<table>
<thead>
<tr>
<th>Error</th>
<th>MQX RTOS found an error in:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CORRUPT_QUEUE</td>
<td>A semaphore queue</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Semaphore component data</td>
</tr>
<tr>
<td>SEM_INVALID_SEMAPHORE</td>
<td>Semaphore data</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Wait Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>_sem_wait()</td>
<td>For the number of milliseconds</td>
</tr>
<tr>
<td>_sem_wait_for()</td>
<td>For the number of ticks (in tick time)</td>
</tr>
<tr>
<td>_sem_wait_ticks()</td>
<td>For the number of ticks</td>
</tr>
<tr>
<td>_sem_wait_until()</td>
<td>Until the specified time (in tick time)</td>
</tr>
</tbody>
</table>

### Prototype

```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,  
    _mqx_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

- Wait for the semaphore:
  - _sem_wait(): Returns the number of milliseconds remaining before the semaphore was signalled.
  - _sem_wait_for(): Returns 0 if the semaphore was signalled within the timeout period; otherwise, it returns the remaining number of milliseconds before the timeout.
  - _sem_wait_ticks(): Returns 0 if the semaphore was signalled within the timeout period; otherwise, it returns the remaining number of ticks.
  - _sem_wait_until(): Returns 0 if the semaphore was signalled by the time specified; otherwise, it returns the remaining time (in tick time).
MQX RTOS Functions and Macros

- MQX_OK
- Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_EDEADLK</td>
<td>Function was aborted to prevent deadlock: the task has all the semaphore</td>
</tr>
<tr>
<td></td>
<td>locks and, since the semaphore is strict, the task cannot post to “wake”</td>
</tr>
<tr>
<td></td>
<td>itself.</td>
</tr>
<tr>
<td>SEM_INVALID_SEMAPHORE</td>
<td>sem_handle is for a semaphore that is no longer valid.</td>
</tr>
<tr>
<td>SEM_INVALID_SEMAPHORE_HANDLE</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• sem_handle is not a valid semaphore handle</td>
</tr>
<tr>
<td></td>
<td>• sem_handle was obtained by another task</td>
</tr>
<tr>
<td>SEM_SEMAPHORE_DELETED</td>
<td>MQX RTOS is in the process of destroying the semaphore.</td>
</tr>
<tr>
<td>SEM_WAIT_TIMEOUT</td>
<td>Timeout expired before the task can get the semaphore.</td>
</tr>
</tbody>
</table>

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

    _strnlen
2.1.255  _strnlen

Gets the length of the length-limited string.

Prototype

```
#include "string/strnlen.c"

_mqx_uint _strnlen(
    char       *string_ptr
    _mqx_uint  max_length)
```

Parameters

- `string_ptr [IN]` — Pointer to the string
- `max_length [OUT]` — Maximum number of 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**

```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.

```c
#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

```c
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

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.

Returns

- Task ID of the child task (success)
- MQX_NULL_TASK_ID (failure)
Task Error Codes

<table>
<thead>
<tr>
<th>Task Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_INVALID_PROCESSOR_NUMBER</td>
<td>processor_number is not one of the allowed processor numbers.</td>
</tr>
<tr>
<td>MQX_NO_TASK_TEMPLATE</td>
<td>template_index is not in the task template list.</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>MQX RTOS cannot allocate memory for the task data structures.</td>
</tr>
</tbody>
</table>

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_INITIALIZATIONSTRUCT`  
`TASK_TEMPLATE_STRUCT`  

Example

Create an instance of Receiver task.

```c
#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 */
    ...
```

Freescale MQX™ RTOS Reference Manual, Rev. 19
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

```c
source\kernel\task.c
void _task_disable_fp(void)
void _task_enable_fp(void)
```

Traits

Changes context information that MQX RTOS stores

Description

<table>
<thead>
<tr>
<th>Function</th>
<th>When MQX RTOS performs a context switch, floating-point registers are saved and restored?</th>
</tr>
</thead>
<tbody>
<tr>
<td>_task_disable_fp()</td>
<td>No</td>
</tr>
<tr>
<td>_task_enable_fp()</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example

Task is about to do some floating-point work, so change the type of context switch.

```c
_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_MXQ_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

_task_get_environment()  Gets a pointer to the application-specific environment data for the task.
_task_set_environment()  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

_get_task_exception_handler() Gets a pointer to the task exception handler.
_set_task_exception_handler() Sets the address of the task exception handler.

Prototype

```c
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

\[
\text{source/kernel/task.c}
\]

\[
\_task_id \ _task_get_id(\text{void})
\]

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

    _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

```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 ...

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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td>new_priority is numerically greater than the lowest-allowable priority of an application task. Valid just for _task_set_priority() function.</td>
</tr>
<tr>
<td>MQX_INVALID_TASK_ID</td>
<td>task_id does not represent a currently valid task.</td>
</tr>
</tbody>
</table>

Traits

Might dispatch a task

See Also

_task_get_creator
_task_get_processor
_sem_create
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 \texttt{task\_id}, \texttt{_task\_set\_priority()} raises but not lower the task’s priority.

<table>
<thead>
<tr>
<th>If the task is in this state:</th>
<th>Priority change takes place:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocked</td>
<td>When task is ready</td>
</tr>
<tr>
<td>Ready</td>
<td>Immediately</td>
</tr>
</tbody>
</table>

Example

Raise the priority of the current task.

\texttt{_task\_get\_priority(_task\_get\_id(), &priority);}
\texttt{if (priority > 0) {}
  priority--;
  if (_task\_set\_priority(_task\_get\_id(), priority, &temp) = MQX\_OK)
    ...}

\texttt{}
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

```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

<table>
<thead>
<tr>
<th>Task Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_INVALID_TASK_ID</td>
<td><code>task_id</code> is not valid for this processor.</td>
</tr>
<tr>
<td>MQX_INVALID_TASK_STATE</td>
<td>Task is already in its ready queue.</td>
</tr>
</tbody>
</table>

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.

```c
#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

```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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_INVALID_TASK_ID</td>
<td><code>task_id</code> is invalid.</td>
</tr>
</tbody>
</table>

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(
    _mqx_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 function is called from an ISR, the function sets the interrupt error code.

<table>
<thead>
<tr>
<th>If the current task error code is:</th>
<th>Function changes the task error code:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_OK</td>
<td>To error_code</td>
</tr>
<tr>
<td>Not MQX_OK</td>
<td>To error_code if error_code is MQX_OK</td>
</tr>
</tbody>
</table>

Example

Reset the task error code and check whether it was set.

_mqx_uint error;

error = _task_set_error(MQX_OK);
if (error != MQX_OK) {
    /* Handle the error. */
}
2.1.280  _task_start_preemption,  _task_stop_preemption

_protect_start_preemption()  Enables preemption of the current task.
_protect_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

<table>
<thead>
<tr>
<th>Task error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error from _mem_alloc_system()</td>
<td>MQX RTOS cannot allocate memory for the task queue.</td>
</tr>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td><code>policy</code> is not one of the allowed policies.</td>
</tr>
</tbody>
</table>

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

```c
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

```c
_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td><code>task_queue_ptr</code> is NULL.</td>
</tr>
<tr>
<td>MQX_INVALID_TASK_QUEUE</td>
<td><code>task_queue_ptr</code> does not point to a valid queue.</td>
</tr>
</tbody>
</table>

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**

```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**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQ_INVALID_PARAMETER</td>
<td><code>task_queue_ptr</code> is NULL.</td>
</tr>
<tr>
<td>MQ_INVALID_TASK_QUEUE</td>
<td><code>task_queue_ptr</code> does not point to a valid task queue.</td>
</tr>
</tbody>
</table>

**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

```c
#include <taskq.h>

_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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td>task_queue_ptr is not valid.</td>
</tr>
<tr>
<td>MQX_INVALID_TASK_QUEUE</td>
<td>task_queue_ptr is NULL.</td>
</tr>
<tr>
<td>MQX_TASK_QUEUE_EMPTY</td>
<td>Task queue is empty.</td>
</tr>
</tbody>
</table>

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

```c
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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td><code>task_queue_ptr</code> is NULL.</td>
</tr>
<tr>
<td>MQX_INVALID_TASK_QUEUE</td>
<td><code>task_queue_ptr</code> does not point to a valid task queue.</td>
</tr>
</tbody>
</table>

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

```c
#include <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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td><code>task_queue_ptr</code> is NULL.</td>
</tr>
<tr>
<td>MQX_INVALID_TASK_ID</td>
<td><code>task_id</code> is not a valid task descriptor.</td>
</tr>
<tr>
<td>MQX_INVALID_TASK_QUEUE</td>
<td><code>task_queue_ptr</code> does not point to a valid task queue.</td>
</tr>
<tr>
<td>MQX_INVALID_TASK_STATE</td>
<td>Task is not in the ready state.</td>
</tr>
</tbody>
</table>

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

```c
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

```
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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
<tr>
<td>MQX_CORRUPT_QUEUE</td>
<td>A task on a task queue is not valid.</td>
</tr>
<tr>
<td>MQX_INVALID_TASK_QUEUE</td>
<td>A task queue is not valid.</td>
</tr>
</tbody>
</table>

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

```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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>TICKS</td>
<td>0</td>
<td>$(2^{64}) - 1$</td>
</tr>
<tr>
<td>HW_TICKS</td>
<td>0</td>
<td>$(2^{32}) - 1$</td>
</tr>
</tbody>
</table>
### 2.1.289 _time_add ...

Add time in these units to tick time:

<table>
<thead>
<tr>
<th>Function</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>_time_add_day_to_ticks()</td>
<td>Days</td>
</tr>
<tr>
<td>_time_add_hour_to_ticks()</td>
<td>Hours</td>
</tr>
<tr>
<td>_time_add_min_to_ticks()</td>
<td>Minutes</td>
</tr>
<tr>
<td>_time_add_sec_to_ticks()</td>
<td>Seconds</td>
</tr>
<tr>
<td>_time_add_msec_to_ticks()</td>
<td>Milliseconds</td>
</tr>
<tr>
<td>_time_add_usec_to_ticks()</td>
<td>Microseconds</td>
</tr>
<tr>
<td>_time_add_nsec_to_ticks()</td>
<td>Nanoseconds</td>
</tr>
<tr>
<td>_time_add_psec_to_ticks()</td>
<td>Picoseconds</td>
</tr>
</tbody>
</table>

**Prototype**

```c
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 ...

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_time_delay()</td>
<td>For the number of milliseconds</td>
</tr>
<tr>
<td>_time_delay_for()</td>
<td>For the number of ticks (in tick time)</td>
</tr>
<tr>
<td>_time_delay_ticks()</td>
<td>For the number of ticks</td>
</tr>
<tr>
<td>_time_delay_until()</td>
<td>Until the specified time (in tick time)</td>
</tr>
</tbody>
</table>

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

```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.

```c
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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_time_diff()</td>
<td>Second/millisecond times</td>
</tr>
<tr>
<td>_time_diff_ticks()</td>
<td>Tick times</td>
</tr>
</tbody>
</table>

Prototype

```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.

```c
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_ ...

- **_time_diff_days()**
- **_time_diff_hours()**
- **_time_diff_minutes()**
- **_time_diff_seconds()**
- **_time_diff_milliseconds()**
- **_time_diff_microseconds()**
- **_time_diff_nanoseconds()**
- **_time_diff_ticks()**

<table>
<thead>
<tr>
<th>Function</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>_time_diff_days()</td>
<td>Days</td>
</tr>
<tr>
<td>_time_diff_hours()</td>
<td>Hours</td>
</tr>
<tr>
<td>_time_diff_minutes()</td>
<td>Minutes</td>
</tr>
<tr>
<td>_time_diff_seconds()</td>
<td>Seconds</td>
</tr>
<tr>
<td>_time_diff_milliseconds()</td>
<td>Milliseconds</td>
</tr>
<tr>
<td>_time_diff_microseconds()</td>
<td>Nanoseconds</td>
</tr>
<tr>
<td>_time_diff_nanoseconds()</td>
<td>Picoseconds</td>
</tr>
<tr>
<td>_time_diff_ticks()</td>
<td>See _time_diff(), _time_diff_ticks()</td>
</tr>
</tbody>
</table>

### Prototype

```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)
```

Freescale MQX™ RTOS Reference Manual, Rev. 19
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

```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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
<td>1970</td>
<td>2099</td>
</tr>
<tr>
<td>MONTH</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>DAY</td>
<td>1</td>
<td>31 (depending on the month)</td>
</tr>
<tr>
<td>HOUR</td>
<td>0</td>
<td>23 (since midnight)</td>
</tr>
<tr>
<td>MINUTE</td>
<td>0</td>
<td>59</td>
</tr>
<tr>
<td>SECOND</td>
<td>0</td>
<td>59</td>
</tr>
<tr>
<td>MILISEC</td>
<td>0</td>
<td>999</td>
</tr>
</tbody>
</table>

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.

```c
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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_time_get()</td>
<td>Second/millisecond time</td>
</tr>
<tr>
<td>_time_get_ticks()</td>
<td>Tick time</td>
</tr>
</tbody>
</table>

Prototype

```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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_time_get_elapsed()</td>
<td>Second/millisecond time</td>
</tr>
<tr>
<td>_time_get_elapsed_ticks()</td>
<td>Tick time</td>
</tr>
</tbody>
</table>

Prototype

```c
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 time
- `tick_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

```c
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**

```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**

```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`
MQX RTOS Functions and Macros

2.1.302 _time_get_resolution, _time_set_resolution

_get_resolution() Gets the resolution of the periodic timer interrupt.
_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(
    _mqx_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

```c
source\kernel\time.c
_mqx_uint _time_get_ticks_per_sec(void)

void _time_set_ticks_per_sec(
    _mqx_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

```c
_mqx_uint _time_init_ticks(
    MQX_TICK_STRUCT_PTR tick_time_ptr,
    _mqx_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:

- _time_set()  Second/millisecond time
- _time_set_ticks()  Tick time

Prototype

```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_TIMEPrototype**

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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECONDS</td>
<td>0</td>
<td>MAXIMUM_SECONDS_IN_TIME (4,102,444,800)</td>
</tr>
<tr>
<td>MILLISECONDS</td>
<td>0</td>
<td>999</td>
</tr>
</tbody>
</table>

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

```c
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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECONDS</td>
<td>0</td>
<td>MAXIMUM_SECONDS_IN_TIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(4,102,444,800)</td>
</tr>
<tr>
<td>MILISECONDS</td>
<td>0</td>
<td>999</td>
</tr>
</tbody>
</table>

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**

```c
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

```c
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

```c
#include <time.h>
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

```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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_COMPONENT_DOES_NOT_EXIST</td>
<td>Timer component is not created.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Timer component data is no longer valid.</td>
</tr>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td><code>id</code> is not valid.</td>
</tr>
<tr>
<td>MQX_CANNOT_CALL_FUNCTION_FROM_ISR</td>
<td>Function cannot be called from an ISR.</td>
</tr>
</tbody>
</table>

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

```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.

```c
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("Set 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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>_timer_start_oneshot_after()</td>
<td>Milliseconds</td>
</tr>
<tr>
<td>_timer_start_oneshot_after_ticks()</td>
<td>Ticks (in tick time)</td>
</tr>
</tbody>
</table>

Prototype

```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

<table>
<thead>
<tr>
<th>Task Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Timer component data is no longer valid.</td>
</tr>
<tr>
<td>MQX_INVALID_PARAMETER</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• mode is not one of the allowed modes</td>
</tr>
<tr>
<td></td>
<td>• notification_function is NULL</td>
</tr>
<tr>
<td></td>
<td>• milliseconds is 0</td>
</tr>
<tr>
<td></td>
<td>• tick_time_ptr is NULL</td>
</tr>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>MQX RTOS cannot allocate memory for the timer data.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Time type</th>
</tr>
</thead>
<tbody>
<tr>
<td>_timer_start_oneshot_at()</td>
<td>Second/millisecond time</td>
</tr>
<tr>
<td>_timer_start_oneshot_at_ticks()</td>
<td>Tick time</td>
</tr>
</tbody>
</table>

Prototype

```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)
```

```c
#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

<table>
<thead>
<tr>
<th>Task Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_INVALID_COMPONENT_BASE</td>
<td>Timer component data is no longer valid.</td>
</tr>
</tbody>
</table>
| MQX_INVALID_PARAMETER | One of the following:  
  • mode is not one of the allowed modes  
  • notification_function is NULL  
  • time_ptr is NULL |
| MQX_OUT_OF_MEMORY | MQX RTOS cannot allocate memory for the timer data. |
### 2.1.318 \_timer\_start\_periodic\_at ... 

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_timer_start_periodic_at()</td>
<td>Second/millisecond time</td>
</tr>
<tr>
<td>_timer_start_periodic_at_ticks()</td>
<td>Tick time</td>
</tr>
</tbody>
</table>

#### Prototype

```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)
```

```c
#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

Start a periodic timer at the specified time in: \_timer\_start\_periodic\_at() or \_timer\_start\_periodic\_at\_ticks()
• 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  ...

Prototype

```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)
```

```c
#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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_CORRUPT_QUEUE</td>
<td>Queue of timers is not valid.</td>
</tr>
<tr>
<td>MQX_INVALID_COMPONENT_HANDLE</td>
<td>One of the timer entries in the timer queue is not valid</td>
</tr>
</tbody>
</table>

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

```
#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

```
#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

```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

```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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_usr_lwevent_wait_for()</td>
<td>Wait for the specified lightweight event bits to be set in the lightweight event group: For the number of ticks (in tick time)</td>
</tr>
<tr>
<td>_usr_lwevent_wait_ticks()</td>
<td>For the number of ticks</td>
</tr>
<tr>
<td>_usr_lwevent_wait_until()</td>
<td>Until the specified time (in tick time)</td>
</tr>
</tbody>
</table>

Prototype

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

_mqx_uint _usr_lwevent_wait_ticks(
  LWEVENT_STRUCT_PTR event_ptr,
  _mqx_uint bit_mask,
  bool all,
  _mqx_uint timeout_in_ticks)

_mqx_uint _usr_lwevent_wait_until(
  LWEVENT_STRUCT_PTR event_ptr,
  _mqx_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

```c
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

```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

```c
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

```c
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

```
#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

```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

```c
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

```c
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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_usr_lwsem_wait()</td>
<td>Until it is available</td>
</tr>
<tr>
<td>_usr_lwsem_wait_for()</td>
<td>For the number of ticks (in tick time)</td>
</tr>
<tr>
<td>_usr_lwsem_wait_ticks()</td>
<td>For the number of ticks</td>
</tr>
<tr>
<td>_usr_lwsem_wait_until()</td>
<td>Until the specified time (in tick time)</td>
</tr>
</tbody>
</table>

Prototype

```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

_usrlwsem_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

```c
#include <kernel/task.h>
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

```c
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

```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

```c
#include "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

```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(
    _mqx_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.

### 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

```c
#include <watchdog.h>
_mqx_uint _watchdog_create_component(
  _mqx_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)

<table>
<thead>
<tr>
<th>Errors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQX_OUT_OF_MEMORY</td>
<td>MQX RTOS cannot allocate memory for watchdog component data.</td>
</tr>
<tr>
<td>WATCHDOG_INVALID_ERROR_FUNCTION</td>
<td><code>expiry_function</code> is NULL.</td>
</tr>
<tr>
<td>WATCHDOG_INVALID_INTERRUPT_VECTOR</td>
<td>MQX RTOS cannot install the periodic timer interrupt vector.</td>
</tr>
</tbody>
</table>

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

```c
_mqx_uint result;
extern void task_watchdog_error(TD_STRUCT_PTR td_ptr);
...
```
MQX RTOS Functions and Macros

/* Create watchdog component. */
result = _watchdog_create_component(TIMER_INTERRUPT_VECTOR,
task_watchdog_error);
if (result != MQX_OK) {
/* An error occurred. */
}

Freescale MQX™ RTOS Reference Manual, Rev. 19
Freescale Semiconductor, Inc.

463


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_TICKSTRUCT_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

```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

```c
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

```c
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

<table>
<thead>
<tr>
<th>Data type</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mqx_int</td>
<td>See note 1</td>
<td>See note 1</td>
</tr>
<tr>
<td>_mqx_uint</td>
<td>See note 1</td>
<td>See note 1</td>
</tr>
<tr>
<td>_mqx_max_type</td>
<td></td>
<td>Largest type available (e.g., on a 32-bit processor, _mqx_max_type is defined as uint32_t)</td>
</tr>
<tr>
<td>_mqx_max_type_ptr</td>
<td>See note 3</td>
<td>Pointer to _mqx_max_type</td>
</tr>
<tr>
<td>_mem_size</td>
<td>See note 2</td>
<td>See note 2</td>
</tr>
<tr>
<td>_mem_size_ptr</td>
<td>See note 3</td>
<td>Pointer to _mem_size</td>
</tr>
<tr>
<td>_psp_code_addr</td>
<td>Large enough to hold the address of a code location</td>
<td></td>
</tr>
<tr>
<td>_psp_code_addr_ptr</td>
<td>See note 3</td>
<td>Pointer to _psp_code_addr</td>
</tr>
<tr>
<td>_psp_data_addr</td>
<td>Large enough to hold the address of a data location</td>
<td></td>
</tr>
<tr>
<td>_psp_data_addr_ptr</td>
<td>See note 3</td>
<td>Pointer to _psp_data_addr</td>
</tr>
<tr>
<td>_file_size</td>
<td>uint32_t</td>
<td>Number of bytes in a file</td>
</tr>
<tr>
<td>_file_offset</td>
<td>int32_t</td>
<td>Maximum offset (in bytes) in a file</td>
</tr>
<tr>
<td>ieee_single</td>
<td>32 bits</td>
<td>Single-precision IEEE floating-point number</td>
</tr>
<tr>
<td>ieee_double</td>
<td>32 or 64 bits depending on the compiler</td>
<td>Double-precision IEEE floating-point number</td>
</tr>
</tbody>
</table>

1. _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).
MQX Data Types

2. \texttt{\_mem\_size}: MQX RTOS equates \texttt{\_mem\_size} to the type that can hold the maximum data address for the processor. It is defined in \texttt{psptypes.h} for the PSP.

3. \texttt{\_ptr} are large enough to hold a data address (\texttt{\_mem\_size}).

Table 3-2. MQX RTOS Simple Data Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Defined in</th>
</tr>
</thead>
<tbody>
<tr>
<td>_CODE_PTR_</td>
<td><strong>CODE_PTR</strong></td>
<td>\texttt{psptypes.h} for the PSP</td>
</tr>
<tr>
<td>_lwmem_pool_id</td>
<td>void*</td>
<td>\texttt{lwmem.h}</td>
</tr>
<tr>
<td>_mem_pool_id</td>
<td>void*</td>
<td>\texttt{mqx.h}</td>
</tr>
<tr>
<td>_msg_size</td>
<td>uint16_t</td>
<td>\texttt{message.h}</td>
</tr>
<tr>
<td>_partition_id</td>
<td>void*</td>
<td>\texttt{part.h}</td>
</tr>
<tr>
<td>_pool_id</td>
<td>void*</td>
<td>\texttt{message.h}</td>
</tr>
<tr>
<td>_processor_number</td>
<td>uint16_t</td>
<td>\texttt{mqx.h}</td>
</tr>
<tr>
<td>_queue_id</td>
<td>uint16_t or uint32_t</td>
<td>\texttt{message.h}</td>
</tr>
<tr>
<td>_queue_number</td>
<td>uint16_t or uint32_t</td>
<td>\texttt{message.h}</td>
</tr>
<tr>
<td>_task_id</td>
<td>uint32_t</td>
<td>\texttt{mqx.h}</td>
</tr>
<tr>
<td>_timer_id</td>
<td>_mqx_uint</td>
<td>\texttt{timer.h}</td>
</tr>
</tbody>
</table>

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 \texttt{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

```c
#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;
    int16 WDAY;
    int16 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

<table>
<thead>
<tr>
<th>Field</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td>To</td>
</tr>
<tr>
<td>YEAR</td>
<td>1970 to 2099</td>
</tr>
<tr>
<td>MONTH</td>
<td>1 to 12</td>
</tr>
<tr>
<td>DAY</td>
<td>1 to 28, 29, 30, 31 (depending on the month)</td>
</tr>
<tr>
<td>HOUR</td>
<td>0 to 23</td>
</tr>
<tr>
<td>MINUTE</td>
<td>0 to 59</td>
</tr>
<tr>
<td>SECOND</td>
<td>0 to 59</td>
</tr>
<tr>
<td>MILLISEC</td>
<td>0 to 999</td>
</tr>
<tr>
<td>WDAY</td>
<td>0 to 6</td>
</tr>
<tr>
<td>YDAY</td>
<td>0 to 365</td>
</tr>
</tbody>
</table>
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

```c
#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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_PCB_DEVICE_NAME</td>
<td>String name of the PCB device driver to be opened by the IPC.</td>
</tr>
<tr>
<td>DEVICE_INSTALL</td>
<td>Function to call to install the PCB device (if required)</td>
</tr>
<tr>
<td>DEVICE_INSTALL_PARAMETER</td>
<td>Parameter to pass to the installation function.</td>
</tr>
<tr>
<td>IN_MESSAGES_MAX_SIZE</td>
<td>Maximum size of all messages arriving at the IPC.</td>
</tr>
<tr>
<td>IN_MESSAGES_TO_ALLOCATE</td>
<td>Initial number of input messages to allocate.</td>
</tr>
<tr>
<td>IN_MESSAGES_TO_GROW</td>
<td>Number of input messages to add to the pool when messages are all in use.</td>
</tr>
<tr>
<td>IN_MESSAGES_MAX_ALLOCATE</td>
<td>Maximum number of messages in the input message pool.</td>
</tr>
<tr>
<td>OUT_PCBS_INITIAL</td>
<td>Initial number of PCBs in the output PCB pool.</td>
</tr>
<tr>
<td>OUT_PCBS_TO_GROW</td>
<td>Number of PCBs to add to the output PCB pool when all the PCBs are in use.</td>
</tr>
<tr>
<td>OUT_PCBS_MAX</td>
<td>Maximum number of PCBs in the output PCB pool.</td>
</tr>
</tbody>
</table>
3.2.3   IPC_PROTOCOL_INIT_STRUCT

IPC initialization information.

Prototype

```c
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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPC_PROTOCOL_INIT_INIT</td>
<td>Function that initializes the IPC. The function depends on the IPC.</td>
</tr>
<tr>
<td>IPC_PROTOCOL_INIT_DATA</td>
<td>Pointer to the initialization data that is specific to the IPC protocol.</td>
</tr>
<tr>
<td></td>
<td>The format of the data depends on the IPC.</td>
</tr>
<tr>
<td>IPC_NAME</td>
<td>String name that identifies the IPC.</td>
</tr>
<tr>
<td>IPC_OUT_QUEUE</td>
<td>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.</td>
</tr>
</tbody>
</table>
3.2.4 IPC_ROUTING_STRUCT

Entry in the IPC routing table for interprocessor communication.

Prototype

```c
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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN_PROC_NUMBER MAX_PROC_NUMBER</td>
<td>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.</td>
</tr>
<tr>
<td>QUEUE</td>
<td>Queue number of the IPC output queue.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROUTING_LIST_PTR</td>
<td>Pointer to the IPC routing table.</td>
</tr>
<tr>
<td>PROTOCOL_LIST_PTR</td>
<td>Pointer to the IPC initialization table.</td>
</tr>
</tbody>
</table>
3.2.6  LOG_ENTRY_STRUCT

Header of an entry in a user log.

Prototype

```c
#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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZE</td>
<td>Number of long words in the entry.</td>
</tr>
<tr>
<td>SEQUENCE_NUMBER</td>
<td>Sequence number for the entry.</td>
</tr>
<tr>
<td>SECONDS MILLISECONDS MICROSECONDS</td>
<td>Time at which MQX RTOS wrote the entry.</td>
</tr>
</tbody>
</table>
3.2.7 LWEVENT_STRUCT

Lightweight event group.

Prototype

```c
#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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK</td>
<td>Queue data structures.</td>
</tr>
<tr>
<td>WAITING_TASKS</td>
<td>Queue of tasks waiting for event bits to be set.</td>
</tr>
<tr>
<td>VALID</td>
<td>Validation stamp.</td>
</tr>
<tr>
<td>VALUE</td>
<td>Current bit value of the lightweight event group.</td>
</tr>
<tr>
<td>FLAGS</td>
<td>Flags associated with the lightweight event group; currently only LWEVENT_AUTO_CLEAR.</td>
</tr>
<tr>
<td>AUTO</td>
<td>Mask specifying lightweight event bits that are configured as auto-clear.</td>
</tr>
</tbody>
</table>
3.2.8 LWLOG_ENTRY_STRUCT

Entry in kernel log or a lightweight log.

Prototype

```c
#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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQUENCE_NUMBER</td>
<td>The sequence number for the entry.</td>
</tr>
<tr>
<td>SECONDS MILLISECONDS MICROSECONDS</td>
<td>The time at which the entry was written if MQX RTOS is not configured at compile time to timestamp in ticks.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>The time in tick time at which the entry was written if MQX RTOS is configured at compile time to timestamp in ticks.</td>
</tr>
<tr>
<td>DATA</td>
<td>Data for the entry.</td>
</tr>
<tr>
<td>NEXT_PTR</td>
<td>Pointer to the next lightweight-log entry.</td>
</tr>
</tbody>
</table>
3.2.9  LWSEM_STRUCT

Lightweight semaphore.

Prototype

```c
#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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT</td>
<td>Pointer to the next lightweight semaphore in the list of lightweight semaphores.</td>
</tr>
<tr>
<td>PREV</td>
<td>Pointer to the previous lightweight semaphore in the list of lightweight semaphores.</td>
</tr>
<tr>
<td>TD_QUEUE</td>
<td>Manages the queue of tasks that are waiting for the lightweight semaphore. The NEXT and PREV fields in the task descriptors link the tasks.</td>
</tr>
<tr>
<td>VALID</td>
<td>When MQX RTOS creates the lightweight semaphore, it initializes the field. When MQX RTOS destroys the lightweight semaphore, it clears the field.</td>
</tr>
<tr>
<td>VALUE</td>
<td>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.</td>
</tr>
</tbody>
</table>
3.2.10 LWTIMER_PERIOD_STRUCT

Lightweight timer queue.

Prototype

```c
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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK</td>
<td>Queue of lightweight timers.</td>
</tr>
<tr>
<td>PERIOD</td>
<td>Period (in ticks) of the timer queue; a multiple of BSP_ALARM_RESOLUTION.</td>
</tr>
<tr>
<td>EXPIRY</td>
<td>Number of ticks that have elapsed in the period.</td>
</tr>
<tr>
<td>WAIT</td>
<td>Number of ticks to wait before starting to process the queue.</td>
</tr>
<tr>
<td>TIMERS</td>
<td>Queue of timers to expire at the periodic rate.</td>
</tr>
<tr>
<td>TIMER_PTR</td>
<td>Pointer to the last timer that was processed.</td>
</tr>
<tr>
<td>VALID</td>
<td>When the timer queue is created, MQX RTOS initializes the field. When the queue is cancelled, MQX RTOS clears the field.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK</td>
<td>Queue data structures.</td>
</tr>
<tr>
<td>RELATIVE_TICKS</td>
<td>Relative number of ticks until the timer is to expire.</td>
</tr>
<tr>
<td>VALID</td>
<td>When the timer is added to the timer queue, MQX RTOS initializes the field.</td>
</tr>
<tr>
<td></td>
<td>When the timer or the timer queue that the timer is in is cancelled, MQX</td>
</tr>
<tr>
<td></td>
<td>RTOS clears the field.</td>
</tr>
<tr>
<td>TIMER_FUNCTION</td>
<td>Function that is called when the timer expires.</td>
</tr>
<tr>
<td>PARAMETER</td>
<td>Parameter that is passed to the timer function.</td>
</tr>
<tr>
<td>PERIOD_PTR</td>
<td>Pointer to the lightweight timer queue to which the timer is attached.</td>
</tr>
</tbody>
</table>
3.2.12 MESSAGE_HEADER_STRUCT

Message header.

Prototype

```
#include <message.h>
typedef struct message_header_struct
{
    _msg_size       SIZE;
#if MQX_USE_32BIT_MESSAGE_QIDS
    uint16_t        PAD;
#endif
    _queue_id       TARGET_QID;
    _queue_id       SOURCE_QID;
    unsigned char   CONTROL;
#if MQX_USE_32BIT_MESSAGE_QIDS
    unsigned char   RESERVED[3];
#else
    unsigned char   RESERVED;
#endif
} MESSAGE_HEADER_STRUCT, * MESSAGE_HEADER_STRUCT_PTR;
```

See Also

_msgs_alloc
_msgs_alloc_system
_msgs_free
_msgq_poll
_msgq_receive ...
_msgq_send

Description

All messages must start with a message header.
Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZE</td>
<td>Number of single-addressable units in the message, including the header. The maximum value is MAX_MESSAGE_SIZE. The application sets the field.</td>
</tr>
<tr>
<td>TARGET_QID</td>
<td>Queue ID of the queue to which MQX RTOS is to send the message. The application sets the field.</td>
</tr>
<tr>
<td>SOURCE_QID</td>
<td>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.</td>
</tr>
<tr>
<td>CONTROL</td>
<td>Indicates the following for the message: endian format, priority, urgency</td>
</tr>
<tr>
<td>RESERVED</td>
<td>Not used</td>
</tr>
</tbody>
</table>

Example

See `_msgq_send()`. 
3.2.13 MQX_INITIALIZATION_STRUCT

MQX RTOS initialization structure for each processor.

Prototype

```c
#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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESSOR_NUMBER</td>
<td>Application-unique processor number of the processor. Minimum is 1, maximum is 255.</td>
</tr>
<tr>
<td>START_OF_KERNEL_MEMORY</td>
<td>Lowest address from which MQX RTOS allocates dynamic memory and task stacks.</td>
</tr>
<tr>
<td>END_OF_KERNEL_MEMORY</td>
<td>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.</td>
</tr>
<tr>
<td>INTERRUPT_STACK_SIZE</td>
<td>Maximum number of single-addressable units used by all ISR stacks.</td>
</tr>
<tr>
<td>TASK_TEMPLATE_LIST</td>
<td>Pointer to the task template list for the processor. The default name for the list is <code>MQX_template_list[]</code>.</td>
</tr>
<tr>
<td>MQX_HARDWARE_INTERRUPT_LEVEL_MAX</td>
<td>Hardware priority at which MQX RTOS runs (for processors with multiple interrupt priority levels). All tasks and interrupts run at lower priority.</td>
</tr>
<tr>
<td>MAX_MSGPOOLS</td>
<td>Maximum number of message pools.</td>
</tr>
<tr>
<td>MQX_MSGQS</td>
<td>Maximum number of message queues. Minimum is <code>MSGQ_FIRST_USER_QUEUE</code>, maximum is 255.</td>
</tr>
<tr>
<td>IO_CHANNEL</td>
<td>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.</td>
</tr>
<tr>
<td>IO_OPEN_MODE</td>
<td>Parameter that MQX RTOS passes to the device initialization function when it opens the device.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>Reserved for future enhancements to MQX RTOS; each element of the array must be initialized to 0.</td>
</tr>
</tbody>
</table>

Example

Typical MQX RTOS initialization structure.

```c
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

```c
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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TICKS[:]</td>
<td>Ticks since MQX RTOS started. The field is a minimum of 64 bits; the exact size depends on the PSP.</td>
</tr>
<tr>
<td>HW_TICKS</td>
<td>Hardware ticks (timer counter increments) between ticks. The field increases the accuracy over counting the time simply in ticks.</td>
</tr>
</tbody>
</table>
### 3.2.15 MUTEX_ATTR_STRUCT

Mutex attributes, which are used to initialize a mutex.

#### Prototype

```c
#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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHED_PROTOCOL</td>
<td>Scheduling protocol; one of the following:</td>
</tr>
<tr>
<td></td>
<td>- MUTEX_NO_PRIO_INHERIT</td>
</tr>
<tr>
<td></td>
<td>- MUTEX_PRIO_INHERIT</td>
</tr>
<tr>
<td></td>
<td>- MUTEX_PRIO_PROTECT</td>
</tr>
<tr>
<td></td>
<td>- MUTEX_PRIO_INHERIT</td>
</tr>
<tr>
<td>VALID</td>
<td>When a task calls <code>_mutatr_init()</code>, MQX RTOS sets the field to MUTEX_VALID (defined in <code>mutex.h</code>) 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 TRUE; <code>_mutatr_destroy()</code> sets it to FALSE.</td>
</tr>
<tr>
<td>PRIORITY_CEILING</td>
<td>Priority of the mutex; applicable only if the scheduling protocol is priority protect.</td>
</tr>
<tr>
<td>COUNT</td>
<td>Number of spins to use if the waiting protocol is limited spin.</td>
</tr>
<tr>
<td>WAIT_PROTOCOL</td>
<td>Waiting protocol; one of the following:</td>
</tr>
<tr>
<td></td>
<td>- MUTEX_SPIN_ONLY</td>
</tr>
<tr>
<td></td>
<td>- MUTEX_LIMITED_SPIN</td>
</tr>
<tr>
<td></td>
<td>- MUTEX_QUEUEING</td>
</tr>
<tr>
<td></td>
<td>- MUTEX_PRIORITY_QUEUEING</td>
</tr>
</tbody>
</table>
3.2.16 MUTEX_STRUCT

A mutex.

Prototype

```c
#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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT</td>
<td>Queue of mutexes. MQX RTOS stores the start and end of the queue in MUXEXES of the MUTEX_COMPONENT_STRUCT.</td>
</tr>
<tr>
<td>PREV</td>
<td>Waiting protocol (most significant word) and scheduling protocol (least significant word) for the mutex.</td>
</tr>
<tr>
<td>PROTOCOLS</td>
<td>When a task calls _mutex_init(), MQX RTOS sets the field to MUTEX_VALID (defined in mutex.h) and does not change it. If the field changes, MQX RTOS considers the mutex invalid.</td>
</tr>
<tr>
<td>VALID</td>
<td>Priority of the mutex. If the scheduling protocol is priority protect, MQX RTOS grants the mutex only to tasks with at least this priority.</td>
</tr>
<tr>
<td>PRIORITY_CEILING</td>
<td>Maximum number of spins. The field is used only if the waiting protocol is limited spin.</td>
</tr>
<tr>
<td>COUNT</td>
<td>TRUE if the mutex is being destroyed.</td>
</tr>
<tr>
<td>FILLER</td>
<td>Most significant bit is set when the mutex is locked.</td>
</tr>
<tr>
<td>OWNER_TD</td>
<td>Not used.</td>
</tr>
</tbody>
</table>

Freescale MQX™ RTOS Reference Manual, Rev. 19
<table>
<thead>
<tr>
<th><strong>WAITING_TASKS</strong></th>
<th>Queue of tasks that are waiting to lock the mutex. If PRIORITY_INHERITANCE is set, the queue is in priority order; otherwise, it is in FIFO order.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OWNER_TD</strong></td>
<td>Task descriptor of the task that has locked the mutex.</td>
</tr>
<tr>
<td><strong>BOOSTED</strong></td>
<td>Number of times that MQX RTOS has boosted the priority of the task that has locked the mutex.</td>
</tr>
</tbody>
</table>
3.2.17 QUEUE_ELEMENT_STRUCT

Header for a queue element.

Prototype

```c
#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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT</td>
<td>Pointer to the next element in the queue.</td>
</tr>
<tr>
<td>PREV</td>
<td>Pointer to the previous element in the queue.</td>
</tr>
</tbody>
</table>
3.2.18 QUEUE_STRUCT

Queue of any type of element that has a header of type QUEUE_ELEMENT_STRUCT.

Prototype

```c
#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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT</td>
<td>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.</td>
</tr>
<tr>
<td>PREV</td>
<td>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.</td>
</tr>
<tr>
<td>SIZE</td>
<td>Number of elements in the queue.</td>
</tr>
<tr>
<td>MAX</td>
<td>Maximum number of elements that the queue can hold. If the field is 0, the number is unlimited.</td>
</tr>
</tbody>
</table>
3.2.19 TASK_TEMPLATE_STRUCT

Task template that MQX RTOS uses to create instances of a task.

Prototype

```c
#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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TASK_TEMPLATE_INDEX</td>
<td>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 _task_create() or _task_create_blocked() or _task_create_at() with a template index equal to 0 and a creation parameter set to a pointer to a task template.</td>
</tr>
<tr>
<td>TASK_ADDRESS</td>
<td>Pointer to the root function for the task. When MQX RTOS creates the task, the task begins running at this address.</td>
</tr>
<tr>
<td>TASK_STACKSIZE</td>
<td>Number of single-addressable units of stack space that the task needs.</td>
</tr>
<tr>
<td>TASK_PRIORITY</td>
<td>Software priority of the task. Priorities start at 0, which is the highest priority; 1, 2, 3, and so on, are progressively lower priorities.</td>
</tr>
<tr>
<td>TASK_NAME</td>
<td>Pointer to a name for tasks that MQX RTOS creates from the template.</td>
</tr>
<tr>
<td>TASK_ATTRIBUTES</td>
<td>Attributes of tasks that MQX RTOS creates from the template; any combination of:</td>
</tr>
<tr>
<td>NULL</td>
<td>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.</td>
</tr>
<tr>
<td>MQX_AUTO_START_TASK</td>
<td>When MQX RTOS starts, it creates one instance of the task.</td>
</tr>
<tr>
<td>MQX_DSP_TASK</td>
<td>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.</td>
</tr>
<tr>
<td>MQX_FLOATING_POINT_TASK</td>
<td>MQX RTOS saves floating-point registers as part of the task’s context.</td>
</tr>
<tr>
<td>MQX_TIME_SLICE_TASK</td>
<td>MQX RTOS uses round robin scheduling for the task (the default is FIFO scheduling).</td>
</tr>
<tr>
<td>CREATION_PARAMETER</td>
<td>Passed to tasks that MQX RTOS creates from the template.</td>
</tr>
<tr>
<td>DEFAULT_TIME_SLICE</td>
<td>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.</td>
</tr>
</tbody>
</table>
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},
};
3.2.20  TIME_STRUCT

Time in millisecond format.

Prototype

```c
#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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECONDS</td>
<td>Number of seconds.</td>
</tr>
<tr>
<td>MILLISECONDS</td>
<td>Number of milliseconds.</td>
</tr>
</tbody>
</table>
3.2.21 time_t

Time in second format, representing the number of seconds elapsed since 00:00:00, Jan 1, 1970 UTC.

Prototype

```c
#include <mqx.h>
/*
 * time_t */
*/

#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

```c
#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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tm_sec</td>
<td>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.</td>
</tr>
<tr>
<td>tm_min</td>
<td>The number of minutes after the hour, in the range 0 to 59.</td>
</tr>
<tr>
<td>tm_hour</td>
<td>The number of hours past midnight, in the range 0 to 23.</td>
</tr>
<tr>
<td>tm_mday</td>
<td>The day of the month, in the range 1 to 31.</td>
</tr>
<tr>
<td>tm_mon</td>
<td>The number of months since January, in the range 0 to 11.</td>
</tr>
<tr>
<td>tm_year</td>
<td>The number of years since 1900.</td>
</tr>
<tr>
<td>tm_wday</td>
<td>The number of days since Sunday, in the range 0 to 6.</td>
</tr>
<tr>
<td>tm_yday</td>
<td>The number of days since January 1, in the range 0 to 365.</td>
</tr>
<tr>
<td>tm_isdst</td>
<td>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.</td>
</tr>
</tbody>
</table>