Freescale MQX™ Lite RTOS

Reference Manual

Document Number: MQXLITE_RM
Rev. Version 1.1

Contents

Section number **Title** **Page**

Chapter 1
Before You Begin

1.1 About MQX Lite....11

1.2 About This Book....11

1.3 Function Listing Format....11

1.4 Conventions....13

1.4.1 Notes....13

1.4.2 Cautions....13

Chapter 2
MQX Lite Functions

2.1 Interrupt handling....15

2.1.1 _int_default_isr....15

2.1.2 _int_disable....16

2.1.3 _int_enable....17

2.1.4 _int_get_default_isr....17

2.1.5 _int_get_exception_handler....17

2.1.6 _int_get_isr....18

2.1.7 _int_get_isr_data....19

2.1.8 _int_get_isr_depth....19

2.1.9 _int_get_kernel_isr....20

2.1.10 _int_init....20

2.1.11 _int_install_default_isr....21

2.1.12 _int_install_exception_isr....22

2.1.13 _int_install_isr....22

2.1.14 _int_install_kernel_isr....23

2.1.15 _int_install_unexpected_isr....24

2.1.16 _int_set_exception_handler....25

2.1.17 _int_set_isr_data....25

2.1.18 _int_unexpected_isr....26
2.2 Kernel log....27
2.2.1 _klog_control....27
2.2.2 _klog_create_at....28
2.2.3 _klog_disable_logging_task....29
2.2.4 _klog_display....30
2.2.5 _klog_enable_logging_task....30
2.2.6 _klog_get_interrupt_stack_usage....31
2.2.7 _klog_get_task_stack_usage....32
2.2.8 _klog_log....33
2.2.9 _klog_log_function....33
2.2.10 _klog_show_stack_usage....33
2.3 Lightweight events....34
2.3.1 _lwevent_clear....34
2.3.2 _lwevent_create....35
2.3.3 _lwevent_destroy....36
2.3.4 _lwevent_get_signalled....37
2.3.5 _lwevent_set....37
2.3.6 _lwevent_set_auto_clear....38
2.3.7 _lwevent_test....39
2.3.8 _lwevent_wait_for....40
2.3.9 _lwevent_wait_ticks....41
2.3.10 _lwevent_wait_until....42
2.4 Lightweight logs....43
2.5 Lightweight message queue....43
2.5.1 _lwmsgq_init....44
2.5.2 _lwmsgq_receive....44
2.5.3 _lwmsgq_send....45

Section number	Title	Page

2.6 Lightweight semaphores....46

2.6.1 _lwsem_create....46

2.6.2 _lwsem_destroy....47

2.6.3 _lwsem_poll....48

2.6.4 _lwsem_post....48

2.6.5 _lwsem_test....49

2.6.6 _lwsem_wait....50

2.6.7 _lwsem_wait_for....51

2.6.8 _lwsem_wait_ticks....52

2.6.9 _lwsem_wait_until....53

2.7 Lightweight timers....54

2.7.1 _lwtimer_add_timer_to_queue....54

2.7.2 _lwtimer_cancel_period....55

2.7.3 _lwtimer_cancel_timer....56

2.7.4 _lwtimer_create_periodic_queue....56

2.7.5 _lwtimer_test....57

2.8 Miscellaneous....58

2.8.1 _mqx_exit....58

2.8.2 _mqx_fatal_error....59

2.8.3 _mqx_get_counter....59

2.8.4 _mqx_get_cpu_type....60

2.8.5 _mqx_get_exit_handler....60

2.8.6 _mqx_get_initialization....61

2.8.7 _mqx_get_kernel_data....61

2.8.8 _mqx_get_system_task_id....61

2.8.9 _mqx_idle_task....62

2.8.10 _mqx_set_cpu_type....62

2.8.11 _mqx_set_exit_handler....63

2.8.12 _mqxlite_init....63

2.9 Mutexes....64
2.9.1 _mutatr_destroy....64
2.9.2 _mutatr_get_priority_ceiling....65
2.9.3 _mutatr_get_sched_protocol....65
2.9.4 _mutatr_get_spin_limit....66
2.9.5 _mutatr_get_wait_protocol....67
2.9.6 _mutatr_init....67
2.9.7 _mutatr_set_priority_ceiling....68
2.9.8 _mutatr_set_sched_protocol....69
2.9.9 _mutatr_set_spin_limit....70
2.9.10 _mutatr_set_wait_protocol....70
2.9.11 _mutex_cleanup....71
2.9.12 _mutex_create_component....71
2.9.13 _mutex_destroy....72
2.9.14 _mutex_get_priority_ceiling....73
2.9.15 _mutex_get_wait_count....73
2.9.16 _mutex_init....74
2.9.17 _mutex_lock....75
2.9.18 _mutex_set_priority_ceiling....76
2.9.19 _mutex_test....76
2.9.20 _mutex_try_lock....77
2.9.21 _mutex_unlock....78
2.10 Scheduling....79
2.10.1 _sched_get_max_priority....79
2.10.2 _sched_get_min_priority....80
2.10.3 _sched_yield....80
2.11 Task management....81
2.11.1 _task_abort....81
2.11.2 _task_check_stack....81

2.11.3 _task_create_at....82
2.11.4 _task_destroy....83
2.11.5 _task_get_creator....84
2.11.6 _task_get_environment....84
2.11.7 _task_get_error....85
2.11.8 _task_get_error_ptr....85
2.11.9 _task_get_exception_handler....86
2.11.10 _task_get_exit_handler....86
2.11.11 _task_get_id....87
2.11.12 _task_get_id_from_name....88
2.11.13 _task_get_id_from_td....88
2.11.14 _task_get_index_from_id....89
2.11.15 _task_get_parameter....89
2.11.16 _task_get_parameter_for....90
2.11.17 _task_get_priority....90
2.11.18 _task_get_td....91
2.11.19 _task_get_template_index....91
2.11.20 _task_get_template_ptr....92
2.11.21 _task_ready....92
2.11.22 _task_restart....93
2.11.23 _task_restart_func....94
2.11.24 _task_set_environment....95
2.11.25 _task_set_error....96
2.11.26 _task_set_exception_handler....96
2.11.27 _task_set_exit_handler....97
2.11.28 _task_set_parameter....98
2.11.29 _task_set_parameter_for....99
2.11.30 _task_set_priority....99
2.11.31 _task_start_preemption....100

2.11.32 _task_stop_preemption....101

2.12 Timing....101

2.12.1 _time_delay_for....101

2.12.2 _time_delay_until....102

2.12.3 _time_dequeue....103

2.12.4 _time_dequeue_td....103

2.12.5 _time_diff_ticks....104

2.12.6 _time_diff_ticks_int32....104

2.12.7 _time_get_elapsed_ticks....105

2.12.8 _time_get_elapsed_ticks_fast....106

2.12.9 _time_get_hwticks....106

2.12.10 _time_get_hwticks_per_tick....107

2.12.11 _time_init_ticks....107

2.12.12 _time_notify_kernel....107

2.12.13 _time_set_hwtick_function....108

2.12.14 _time_set_hwticks_per_tick....108

2.12.15 _time_set_timer_vector....109

Chapter 3
MQX Lite Data Types

3.1 LOG_ENTRY_STRUCT....111

3.2 LWEVENT_STRUCT....111

3.3 LWLOG_ENTRY_STRUCT....112

3.4 LWMSGQ_STRUCT....113

3.5 LWSEM_STRUCT....114

3.6 LWTIMER_PERIOD_STRUCT....115

3.7 LWTIMER_STRUCT....116

3.8 MQX_TICK_STRUCT....117

3.9 MQXLITE_INITIALIZATION_STRUCT....118

3.10 MUTEX_ATTR_STRUCT....119

3.11 MUTEX_STRUCT....121

3.12 QUEUE_ELEMENT_STRUCT....122

3.13 QUEUE_STRUCT....122

3.14 TASK_TEMPLATE_STRUCT....123

Chapter 4
Version history

Chapter 1
Before You Begin

1.1 About MQX Lite

MQX Lite is the lightweight version of the MQX™ Real-Time Operating System (RTOS) kernel targeted for resource limited MCUs.

This product is not intended to be used as standalone but it is integrated into the ProcessorExpert (PEx) technology as PEx component to allow PEx users RTOS features.

MQX Lite is not a part of the standard MQX RTOS release. MQX Lite is distributed via Processor Expert Software, i.e. as part of the CodeWarrior tool suite or as an Eclipse-based plug-in feature for installation into an independent Eclipse environment.

1.2 About This Book

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

Use this book in conjunction with other MQX Lite documentation and application notes. The list of these documents will be provided here later.

1.3 Function Listing Format

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

function_name()

A brief description of what function function_name() does.

Prototype

Provides a prototype for the function function_name().

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

Parameters

Function parameters are listed in tables. Example table follows.

Type	Name	Direction	Description
pointer	vector_number	**input**	Parameter that MQX passes to the ISR.

- **Type:** Parameter data type.
- **Name:** Parameter name.
- **Direction:**
  - input - It means the function uses one or more values in the parameter you give it, without storing any changes.
  - output - 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.
  - input, output - 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.
- **Description:** Description for each parameter.

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

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

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 Notes

Notes point out important information.

Note

Non-strict semaphores do not have priority inheritance.

1.4.2 Cautions

Cautions 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

CAUTION

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

Chapter 2
MQX Lite Functions

MQX functions are easy to distinguish and categorize to different components. Each function has a prefix as described in the table below.

Table 2-1. Function overview table

Component	Prefix
Interrupt handling	_int_
Kernel log	_klog_
Lightweight events	_lwevent_
Lightweight message queue	_lwmsgq_
Lightweight semaphores	_lwsem_
Lightweight timers	_lwtimer_
Miscellaneous	_mqx_ _mqxlite_
Mutexes	_mutatr_ _mutex_
Scheduling	_sched_
Task management	_task_
Timing	_time_

2.1 Interrupt handling

2.1.1 _int_default_isr

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

Source :

Prototype : `void _int_default_isr(pointer vector_number);`

Table 2-2. _int_default_isr arguments

Name	Type	Direction	Description
pointer	vector_number	**input**	Parameter that MQX passes to the ISR.

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, in both cases MQX-provided default ISR is installed.

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

MQX changes the state of the active task to UNHANDLED_INT_BLOCKED and blocks it.

CAUTION

Blocks the active task.

2.1.2 _int_disable

This function disables all interrupts for this task.

Source :

Prototype : `void _int_disable(void);`

See also :

- _int_enable

Description :

The function _int_disable disables all hardware interrupts at priorities up to and including the MQX 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 makes ready.

Keep minimum code between calls to _int_disable and its matching _int_enable. If _int_disable or _int_enable are nested, MQX re-enables interrupts only after the number of calls to _int_enable are equal to the number of calls to _int_disable.

2.1.3 _int_enable

This function enables all interrupts for this task.

Source :

Prototype : `void _int_enable(void);`

See also :
- _int_disable

Description :

The function _int_enable resets the processor priority to the hardware priority that corresponds to the active task's software priority. Keep minimum code between calls to _int_disable and its matching _int_enable.

If _int_disable or _int_enable are nested, MQX re-enables interrupts only after the number of calls to _int_enable are equal to the number of calls to _int_disable.

2.1.4 _int_get_default_isr

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

Source :

Prototype : `INT_ISR_FPTR _int_get_default_isr(void);`

Returns :
- Pointer to the default ISR for unhandled interrupts.
- NULL (Failure.)

See also :
- _int_install_default_isr

2.1.5 _int_get_exception_handler

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

Source :

Prototype : `INT_EXCEPTION_FPTR _int_get_exception_handler(_mqx_uint vector);`

Table 2-3. _int_get_exception_handler arguments

Name	Type	Direction	Description
_mqx_uint	vector	**input**	Number of a vector whose exception handler is to be returned.

Returns :

- Pointer to the current exception handler.
- NULL (Failure.)

See also :

- _int_set_exception_handler
- _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.

CAUTION

On failure, calls _task_set_error to set the task error code.

2.1.6 _int_get_isr

Gets the current ISR for the specified vector.

Source :

Prototype : `INT_ISR_FPTR _int_get_isr(_mqx_uint vector);`

Table 2-4. _int_get_isr arguments

Name	Type	Direction	Description
_mqx_uint	vector	**input**	Number of the vector whose ISR is to be returned.

Returns :

- Pointer to the ISR. (Success.)
- NULL (Failure.)

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.

CAUTION

On failure, calls _task_set_error to set the task error code.

2.1.7 _int_get_isr_data

Retrieves a pointer of the interrupt handler data for the specified vector.

Source :

Prototype :`pointer _int_get_isr_data(_mqx_uint vector);`

Table 2-5. _int_get_isr_data arguments

Name	Type	Direction	Description
_mqx_uint	vector	**input**	Number of the vector whose ISR data are to be returned.

Returns :

- Pointer to the ISR data.
- NULL (Failure.)

See also :

- _int_get_isr
- _int_install_isr
- _int_set_isr_data

Description :

ISR data can be installed with _int_set_isr_data.

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

CAUTION

On failure, calls _task_set_error to set the task error code.

2.1.8 _int_get_isr_depth

Gets the depth of nesting of the current interrupt stack.

Source :

Prototype : `_mqx_uint _int_get_isr_depth(void);`

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

2.1.9 _int_get_kernel_isr

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

Source :

Prototype : `INT_KERNEL_ISR_FPTR _int_get_kernel_isr(_mqx_uint);`

Table 2-6. _int_get_kernel_isr arguments

Name	Type	Direction	Description
_mqx_uint	vector	**input**	Vector number whose kernel ISR is requested.

Returns :

- Pointer to the kernel ISR (Success.)
- NULL

See also :

- _int_install_kernel_isr

CAUTION

On failure, calls _task_set_error to set the task error code.

2.1.10 _int_init

This function initializes the kernel interrupt table.

Source :

Prototype : `_mqx_uint _int_init(_mqx_uint first_user_isr_vector_number, _mqx_uint last_user_isr_vector_number);`

Table 2-7. _int_init arguments

Name	Type	Direction	Description
_mqx_uint	first_user_isr_vector_number	**input**	The first (lower) user ISR vector number.
_mqx_uint	last_user_isr_vector_number	**input**	The last user ISR vector number.

Returns :

- MQX_OK (Success.)
- MQX_INVALID_PARAMETER (first_user_isr_vector_number is greater than last_user_isr_vector_number.)
- MQX_OUT_OF_MEMORY (Not enough free memory for the interrupt table.)

2.1.11 _int_install_default_isr

Installs the provided function as the default ISR, called whenever an unhandled interrupt occurs.

Source :

Prototype : `INT_ISR_FPTR _int_install_default_isr(INT_ISR_FPTR default_isr);`

Table 2-8. _int_install_default_isr arguments

Name	Type	Direction	Description
INT_ISR_FPTR	default_isr	**input**	The new default ISR function.

Returns :

- Pointer to the default ISR before the function was called.

See also :

- _int_get_default_isr
- _int_install_isr

Description :

MQX 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.

2.1.12 _int_install_exception_isr

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

Source :

Prototype : `INT_ISR_FPTR _int_install_exception_isr(void);`

Returns :

- Pointer to the default exception handler before the function was called.

See also :

- _int_get_default_isr

Description :

The exception ISR handler performs the following service:

If an unhandled interrupt occurs and

a) A task is running

- If the task has an exception handler, this handler is called

- Otherwise, the task is aborted (_task_abort)

b) An ISR is running

- If the ISR has an exception handler installed, then the exception handler is called. Finally, both exception and ISR interrupt frames are removed.

2.1.13 _int_install_isr

Installs the ISR.

Source :

Prototype : `INT_ISR_FPTR _int_install_isr(_mqx_uint vector, INT_ISR_FPTR isr_ptr, pointer isr_data);`

Table 2-9. _int_install_isr arguments

Name	Type	Direction	Description
_mqx_uint	vector	**input**	Vector number (not the offset) of the interrupt.
INT_ISR_FPTR	isr_ptr	**input**	Pointer to the ISR

Table continues on the next page...

Table 2-9. _int_install_isr arguments (continued)

Name	Type	Direction	Description
pointer	isr_data	**input**	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.
- NULL (Failure.)

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

2.1.14 _int_install_kernel_isr

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

Source :

Prototype : `INT_KERNEL_ISR_FPTR _int_install_kernel_isr(_mqx_uint, INT_KERNEL_ISR_FPTR);`

Table 2-10. _int_install_kernel_isr arguments

Name	Type	Direction	Description
_mqx_uint	vector	**input**	Vector where the ISR is to be installed.
INT_KERNEL_ISR_FPTR	isr_ptr	**input**	Pointer to the ISR to install into the vector table.

Returns :

- Pointer to the previous kernel ISR for the vector (Success.).
- NULL

See also :
- _int_get_kernel_isr

Description :

Some real-time applications need special event handling to occur outside the scope of MQX. The need might arise that the latency in servicing an interrupt be less than the MQX interrupt latency. If this is the case, an application can use _int_install_kernel_isr to bypass MQX 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 functions. However, it can put data in global data, which a task can access.

NOTE

The function is not available for all PSPs.

2.1.15 _int_install_unexpected_isr

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

Source :

Prototype : `INT_ISR_FPTR _int_install_unexpected_isr(void);`

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.

2.1.16 _int_set_exception_handler

Sets the ISR exception handler for the interrupt vector.

Source :

Prototype : `INT_EXCEPTION_FPTR _int_set_exception_handler(_mqx_uint vector, INT_EXCEPTION_FPTR error_handler_address);`

Table 2-11. _int_set_exception_handler arguments

Name	Type	Direction	Description
_mqx_uint	vector	**input**	Interrupt vector that this exception handler is for.
INT_EXCEPTION_FPTR	error_handler_address	**input**	Pointer to the exception handler.

Returns :

- Pointer to the exception handler before the function was called.
- NULL (Failure.)

See also :

- _int_get_exception_handler
- _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 calls the exception handler and terminates the ISR.

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

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

CAUTION

On failure, the exception handler is not installed and _task_set_error is called to set the task error code.

2.1.17 _int_set_isr_data

Sets the address of the interrupt handler data for the specified vector, and returns the old value.

Source :

Prototype :`pointer _int_set_isr_data(_mqx_uint vector, pointer data);`

Table 2-12. _int_set_isr_data arguments

Name	Type	Direction	Description
_mqx_uint	vector	**input**	The interrupt vector that this data is for.
pointer	data	**input**	Data that MQX passes to the ISR as its first parameter.

Returns :

- ISR data before the function was called.
- NULL (Failure.)

See also :

- _int_get_isr
- _int_get_isr_data

CAUTION

On failure, calls _task_set_error to set the task error code.

2.1.18 _int_unexpected_isr

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

Source :

Prototype :`void _int_unexpected_isr(pointer);`

Table 2-13. _int_unexpected_isr arguments

Name	Type	Direction	Description
pointer	parameter	**input**	Parameter passed to the default ISR.

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.

CAUTION

Blocks the active task.

2.2 Kernel log

2.2.1 _klog_control

Controls logging in kernel log.

Source :

Prototype : `void _klog_control(uint_32 bit_mask, boolean set_bits);`

Table 2-14. _klog_control arguments

Name	Type	Direction	Description
uint_32	bit_mask	**input**	Which bits of the kernel log control variable to modify.
boolean	set_bits	**input**	TRUE (Bits set in bit_mask are set in the control variable.), FALSE (Bits set in bit_mask are cleared in the control variable.)

See also :

- _klog_create_at
- _klog_disable_logging_task
- _klog_enable_logging_task

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 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 logs to kernel log only if KLOG_ENABLED is set in bit_mask.

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

If combinations of these bits are set:	Select combinations from:
KLOG_ENABLED (log calls to MQX 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 or** **_klog_enable_logging_task**
KLOG_INTERRUPTS_ENABLED (log interrupts), **KLOG_SYSTEM_CLOCK_INT_ENABLED** (log periodic timer interrupts), **KLOG_CONTEXT_ENABLED** (log context switches)	

2.2.2 _klog_create_at

Creates the kernel log at specified location.

Source :

Prototype : `_mqx_uint _klog_create_at(_mqx_uint max_size, _mqx_uint flags, pointer where);`

Table 2-15. _klog_create_at arguments

Name	Type	Direction	Description
_mqx_uint	max_size	**input**	Maximum size (in mqx_max_types) of the data to be stored.
_mqx_uint	flags	**input**	One of the following: - LOG_OVERWRITE (When the log is full, oldest entries are overwritten.) - 0 (When the log is full, no more entries are written; default.)
pointer	where	**input**	Where in memory is the log to start.

Returns :

- MQX_OK
- LOG_EXISTS (Lightweight log with log number log_number exists.)
- LOG_INVALID (Log_number is out of range.)
- LOG_INVALID_SIZE (Max_size is 0.)
- MQX_INVALID_COMPONENT_BASE (Invalid data for the lightweight log component.)
- MQX_INVALID_POINTER (Pointer "where" is NULL.)

See also :

- _klog_control
- _klog_disable_logging_task
- _klog_enable_logging_task

Description :

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

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

The MQX Embedded PerformanceTool uses kernel log to analyze how the application operates and uses resources.

2.2.3 _klog_disable_logging_task

Disables kernel logging for the task.

Source :

Prototype : `void _klog_disable_logging_task(_task_id tid);`

Table 2-16. _klog_disable_logging_task arguments

Name	Type	Direction	Description
_task_id	tid	**input**	Task ID of the task for which kernel logging is to be disabled.

See also :

- _klog_enable_logging_task
- _klog_control

Description :

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

CAUTION

Disables interrupts.

2.2.4 _klog_display

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

Source :

Prototype :`boolean _klog_display(void);`

Returns :

- TRUE (Entry is found and displayed.)
- FALSE (Entry is not found.)

See also :

- _klog_control
- _klog_create_at

Description :

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

CAUTION

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

2.2.5 _klog_enable_logging_task

Enables kernel logging for the task.

Source :

Prototype :`void _klog_enable_logging_task(_task_id tid);`

Table 2-17. _klog_enable_logging_task arguments

Name	Type	Direction	Description
_task_id	tid	**input**	Task ID of the task for which kernel logging is to be enabled.

See also :

- _klog_disable_logging_task
- _klog_control

Description :

If the application calls _klog_control with KLOG_TASK_QUALIFIED, it must also call _klog_enable_logging_task for each task for which it wants to log information.

Note

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

CAUTION

Enables interrupts.

2.2.6 _klog_get_interrupt_stack_usage

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

Source :

Prototype : `_mqx_uint _klog_get_interrupt_stack_usage(_mem_size_ptr stack_size_ptr, _mem_size_ptr stack_used_ptr);`

Table 2-18. _klog_get_interrupt_stack_usage arguments

Name	Type	Direction	Description
_mem_size_ptr	stack_size_ptr	**output**	Where to write the size (in single-addressable units) of the stack.
_mem_size_ptr	stack_used_ptr	**output**	Where to write the amount (in single-addressable units) of stack used.

Returns :

- MQX_OK
- 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

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 must be configured at compile time with MQX_MONITOR_STACK set to 1. For information on configuring MQX, see MQX User's Guide.

2.2.7 _klog_get_task_stack_usage

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

Source :

Prototype : `_mqx_uint _klog_get_task_stack_usage(_task_id task_id, _mem_size_ptr stack_size_ptr, _mem_size_ptr stack_used_ptr);`

Table 2-19. _klog_get_task_stack_usage arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	Task ID of task to display.
_mem_size_ptr	stack_size_ptr	**output**	Where to write the size (in single-addressable units) of the stack.
_mem_size_ptr	stack_used_ptr	**output**	Where to write the amount (in single-addressable units) of stack used.

Returns :

- MQX_OK
- MQX_INVALID_TASK_ID (Task_id is not valid.)
- MQX_INVALID_CONFIGURATION (Compile-time configuration option MQX_MONITOR_STACK is not set.)

See also :

- _klog_get_interrupt_stack_usage
- _klog_show_stack_usage

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.

Note

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

2.2.8 _klog_log

Logs information into the kernel log.

Source :

Prototype : `void _klog_log(_mqx_uint type, _mqx_max_type p1, _mqx_max_type p2, _mqx_max_type p3, _mqx_max_type p4, _mqx_max_type p5);`

Table 2-20. _klog_log arguments

Name	Type	Direction	Description
_mqx_uint	type	**input**	Defines type of log, see klog.h for possible types.
_mqx_max_type	p1	**input**	Parameter 1.
_mqx_max_type	p2	**input**	Parameter 2.
_mqx_max_type	p3	**input**	Parameter 3.
_mqx_max_type	p4	**input**	Parameter 4.
_mqx_max_type	p5	**input**	Parameter 5.

2.2.9 _klog_log_function

Logs a function address into the kernel log.

Source :

Prototype : `void _klog_log_function(pointer fn);`

Table 2-21. _klog_log_function arguments

Name	Type	Direction	Description
pointer	fn	**input**	Pointer to the function which si to be logged.

See also :

- _klog_log

2.2.10 _klog_show_stack_usage

This function prints out the stack usage for all tasks currently running in the MQX system. It assumes that MQX has been configured with MQX_MONITOR_STACK.

Source :

Prototype : `void _klog_show_stack_usage(void);`

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 must be configured at compile time with MQX_MONITOR_STACK set to 1. For information on configuring MQX, see MQX User's Guide.

CAUTION

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

2.3 Lightweight events

2.3.1 _lwevent_clear

Used by a task to clear the specified event bits in the ligtweight event.

Source :

Prototype : `_mqx_uint _lwevent_clear(LWEVENT_STRUCT_PTR event_ptr, _mqx_uint bit_mask);`

Table 2-22. _lwevent_clear arguments

Name	Type	Direction	Description
LWEVENT_STRUCT_PTR	event_ptr	**input**	Pointer to the event.
_mqx_uint	bit_mask	**input**	Bit mask. Each bit represents an event bit to clear.

Returns :

- MQX_OK
- MQX_LWEVENT_INVALID (Lightweight event is not valid.)

See also :

- _lwevent_create
- _lwevent_destroy
- _lwevent_set
- _lwevent_set_auto_clear
- _lwevent_test
- _lwevent_wait_for
- _lwevent_wait_ticks
- _lwevent_wait_until
- _lwevent_get_signalled
- LWEVENT_STRUCT

2.3.2 _lwevent_create

Used by a task to create an instance of a lightweight event.

Source :

Prototype : `_mqx_uint _lwevent_create(LWEVENT_STRUCT_PTR event_ptr, _mqx_uint flags);`

Table 2-23. _lwevent_create arguments

Name	Type	Direction	Description
LWEVENT_STRUCT_PTR	event_ptr	**input**	Pointer to the lightweight event to initialize.
_mqx_uint	flags	**input**	Creation flag; one of the following: - LWEVENT_AUTO_CLEAR - all bits in the lightweight event 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
- MQX_LWEVENT_INVALID

See also :

- _lwevent_destroy

- _lwevent_set
- _lwevent_set_auto_clear
- _lwevent_clear
- _lwevent_test
- _lwevent_wait_for
- _lwevent_wait_ticks
- _lwevent_wait_until
- _lwevent_get_signalled
- LWEVENT_STRUCT

CAUTION

Disables and enables interrupts.

2.3.3 _lwevent_destroy

Used by a task to destroy an instance of a lightweight event.

Source :

Prototype : `_mqx_uint _lwevent_destroy(LWEVENT_STRUCT_PTR event_ptr);`

Table 2-24. _lwevent_destroy arguments

Name	Type	Direction	Description
LWEVENT_STRUCT_PTR	event_ptr	**input**	Pointer to the event to be deinitialized.

Returns :

- MQX_OK
- MQX_LWEVENT_INVALID (Lightweight event was not valid.)
- MQX_CANNOT_CALL_FUNCTION_FROM_ISR (Function cannot be called from an ISR.)

See also :

- _lwevent_create
- _lwevent_set
- _lwevent_set_auto_clear
- _lwevent_clear
- _lwevent_test
- _lwevent_wait_for
- _lwevent_wait_ticks
- _lwevent_wait_until

- _lwevent_get_signalled
- LWEVENT_STRUCT

Description :

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

CAUTION

Cannot be called from an ISR.

2.3.4 _lwevent_get_signalled

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

Source :

Prototype : `_mqx_uint _lwevent_get_signalled(void);`

Returns :

- bit_mask 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_for
- _lwevent_wait_ticks
- _lwevent_wait_until

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.

2.3.5 _lwevent_set

Used by a task to set the specified event bits in an event.

Source :

Prototype : `_mqx_uint _lwevent_set(LWEVENT_STRUCT_PTR event_ptr, _mqx_uint bit_mask);`

Table 2-25. _lwevent_set arguments

Name	Type	Direction	Description
LWEVENT_STRUCT_PTR	event_ptr	**input**	Pointer to the lightweight event to set bits in.
_mqx_uint	bit_mask	**input**	Bit mask. Each bit represents an event bit to be set.

Returns :

- MQX_OK
- MQX_LWEVENT_INVALID (Lightweight event was invalid.)

See also :

- _lwevent_create
- _lwevent_destroy
- _lwevent_set_auto_clear
- _lwevent_clear
- _lwevent_test
- _lwevent_wait_for
- _lwevent_wait_ticks
- _lwevent_wait_until
- _lwevent_get_signalled
- LWEVENT_STRUCT

CAUTION

Disables and enables interrupts.

2.3.6 _lwevent_set_auto_clear

Sets autoclearing behavior of event bits in the lightweight event.

Source :

Prototype : `_mqx_uint _lwevent_set_auto_clear(LWEVENT_STRUCT_PTR event_ptr, _mqx_uint auto_mask);`

Table 2-26. _lwevent_set_auto_clear arguments

Name	Type	Direction	Description
LWEVENT_STRUCT_PTR	event_ptr	**input**	Pointer to the lightweight event to set bits in.
_mqx_uint	auto_mask	**input**	Mask of events, that is set auto-clear (if corresponding bit of mask is set) or manual-clear (if corresponding bit of mask is clear).

Returns :

- MQX_OK
- MQX_LWEVENT_INVALID (Lightweight event was invalid.)

See also :

- _lwevent_create
- _lwevent_destroy
- _lwevent_set
- _lwevent_clear
- _lwevent_test
- _lwevent_wait_for
- _lwevent_wait_ticks
- _lwevent_wait_until
- _lwevent_get_signalled
- LWEVENT_STRUCT

Description :

Used by a task to set functionality of the specified bits in an event to automatic or manual (1 = automatic, 0 = manual).

CAUTION

Disables and enables interrupts.

2.3.7 _lwevent_test

Tests the event component for validity and consistency.

Source :

Prototype : `_mqx_uint _lwevent_test(pointer _PTR_ event_error_ptr, pointer _PTR_ td_error_ptr);`

Table 2-27. _lwevent_test arguments

Name	Type	Direction	Description
pointer_PTR_	event_error_ptr	**output**	Pointer to the lightweight event that has an error if MQX found an error in the lightweight event component (NULL if no error is found).
pointer_PTR_	td_error_ptr	**output**	TD on the lightweight event in error (NULL if no error is found).

Returns :

- MQX_OK
- MQX_CANNOT_CALL_FUNCTION_FROM_ISR (Function cannot be called from an ISR.)
- MQX_LWEVENT_INVALID (A lightweight event was invalid.)
- code from _queue_test() (Waiting queue for a lightweight event has an error.)

See also :

- _lwevent_create
- _lwevent_destroy
- _lwevent_set
- _lwevent_set_auto_clear
- _lwevent_clear
- _lwevent_wait_for
- _lwevent_wait_ticks
- _lwevent_wait_until
- _lwevent_get_signalled

CAUTION

Cannot be called from an ISR.

2.3.8 _lwevent_wait_for

Used by a task to wait for the number of ticks (in tick time).

Source :

Prototype : `_mqx_uint _lwevent_wait_for(LWEVENT_STRUCT_PTR event_ptr, _mqx_uint bit_mask, boolean all, MQX_TICK_STRUCT_PTR tick_ptr);`

Table 2-28. _lwevent_wait_for arguments

Name	Type	Direction	Description
LWEVENT_STRUCT_PTR	event_ptr	**input**	Pointer to the lightweight event.
_mqx_uint	bit_mask	**input**	Bit mask. Each set bit represents an event bit to wait for.
boolean	all	**input**	TRUE (wait for all bits in bit_mask to be set), FALSE (wait for any bit in bit_mask to be set).
MQX_TICK_STRUCT_PTR	tick_ptr	**input**	Pointer to the maximum number of ticks to wait for the events to be set. If the value is NULL, then the timeout will be infinite.

Returns :

- MQX_OK
- LWEVENT_WAIT_TIMEOUT (The time elapsed before an event signalled.)
- MQX_LWEVENT_INVALID (Lightweight event is no longer valid or was never valid.)
- MQX_CANNOT_CALL_FUNCTION_FROM_ISR (Function cannot be called from an ISR.)

See also :

- _lwevent_create
- _lwevent_destroy
- _lwevent_set
- _lwevent_set_auto_clear
- _lwevent_clear
- _lwevent_wait_ticks
- _lwevent_wait_until
- _lwevent_get_signalled
- LWEVENT_STRUCT
- MQX_TICK_STRUCT

CAUTION

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

2.3.9 _lwevent_wait_ticks

Used by a task to wait for the number of ticks.

Source :

Prototype : `_mqx_uint _lwevent_wait_ticks(LWEVENT_STRUCT_PTR event_ptr, _mqx_uint bit_mask, boolean all, _mqx_uint timeout_in_ticks);`

Table 2-29. _lwevent_wait_ticks arguments

Name	Type	Direction	Description
LWEVENT_STRUCT_PTR	event_ptr	**input**	Pointer to the lightweight event.
_mqx_uint	bit_mask	**input**	Bit mask. Each set bit represents an event bit to wait for.
boolean	all	**input**	TRUE (wait for all bits in bit_mask to be set), FALSE (wait for any bit in bit_mask to be set).
_mqx_uint	timeout_in_ticks	**input**	The maximum number of ticks to wait for the events to be set. If the value is NULL, then the timeout will be infinite.

Returns :

- MQX_OK
- LWEVENT_WAIT_TIMEOUT (The time elapsed before an event signalled.)
- MQX_LWEVENT_INVALID (Lightweight event is no longer valid or was never valid.)
- MQX_CANNOT_CALL_FUNCTION_FROM_ISR (Function cannot be called from an ISR.)

See also :

- _lwevent_create
- _lwevent_destroy
- _lwevent_set
- _lwevent_set_auto_clear
- _lwevent_clear
- _lwevent_wait_for
- _lwevent_wait_until
- _lwevent_get_signalled
- LWEVENT_STRUCT

CAUTION

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

2.3.10 _lwevent_wait_until

Used by a task to wait until the specified time (in tick time).

Source :

Prototype : `_mqx_uint _lwevent_wait_until(LWEVENT_STRUCT_PTR event_ptr, _mqx_uint bit_mask, boolean all, MQX_TICK_STRUCT_PTR tick_ptr);`

Table 2-30. _lwevent_wait_until arguments

Name	Type	Direction	Description
LWEVENT_STRUCT_PTR	event_ptr	**input**	Pointer to the lightweight event.
_mqx_uint	bit_mask	**input**	Bit mask. Each set bit represents an event bit to wait for.
boolean	all	**input**	TRUE (wait for all bits in bit_mask to be set), FALSE (wait for any bit in bit_mask to be set).
MQX_TICK_STRUCT_PTR	tick_ptr	**input**	Pointer to the maximum number of ticks to wait for the events to be set. If the value is NULL, then the timeout will be infinite.

Returns :

- MQX_OK
- LWEVENT_WAIT_TIMEOUT (The time elapsed before an event signalled.)
- MQX_LWEVENT_INVALID (Lightweight event is no longer valid or was never valid.)
- MQX_CANNOT_CALL_FUNCTION_FROM_ISR (Function cannot be called from an ISR.)

See also :

- _lwevent_create
- _lwevent_destroy
- _lwevent_set
- _lwevent_set_auto_clear
- _lwevent_clear
- _lwevent_wait_for
- _lwevent_wait_ticks
- _lwevent_get_signalled
- LWEVENT_STRUCT
- MQX_TICK_STRUCT

CAUTION

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

2.4 Lightweight logs

2.5 Lightweight message queue

2.5.1 _lwmsgq_init

Creates a lightweight message queue.

Source :

Prototype : `_mqx_uint _lwmsgq_init(pointer location, _mqx_uint num_messages, _mqx_uint msg_size);`

Table 2-31. _lwmsgq_init arguments

Name	Type	Direction	Description
pointer	location	**input**	Pointer to memory to create a message queue.
_mqx_uint	num_messages	**input**	Number of messages in the queue.
_mqx_uint	msg_size	**input**	Specifies message size as a multiplier factor of _mqx_max_type items.

Returns :

- MQX_OK
- MQX_EINVAL (The location already points to a valid lightweight message queue.)

See also :

- _lwmsgq_receive
- _lwmsgq_send

Description :

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.

CAUTION

Disables and enables interrupts.

2.5.2 _lwmsgq_receive

Gets a message from a lightweight message queue.

Source :

Prototype : `_mqx_uint _lwmsgq_receive(pointer handle, _mqx_max_type_ptr message, _mqx_uint flags, _mqx_uint ticks, MQX_TICK_STRUCT_PTR tick_ptr);`

Table 2-32. _lwmsgq_receive arguments

Name	Type	Direction	Description
pointer	handle	**input**	Pointer to the message queue created by _lwmsgq_init.
_mqx_max_type_ptr	message	**output**	Received message.
_mqx_uint	flags	**input**	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.
_mqx_uint	ticks	**input**	The maximum number of ticks to wait or NULL (unlimited wait).
MQX_TICK_STRUCT_PTR	tick_ptr	**input**	Pointer to the tick structure to use.

Returns :

- MQX_OK
- LWMSGQ_INVALID (The handle was not valid.)
- LWMSGQ_TIMEOUT (The LWMSGQ_RECEIVE_BLOCK_ON_EMPTY flag was used and no messages were in the message queue.)
- LWMSGQ_EMPTY (No messages were in the message queue before the timeout expired.)

See also :

- _lwmsgq_init
- _lwmsgq_send
- MQX_TICK_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.

CAUTION

Disables and enables interrupts.

2.5.3 _lwmsgq_send

Puts a message on a lightweight message queue.

Source :

Prototype : `_mqx_uint _lwmsgq_send(pointer handle, _mqx_max_type_ptr message, _mqx_uint flags);`

Table 2-33. _lwmsgq_send arguments

Name	Type	Direction	Description
pointer	handle	**input**	Pointer to the message queue created by _lwmsgq_init.
_mqx_max_type_ptr	message	**input**	Pointer to the message to send.
_mqx_uint	flags	**input**	LWMSGQ_SEND_BLOCK_ON_FULL (Block the task if queue is full.) or LWMSGQ_SEND_BLOCK_ON_SEND (Block the task after the message is sent.)

Returns :

- MQX_OK
- LWMSGQ_INVALID (The handle was not valid.)
- LWMSGQ_FULL (The LWMSGQ_SEND_BLOCK_ON_FULL flag was not used and message queue was full.)

See also :

- _lwmsgq_init
- _lwmsgq_receive

Description :

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.

CAUTION

Disables and enables interrupts.

2.6 Lightweight semaphores

2.6.1 _lwsem_create

Creates the lightweight semaphore.

Source :

Prototype : `_mqx_uint _lwsem_create(LWSEM_STRUCT_PTR sem_ptr, _mqx_int initial_number);`

Table 2-34. _lwsem_create arguments

Name	Type	Direction	Description
LWSEM_STRUCT_PTR	sem_ptr	**input**	Pointer to the lightweight semaphore to create.
_mqx_int	initial_number	**input**	Inital number of semaphores available.

Returns :

- MQX_OK
- MQX_EINVAL
- MQX_INVALID_LWSEM

See also :

- _lwsem_destroy
- _lwsem_post
- _lwsem_test
- _lwsem_wait
- _lwsem_wait_for
- _lwsem_wait_ticks
- _lwsem_wait_until
- LWSEM_STRUCT

Description :

Because lightweight semaphores are a core component, an application need not to create the component before it creates lightweight semaphores.

2.6.2 _lwsem_destroy

Destroys the lightweight semaphore.

Source :

Prototype : `_mqx_uint _lwsem_destroy(LWSEM_STRUCT_PTR sem_ptr);`

Table 2-35. _lwsem_destroy arguments

Name	Type	Direction	Description
LWSEM_STRUCT_PTR	sem_ptr	**input**	Pointer to the created lightweight semaphore.

Returns :

- MQX_OK
- MQX_INVALID_LWSEM (Lwsem_ptr does not point to a valid lightweight semaphore.)

See also :

- _lwsem_create
- LWSEM_STRUCT

CAUTION

Puts all waiting tasks in their ready queues. Cannot be called from an ISR.

2.6.3 _lwsem_poll

Poll for the lightweight semaphore.

Source :

Prototype : `boolean _lwsem_poll(LWSEM_STRUCT_PTR sem_ptr);`

Table 2-36. _lwsem_poll arguments

Name	Type	Direction	Description
LWSEM_STRUCT_PTR	sem_ptr	**input**	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
- _lwsem_wait_for
- _lwsem_wait_ticks
- _lwsem_wait_until
- LWSEM_STRUCT

Description :

The function is the nonblocking alternative to the _lwsem_wait family of functions.

2.6.4 _lwsem_post

Posts the lightweight semaphore.

Source :

Prototype : `_mqx_uint _lwsem_post(LWSEM_STRUCT_PTR sem_ptr);`

Table 2-37. _lwsem_post arguments

Name	Type	Direction	Description
LWSEM_STRUCT_PTR	sem_ptr	**input**	Pointer to the created lightweight semaphore.

Returns :

- MQX_OK
- MQX_INVALID_LWSEM (Lwsem_ptr does not point to a valid lightweight semaphore.)

See also :

- _lwsem_create
- _lwsem_wait
- _lwsem_wait_for
- _lwsem_wait_ticks
- _lwsem_wait_until
- LWSEM_STRUCT

Description :

If tasks are waiting for the lightweight semaphore, MQX removes the first one from the queue and puts it in the task's ready queue.

CAUTION

Might put a waiting task in the task's ready queue.

2.6.5 _lwsem_test

Tests the data structures (including queues) of the lightweight semaphores component for consistency and validity.

Source :

Prototype : `_mqx_uint _lwsem_test(pointer _PTR_ lwsem_error_ptr, pointer _PTR_ td_error_ptr);`

Table 2-38. _lwsem_test arguments

Name	Type	Direction	Description
pointer_PTR_	lwsem_error_ptr	**output**	Pointer to the lightweight semaphore in error (NULL if no error is found)
pointer_PTR_	td_error_ptr	**output**	Pointer to the task descriptor of waiting task that has an error (NULL if no error is found).

Returns :

- MQX_OK
- MQX_INVALID_LWSEM (Results of _queue_test().)
- MQX_CANNOT_CALL_FUNCTION_FROM_ISR (Function cannot be called from an ISR.)

See also :

- _lwsem_create
- _lwsem_destroy
- _queue_test

CAUTION

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

2.6.6 _lwsem_wait

Waits (in FIFO order) for the lightweight semaphore until it is available.

Source :

Prototype : `_mqx_uint _lwsem_wait(LWSEM_STRUCT_PTR sem_ptr);`

Table 2-39. _lwsem_wait arguments

Name	Type	Direction	Description
LWSEM_STRUCT_PTR	sem_ptr	**input**	Pointer to the lightweight semaphore.

Returns :

- MQX_OK
- MQX_CANNOT_CALL_FUNCTION_FROM_ISR (Function cannot be called from an ISR.)

- MQX_INVALID_LWSEM (Sem_ptr is for a lightweight semaphore that is not longer valid.)
- MQX_LWSEM_WAIT_TIMEOUT (Timeout expired before the task could get the lightweight semaphore.)

See also :

- _lwsem_create
- _lwsem_post
- LWSEM_STRUCT

Note

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.

CAUTION

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

2.6.7 _lwsem_wait_for

Waits (in FIFO order) for the lightweight semaphore for the number of ticks (in tick time).

Source :

Prototype : `_mqx_uint _lwsem_wait_for(LWSEM_STRUCT_PTR sem_ptr, MQX_TICK_STRUCT_PTR ticks);`

Table 2-40. _lwsem_wait_for arguments

Name	Type	Direction	Description
LWSEM_STRUCT_PTR	sem_ptr	**input**	Pointer to the lightweight semaphore.
MQX_TICK_STRUCT_PTR	ticks	**input**	Pointer to the maximum number of ticks to wait or NULL (unlimited wait).

Returns :

- MQX_OK
- MQX_CANNOT_CALL_FUNCTION_FROM_ISR (Function cannot be called from an ISR.)

- MQX_INVALID_LWSEM (Sem_ptr is for a lightweight semaphore that is not longer valid.)
- MQX_LWSEM_WAIT_TIMEOUT (Timeout expired before the task could get the lightweight semaphore.)

See also :

- _lwsem_create
- _lwsem_post
- LWSEM_STRUCT
- MQX_TICK_STRUCT

Note

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.

CAUTION

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

2.6.8 _lwsem_wait_ticks

Waits (in FIFO order) for the lightweight semaphore for the number of ticks.

Source :

Prototype : `_mqx_uint _lwsem_wait_ticks(LWSEM_STRUCT_PTR sem_ptr, _mqx_uint time_in_ticks);`

Table 2-41. _lwsem_wait_ticks arguments

Name	Type	Direction	Description
LWSEM_STRUCT_PTR	sem_ptr	**input**	Pointer to the lightweight semaphore.
_mqx_uint	time_in_ticks	**input**	Maximum number of ticks to wait or 0 (unlimited wait).

Returns :

- MQX_OK
- MQX_CANNOT_CALL_FUNCTION_FROM_ISR (Function cannot be called from an ISR.)

- MQX_INVALID_LWSEM (Sem_ptr is for a lightweight semaphore that is not longer valid.)
- MQX_LWSEM_WAIT_TIMEOUT (Timeout expired before the task could get the lightweight semaphore.)

See also :

- _lwsem_create
- _lwsem_post
- LWSEM_STRUCT

Note

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.

CAUTION

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

2.6.9 _lwsem_wait_until

Waits (in FIFO order) for the lightweight semaphore until the specified time (in tick time).

Source :

Prototype : `_mqx_uint _lwsem_wait_until(LWSEM_STRUCT_PTR sem_ptr, MQX_TICK_STRUCT_PTR ticks);`

Table 2-42. _lwsem_wait_until arguments

Name	Type	Direction	Description
LWSEM_STRUCT_PTR	sem_ptr	**input**	Pointer to the lightweight semaphore.
MQX_TICK_STRUCT_PTR	ticks	**input**	Pointer to the time (in tick time) until which to wait or NULL (unlimited wait).

Returns :

- MQX_OK
- MQX_CANNOT_CALL_FUNCTION_FROM_ISR (Function cannot be called from an ISR.)

- MQX_INVALID_LWSEM (Sem_ptr is for a lightweight semaphore that is not longer valid.)
- MQX_LWSEM_WAIT_TIMEOUT (Timeout expired before the task could get the lightweight semaphore.)

See also :

- _lwsem_create
- _lwsem_post
- LWSEM_STRUCT
- MQX_TICK_STRUCT

Note

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.

CAUTION

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

2.7 Lightweight timers

2.7.1 _lwtimer_add_timer_to_queue

Adds the lightweight timer to the periodic queue.

Source :

Prototype : `_mqx_uint _lwtimer_add_timer_to_queue(LWTIMER_PERIOD_STRUCT_PTR period_ptr, LWTIMER_STRUCT_PTR timer_ptr, _mqx_uint ticks, LWTIMER_ISR_FPTR func, pointer parameter);`

Table 2-43. _lwtimer_add_timer_to_queue arguments

Name	Type	Direction	Description
LWTIMER_PERIOD_STRUCT_PTR	period_ptr	**input**	Pointer to the periodic queue.
LWTIMER_STRUCT_PTR	timer_ptr	**input**	Pointer to the lightweight timer to add to the queue, must be smaller than queue.
_mqx_uint	ticks	**input**	Tick offset from the timers period to expire at.
LWTIMER_ISR_FPTR	func	**input**	Function to call when the timer expires.
pointer	parameter	**input**	Parameter to pass to the function.

Returns :

- MQX_OK
- MQX_LWTIMER_INVALID (Period_ptr points to an invalid periodic queue.)
- MQX_INVALID_PARAMETER (Ticks is greater than or equal to the periodic queue's period.)

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.

CAUTION

Disables and enables interrupts.

2.7.2 _lwtimer_cancel_period

Cancels all the lightweight timers in the periodic queue.

Source :

Prototype : `_mqx_uint _lwtimer_cancel_period(LWTIMER_PERIOD_STRUCT_PTR period_ptr);`

Table 2-44. _lwtimer_cancel_period arguments

Name	Type	Direction	Description
LWTIMER_PERIOD_STRUCT_PTR	period_ptr	**input**	Pointer to the periodic queue to cancel.

Returns :

- MQX_OK
- MQX_LWTIMER_INVALID (Period_ptr points to an invalid periodic queue.)

See also :

- _lwtimer_add_timer_to_queue
- _lwtimer_cancel_timer

- _lwtimer_create_periodic_queue
- LWTIMER_PERIOD_STRUCT

CAUTION

Disables and enables interrupts.

2.7.3 _lwtimer_cancel_timer

Cancels an outstanding timer request.

Source :

Prototype : `_mqx_uint _lwtimer_cancel_timer(LWTIMER_STRUCT_PTR timer_ptr);`

Table 2-45. _lwtimer_cancel_timer arguments

Name	Type	Direction	Description
LWTIMER_STRUCT_PTR	timer_ptr	**input**	Pointer to the lightweight timer to cancel.

Returns :

- MQX_OK
- MQX_LWTIMER_INVALID (Timer_ptr points to either an invalid timer or to a timer with a periodic queue.)

See also :

- _lwtimer_add_timer_to_queue
- _lwtimer_cancel_period
- _lwtimer_create_periodic_queue
- LWTIMER_STRUCT

CAUTION

Disables and enables interrupts.

2.7.4 _lwtimer_create_periodic_queue

Creates the periodic timer queue.

Source :

Prototype : `_mqx_uint _lwtimer_create_periodic_queue(LWTIMER_PERIOD_STRUCT_PTR period_ptr, _mqx_uint period, _mqx_uint wait_ticks);`

Table 2-46. _lwtimer_create_periodic_queue arguments

Name	Type	Direction	Description
LWTIMER_PERIOD_STRUCT_PTR	period_ptr	**input**	The location of the data structure defining the timing cycle.
_mqx_uint	period	**input**	The cycle length of this timer in ticks.
_mqx_uint	wait_ticks	**input**	The number of ticks to wait before starting this queue.

Returns :

- MQX_OK

See also :

- _lwtimer_add_timer_to_queue
- _lwtimer_cancel_period
- _lwtimer_cancel_timer
- _lwtimer_create_periodic_queue
- LWTIMER_PERIOD_STRUCT

CAUTION

Disables and enables interrupts.

2.7.5 _lwtimer_test

Tests all the periodic queues and their lightweight timers for validity and consistency.

Source :

Prototype : `_mqx_uint _lwtimer_test(pointer _PTR_ period_error_ptr, pointer _PTR_ timer_error_ptr);`

Table 2-47. _lwtimer_test arguments

Name	Type	Direction	Description
pointer_PTR_	period_error_ptr	**output**	Pointer to the first periodic queue that has an error (NULL if no error is found).
pointer_PTR_	timer_error_ptr	**output**	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.)
- MQX_LWTIMER_INVALID (Period_ptr points to an invalid periodic queue.)
- Error from _queue_test() (A periodic queue or its queue was in error.)

See also :

- _lwtimer_add_timer_to_queue
- _lwtimer_cancel_period
- _lwtimer_cancel_timer
- _lwtimer_create_periodic_queue

CAUTION

Disables and enables interrupts.

2.8 Miscellaneous

2.8.1 _mqx_exit

Terminate the MQX application and return to the environment that started the application.

Source :

Prototype : `void _mqx_exit(_mqx_uint error);`

Table 2-48. _mqx_exit arguments

Name	Type	Direction	Description
_mqx_uint	error	**input**	Error code to return to the function that called _mqxlite_init or _mqxlite.

Description :

The function returns back to the environment that called _mqxlite. If the application has installed the MQX exit handler (_mqx_set_exit_handler), _mqx_exit calls the MQX exit handler before it exits. By default, _bsp_exit_handler() is installed as the MQX exit handler in each BSP.

Note

It is important to ensure that the environment (boot call stack) the MQX is returning to is in the consistent state. This is not provided by distributed MQX BSPs, because the boot stack is reused (rewritten) by MQX Kernel data. Set the boot stack outside of Kernel data section to support correct _mqx_exit functionality.

CAUTION

Behavior depends on the BSP.

2.8.2 _mqx_fatal_error

Indicates that an error occurred that is so severe that MQX or the application can no longer function.

Source :

Prototype : `void _mqx_fatal_error(_mqx_uint error);`

Table 2-49. _mqx_fatal_error arguments

Name	Type	Direction	Description
_mqx_uint	error	**input**	Error code.

See also :

- _mqx_exit

Description :

The function logs an error in kernel log (if it has been created and configured to log errors) and calls _mqx_exit.

MQX 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.

CAUTION

Terminates the application by calling _mqx_exit.

2.8.3 _mqx_get_counter

Gets a unique number.

Source :

Prototype : `_mqx_uint _mqx_get_counter(void);`

Returns :

- 16-bit number for 16-bit processors or a 32-bit number for 32-bit processors (unique for the processor and never 0).

Description :

This function increments the counter and then returns value of the counter.

This provides a unique number for whoever requires it.

Note

The unique number will never be 0.

2.8.4 _mqx_get_cpu_type

Gets the CPU type.

Source :

Prototype : `_mqx_uint _mqx_get_cpu_type(void);`

Returns :

- CPU_TYPE field of kernel data.

See also :

- _mqx_set_cpu_type

Description :

CPU types begin with PSP_CPU_TYPE_ and are defined in "source\psp\<cpu_family> \<cpu_family.h>".

2.8.5 _mqx_get_exit_handler

Gets a pointer to the MQX exit handler function called when MQX exits.

Source :

Prototype : `MQX_EXIT_FPTR _mqx_get_exit_handler(void);`

Returns :

- Pointer to the MQX exit handler.

See also :

- _mqx_exit
- _mqx_set_exit_handler

2.8.6 _mqx_get_initialization

Gets a pointer to the MQX initialization structure for this processor.

Source :

Prototype : `MQX_INITIALIZATION_STRUCT_PTR _mqx_get_initialization(void);`

Returns :

- Pointer to the MQX initialization structure in kernel data.

See also :

- _mqxlite_init

2.8.7 _mqx_get_kernel_data

Gets a pointer to kernel data.

Source :

Prototype : `pointer _mqx_get_kernel_data(void);`

Returns :

- Pointer to kernel data.

Description :

The address of kernel data corresponds to START_OF_KERNEL_MEMORY in the MQX initialization structure that the application used to start MQX on the processor.

2.8.8 _mqx_get_system_task_id

Gets the task ID of the System Task.

Source :

Prototype : `_task_id _mqx_get_system_task_id(void);`

Returns :

- TASK_ID Task ID of System Task.

Description :

System resources are owned by System Task.

2.8.9 _mqx_idle_task

This function is the code for the idle task.

Source :

Prototype : `void _mqx_idle_task(uint_32 parameter);`

Table 2-50. _mqx_idle_task arguments

Name	Type	Direction	Description
uint_32	parameter	**input**	Parameter passed to the task when created.

Description :

Idle Task is a MQX task that runs when all application tasks are blocked.

The function implements a simple counter. Size depends on the CPU (64-bit counter for 16-bit CPUs, 128-bit counter for 32-bit CPUs).

Counter can be read from a debugger and idle CPU time can be calculated.

2.8.10 _mqx_set_cpu_type

Sets the CPU type.

Source :

Prototype : `void _mqx_set_cpu_type(_mqx_uint cpu_type);`

Table 2-51. _mqx_set_cpu_type arguments

Name	Type	Direction	Description
_mqx_uint	cpu_type	**input**	The value representing the kernel CPU type.

See also :

- _mqx_get_cpu_type

Description :

The function sets CPU_TYPE in kernel data. The MQX Host Tools family of products uses CPU type. CPU types begin with PSP_CPU_TYPE_ and are defined in source\psp \cpu_family\cpu_family.h.

CAUTION

Does not verify that cpu_type is valid.

2.8.11 _mqx_set_exit_handler

Sets a pointer to the MQX exit handler function called when MQX exits.

Source :

Prototype : `void _mqx_set_exit_handler(MQX_EXIT_FPTR entry);`

Table 2-52. _mqx_set_exit_handler arguments

Name	Type	Direction	Description
MQX_EXIT_FPTR	entry	**input**	Pointer to the exit handler.

See also :

- _mqx_exit
- _mqx_get_exit_handler

2.8.12 _mqxlite_init

Initializes MQXLite on the processor.

Source :

Prototype : `_mqx_uint _mqxlite_init(register MQXLITE_INITIALIZATION_STRUCT_PTR mqx_init);`

Table 2-53. _mqxlite_init arguments

Name	Type	Direction	Description
register MQXLITE_INITIALIZATION_STRUCT_PTR	mqx_init	**input**	Pointer to the MQXLITE initialization structure for the processor.

Returns :

- MQX_OK
- If application calls _mqx_exit(), error code that it passed to _mqx_exit().

See also :

- _mqxlite
- _mqx_exit
- MQXLITE_INITIALIZATION_STRUCT

Description :

The function does the following:

- Initializes kernel data.

- Creates the interrupt stack.

- Creates the ready queues.

- Creates a lightweight semaphore for task creation/destruction.

- Initializes interrupts.

- Initializes system timer.

CAUTION

Must be called exactly once per processor.

2.9 Mutexes

2.9.1 _mutatr_destroy

Deinitializes the mutex attributes structure.

Source :

Prototype : `_mqx_uint _mutatr_destroy(MUTEX_ATTR_STRUCT_PTR attr_ptr);`

Table 2-54. _mutatr_destroy arguments

Name	Type	Direction	Description
MUTEX_ATTR_STRUCT_PTR	attr_ptr	**input**	Pointer to the mutex attributes structure; initialized with _mutatr_init.

Returns :

- MQX_EOK
- MQX_EINVAL (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.

2.9.2 _mutatr_get_priority_ceiling

Gets the priority ceiling from a mutex attributes structure.

Source :

Prototype : `_mqx_uint _mutatr_get_priority_ceiling(MUTEX_ATTR_STRUCT_PTR attr_ptr, _mqx_uint_ptr ceiling_ptr);`

Table 2-55. _mutatr_get_priority_ceiling arguments

Name	Type	Direction	Description
MUTEX_ATTR_STRUCT_PTR	attr_ptr	**input**	Pointer to an initialized mutex attributes structure.
_mqx_uint_ptr	ceiling_ptr	**output**	Pointer to the current priority.

Returns :

- MQX_EOK
- MQX_EINVAL (Attr_ptr is NULL or points to an invalid attributes structure.)

See also :

- _mutatr_set_priority_ceiling
- _mutatr_init
- MUTEX_ATTR_STRUCT

Description :

Priority applies only to mutexes whose scheduling protocol is priority protect.

2.9.3 _mutatr_get_sched_protocol

Gets the scheduling protocol of the mutex attributes structure.

Source :

Prototype : `_mqx_uint _mutatr_get_sched_protocol(MUTEX_ATTR_STRUCT_PTR attr_ptr, _mqx_uint_ptr protocol_ptr);`

Table 2-56. _mutatr_get_sched_protocol arguments

Name	Type	Direction	Description
MUTEX_ATTR_STRUCT_PTR	attr_ptr	**input**	Pointer to an initialized mutex attributes structure.
_mqx_uint_ptr	protocol_ptr	**output**	Pointer to the current scheduling protocol.

Returns :

- MQX_EOK
- MQX_EINVAL (Attr_ptr is NULL or points to an invalid attributes structure.)

See also :

- _mutatr_set_sched_protocol
- _mutatr_init
- _mutatr_get_priority_ceiling
- _mutatr_set_priority_ceiling
- MUTEX_ATTR_STRUCT

2.9.4 _mutatr_get_spin_limit

Gets the spin limit of the mutex attributes structure.

Source :

Prototype : `_mqx_uint _mutatr_get_spin_limit(MUTEX_ATTR_STRUCT_PTR attr_ptr, _mqx_uint_ptr spin_count_ptr);`

Table 2-57. _mutatr_get_spin_limit arguments

Name	Type	Direction	Description
MUTEX_ATTR_STRUCT_PTR	attr_ptr	**input**	Pointer to an initialized mutex attributes structure.
_mqx_uint_ptr	spin_count_ptr	**output**	Pointer to the current spin limit.

Returns :

- MQX_OK
- MQX_EINVAL (Attr_ptr is NULL or points to an invalid attributes structure.)

See also :

- _mutatr_set_spin_limit
- _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.

2.9.5 _mutatr_get_wait_protocol

Gets the waiting policy of the mutex attributes structure.

Source :

Prototype : `_mqx_uint _mutatr_get_wait_protocol(MUTEX_ATTR_STRUCT_PTR attr_ptr, _mqx_uint_ptr waiting_protocol_ptr);`

Table 2-58. _mutatr_get_wait_protocol arguments

Name	Type	Direction	Description
MUTEX_ATTR_STRUCT_PTR	attr_ptr	**input**	Pointer to an initialized mutex attributes structure.
_mqx_uint_ptr	waiting_protocol_ptr	**output**	Pointer to the current waiting protocol.

Returns :

- MQX_EOK
- MQX_EINVAL (Attr_ptr is NULL or points to an invalid attribute structure.)

See also :

- _mutatr_set_wait_protocol
- _mutatr_init
- _mutatr_get_spin_limit
- _mutatr_set_spin_limit
- MUTEX_ATTR_STRUCT

2.9.6 _mutatr_init

Initializes the mutex attributes structure to default values.

Source :

Prototype : `_bmqx_uint _mutatr_init(register MUTEX_ATTR_STRUCT_PTR attr_ptr);`

Table 2-59. _mutatr_init arguments

Name	Type	Direction	Description
register MUTEX_ATTR_STRUCT_PTR	attr_ptr	**input**	Pointer to the mutex attributes structure to initialize.

Returns :

- MQX_EOK
- MQX_EINVAL (Attr_ptr is NULL or points to an invalid attributes structure or attributes structure is already initialized.)

See also :

- _mutex_init
- _mutatr_destroy
- MUTEX_ATTR_STRUCT

Description :

The function initializes the mutex attributes structure to default values and validates the structure. It must be called before a task can modify the values of the mutex attributes structure. The function does not affect any mutexes already initialized with this structure.

Mutex attribute	Field in MUTEX_ATTR_STRUCT	Default value
Scheduling protocol	POLICY	MUTEX_NO_PRIO_INHERIT
Valid	VALID	TRUE
Priority	PRIORITY	0
Spin limit	COUNT	0
Waiting protocol	WAITING POLICY	MUTEX_QUEUEING

2.9.7 _mutatr_set_priority_ceiling

Sets the priority ceiling of a mutex attributes structure.

Source :

Prototype : `_mqx_uint _mutatr_set_priority_ceiling(MUTEX_ATTR_STRUCT_PTR attr_ptr, _mqx_uint ceiling);`

Table 2-60. _mutatr_set_priority_ceiling arguments

Name	Type	Direction	Description
MUTEX_ATTR_STRUCT_PTR	attr_ptr	**input**	Pointer to an initialized mutex attributes structure.
_mqx_uint	ceiling	**input**	New priority ceiling to use.

Returns :

- MQX_EOK
- MQX_EINVAL

See also :

- _mutatr_get_priority_ceiling
- _mutatr_init
- MUTEX_ATTR_STRUCT

Description :

Priority applies only to mutexes whose scheduling protocol is priority protect.

2.9.8 _mutatr_set_sched_protocol

Sets the scheduling protocol of the mutex attributes structure.

Source :

Prototype : `_mqx_uint _mutatr_set_sched_protocol(MUTEX_ATTR_STRUCT_PTR attr_ptr, _mqx_uint protocol);`

Table 2-61. _mutatr_set_sched_protocol arguments

Name	Type	Direction	Description
MUTEX_ATTR_STRUCT_PTR	attr_ptr	**input**	Pointer to an initialized mutex attributes structure.
_mqx_uint	protocol	**input**	New scheduling protocol (see scheduling protocols).

Returns :

- MQX_EOK
- MQX_EINVAL (Attr_ptr is NULL or points to an invalid attributes structure.)

See also :

- _mutatr_get_sched_protocol
- _mutatr_init
- _mutatr_get_priority_ceiling

- _mutatr_set_priority_ceiling
- MUTEX_ATTR_STRUCT

2.9.9 _mutatr_set_spin_limit

Sets the spin limit of the mutex attributes structure.

Source :

Prototype : `_mqx_uint _mutatr_set_spin_limit(MUTEX_ATTR_STRUCT_PTR attr_ptr, _mqx_uint spin_count);`

Table 2-62. _mutatr_set_spin_limit arguments

Name	Type	Direction	Description
MUTEX_ATTR_STRUCT_PTR	attr_ptr	**input**	Pointer to an initialized mutex attributes structure.
_mqx_uint	spin_count	**input**	New spin limit.

Returns :

- MQX_EOK
- MQX_EINVAL (Attr_ptr is NULL or points to an invalid attributes structure.)

See also :

- _mutatr_get_spin_limit
- _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.

2.9.10 _mutatr_set_wait_protocol

Sets the waiting policy of the mutex attributes structure.

Source :

Prototype : `_mqx_uint _mutatr_set_wait_protocol(MUTEX_ATTR_STRUCT_PTR attr_ptr, _mqx_uint waiting_protocol);`

Table 2-63. _mutatr_set_wait_protocol arguments

Name	Type	Direction	Description
MUTEX_ATTR_STRUCT_PTR	attr_ptr	**input**	Pointer to an initialized mutex attributes structure.
_mqx_uint	waiting_protocol	**input**	New waiting protocol (see waiting protocols).

Returns :

- MQX_EOK
- MQX_EINVAL (Attr_ptr is NULL or points to an invalid attribute structure.)

See also :

- _mutatr_get_wait_protocol
- _mutatr_init
- _mutatr_get_spin_limit
- _mutatr_set_spin_limit
- MUTEX_ATTR_STRUCT

CAUTION

Improper use can crash your application.

2.9.11 _mutex_cleanup

Used during task destruction to free up any mutex owned by this task.

Source :

Prototype : `void _mutex_cleanup(TD_STRUCT_PTR td_ptr);`

Table 2-64. _mutex_cleanup arguments

Name	Type	Direction	Description
TD_STRUCT_PTR	td_ptr	**input**	Pointer to the task descriptor of the task to be destroyed.

2.9.12 _mutex_create_component

Installs the mutex component into the kernel.

Source :

Prototype : `_mqx_uint _mutex_create_component(void);`

Returns :

- MQX_OK
- MQX_OUT_OF_MEMORY

See also :

- _mutex_init
- _mutatr_init

Description :

MQX calls the function if the mutex component is not created when a task calls _mutex_init.

2.9.13 _mutex_destroy

Deinitializes the mutex.

Source :

Prototype : `_mqx_uint _mutex_destroy(register MUTEX_STRUCT_PTR mutex_ptr);`

Table 2-65. _mutex_destroy arguments

Name	Type	Direction	Description
register MUTEX_STRUCT_PTR	mutex_ptr	**input**	Pointer to the mutex to be deinitialized.

Returns :

- MQX_EOK
- MQX_COMPONENT_DOES_NOT_EXIST
- MQX_INVALID_COMPONENT_BASE (Mutex component data is not valid.)
- MQX_CANNOT_CALL_FUNCTION_FROM_ISR (From _mutex_lock: function cannot be called from an ISR.)
- MQX_EINVAL (From _mutex_lock: mutex_ptr was destroyed or is NULL.)
- MQX_EDEADLK (From _mutex_lock: task already has the mutex locked.)
- MQX_EBUSY (From _mutex_lock: mutex is already locked.)

See also :

- _mutex_init
- MUTEX_STRUCT

Description :

To reuse the mutex, a task must reinitialize it.

CAUTION

Puts in their ready queues all tasks that are waiting for the mutex; their call to _mutex_lock returns MQX_EINVAL.

2.9.14 _mutex_get_priority_ceiling

Gets the priority of the mutex.

Source :

Prototype : `_mqx_uint _mutex_get_priority_ceiling(MUTEX_STRUCT_PTR mutex_ptr, _mqx_uint_ptr ceiling_ptr);`

Table 2-66. _mutex_get_priority_ceiling arguments

Name	Type	Direction	Description
MUTEX_STRUCT_PTR	mutex_ptr	**input**	Pointer to the mutex.
_mqx_uint_ptr	ceiling_ptr	**output**	Pointer to the previous priority ceiling.

Returns :

- MQX_EOK
- MQX_EINVAL (Mutex_ptr does not point to a valid mutex structure or priority_ptr is NULL)

See also :

- _mutex_set_priority_ceiling
- _mutex_init
- MUTEX_STRUCT

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.

2.9.15 _mutex_get_wait_count

Gets the number of tasks waiting for the specified mutex.

Source :

Prototype : `_mqx_uint _mutex_get_wait_count(register MUTEX_STRUCT_PTR mutex_ptr);`

Table 2-67. _mutex_get_wait_count arguments

Name	Type	Direction	Description
register MUTEX_STRUCT_PTR	mutex_ptr	**input**	Pointer to the mutex.

Returns :

- Number of tasks that are waiting for the mutex.
- MAX_MQX_UINT (Failure.)

See also :

- _mutex_lock
- _task_set_error
- MUTEX_STRUCT

CAUTION

On failure, calls _task_set_error to set the task error code to MQX_EINVAL.

2.9.16 _mutex_init

Initializes the mutex.

Source :

Prototype : `_mqx_uint _mutex_init(register MUTEX_STRUCT_PTR mutex_ptr, register MUTEX_ATTR_STRUCT_PTR attr_ptr);`

Table 2-68. _mutex_init arguments

Name	Type	Direction	Description
register MUTEX_STRUCT_PTR	mutex_ptr	**input**	Pointer to where the mutex is to be initialized.
register MUTEX_ATTR_STRUCT_PTR	attr_ptr	**input**	Pointer to an initialized mutex attributes structure or NULL (use default attributes as defined for _mutatr_init).

Returns :

- MQX_EOK
- MQX_EINVAL (Mutex_ptr is NULL, attr_ptr is not initialized or a value in attr_ptr is not correct.)
- MQX_INVALID_COMPONENT_BASE (Mutex component data is not valid.)
- MQX_OUT_OF_MEMORY

See also :

- _mutex_destroy
- _mutatr_init
- MUTEX_STRUCT
- MUTEX_ATTR_STRUCT

CAUTION

Creates the mutex component if it was not previously created.

2.9.17 _mutex_lock

Locks the mutex.

Source :

Prototype : `_mqx_uint _mutex_lock(register MUTEX_STRUCT_PTR mutex_ptr);`

Table 2-69. _mutex_lock arguments

Name	Type	Direction	Description
register MUTEX_STRUCT_PTR	mutex_ptr	**input**	Pointer to the mutex to be locked.

Returns :

- MQX_EOK
- MQX_CANNOT_CALL_FUNCTION_FROM_ISR (Function cannot be called from an ISR.)
- MQX_EINVAL (Mutex_ptr is NULL or mutex was destroyed.)
- MQX_EDEADLK (Task already has the mutex locked.)
- MQX_EBUSY (Mutex is already locked.)

See also :

- _mutex_init
- _mutex_try_lock
- _mutex_unlock
- _mutatr_init
- _mutatr_get_wait_protocol
- _mutatr_set_wait_protocol
- _mutex_destroy
- MUTEX_STRUCT

Description :

If the mutex is already locked, the task waits according to the waiting protocol of the mutex.

CAUTION

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

2.9.18 _mutex_set_priority_ceiling

Sets the priority of the mutex.

Source :

Prototype : `_mqx_uint _mutex_set_priority_ceiling(MUTEX_STRUCT_PTR mutex_ptr, _mqx_uint ceiling, _mqx_uint_ptr old_ceiling_ptr);`

Table 2-70. _mutex_set_priority_ceiling arguments

Name	Type	Direction	Description
MUTEX_STRUCT_PTR	mutex_ptr	**input**	Pointer to the mutex.
_mqx_uint	ceiling	**input**	New priority ceiling.
_mqx_uint_ptr	old_ceiling_ptr	**output**	Pointer to the previous priority ceiling.

Returns :

- MQX_EOK
- MQX_EINVAL (Mutex_ptr does not point to a valid mutex structure or priority_ptr is NULL)

See also :

- _mutex_get_priority_ceiling
- _mutex_init
- MUTEX_STRUCT

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.

2.9.19 _mutex_test

Tests the mutex component.

Source :

Prototype : `_mqx_uint _mutex_test(pointer _PTR_ mutex_error_ptr);`

Table 2-71. _mutex_test arguments

Name	Type	Direction	Description
pointer_PTR_	mutex_error_ptr	**output**	Pointer to the invalid queue or to the mutex with the error (see return).

Returns :

- MQX_OK No errors were found (mutex_error_ptr = NULL).
- MQX_INVALID_COMPONENT_BASE Mutex component data is not valid (mutex_error_ptr = NULL).
- MQX_EINVAL A mutex is not valid or a mutex queue is not valid (mutex_error_ptr = pointer to the mutex with the error).
- MQX_CORRUPT_QUEUE Queue of mutexes is not valid (mutex_error_ptr = pointer to the invalid queue).

See also :

- _mutex_create_component
- _mutex_init

Description :

The function tests:

- mutex component data

- MQX queue of mutexes

- each mutex

- waiting queue of each mutex

CAUTION

Disables and enables interrupts.

2.9.20 _mutex_try_lock

Tries to lock the mutex.

Source :

Prototype : `_mqx_uint _mutex_try_lock(register MUTEX_STRUCT_PTR mutex_ptr);`

Table 2-72. _mutex_try_lock arguments

Name	Type	Direction	Description
register MUTEX_STRUCT_PTR	mutex_ptr	**input**	Pointer to the mutex.

Returns :

- MQX_EOK
- MQX_EBUSY (Mutex is currently locked.)
- MQX_EDEADLK (Task already has the mutex locked.)
- MQX_EINVAL (Mutex_ptr is NULL or mutex has been destroyed.)

See also :

- _mutex_create_component
- _mutex_init
- _mutex_lock
- _mutex_unlock
- _mutatr_init
- MUTEX_STRUCT

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.

2.9.21 _mutex_unlock

Unlocks the mutex.

Source :

Prototype : `_mqx_uint _mutex_unlock(register MUTEX_STRUCT_PTR mutex_ptr);`

Table 2-73. _mutex_unlock arguments

Name	Type	Direction	Description
register MUTEX_STRUCT_PTR	mutex_ptr	**input**	Pointer to the mutex.

Returns :

- MQX_EOK
- MQX_EINVAL

See also :

- _mutex_create_component
- _mutex_init
- _mutex_lock
- _mutex_try_lock
- _mutatr_init
- MUTEX_STRUCT

Description :

If tasks are waiting for the mutex, MQX removes the first one from the mutex queue and puts the task in the task's ready queue.

CAUTION

Might put a task in the task's ready queue.

2.10 Scheduling

2.10.1 _sched_get_max_priority

Gets the maximum priority that a task can have.

Source :

Prototype : `_mqx_uint _sched_get_max_priority(_mqx_uint);`

Table 2-74. _sched_get_max_priority arguments

Name	Type	Direction	Description
_mqx_uint	policy	**input**	Not used, all task priorities are same for RR or FIFO.

Returns :

- 0 (Always.)

See also :

- _sched_get_min_priority

Description :

This function always returns 0, the highest priority a task may have under MQX.

POSIX compatibility requires the function and the parameter.

2.10.2 _sched_get_min_priority

Gets the minimum priority that an application task can have.

Source :

Prototype : `_mqx_uint _sched_get_min_priority(_mqx_uint);`

Table 2-75. _sched_get_min_priority arguments

Name	Type	Direction	Description
_mqx_uint	policy	**input**	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 starts; it is the priority of the lowest-priority task in the task template list.

2.10.3 _sched_yield

Puts the active task at the end of its ready queue.

Source :

Prototype : `void _sched_yield(void);`

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.

CAUTION

Might dispatch another task.

2.11 Task management

2.11.1 _task_abort

Makes a task run its task exit handler and then destroys itself.

Source :

Prototype : `_mqx_uint _task_abort(_task_id);`

Table 2-76. _task_abort arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	One of the following: - Task ID of the task to be destroyed. - MQX_NULL_TASK_ID (Abort the calling task.)

Returns :

- MQX_OK
- MQX_INVALID_TASK_ID (Task_id does not represent a valid task.)

See also :

- _task_destroy
- _task_get_exit_handler
- _task_set_exit_handler

2.11.2 _task_check_stack

Determines whether the active task's stack is currently overflowed.

Source :

Prototype : `_mqx_uint _task_check_stack(void);`

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.

2.11.3 _task_create_at

Creates the task with the stack location specified.

Source :

Prototype : `_task_id _task_create_at(_processor_number, _mqx_uint, uint_32, pointer, _mem_size);`

Table 2-77. _task_create_at arguments

Name	Type	Direction	Description
_processor_number	processor_number	**input**	One of the following: - Processor number of the processor where the task is to be created. - 0 (Create on the local processor.)
_mqx_uint	template_index	**input**	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.)
uint_32	parameter	**input**	Pointer: - Template_index is not 0 (Pointer to the parameter that MQX passes to the child task.) - Template_index is 0 (Pointer to the task template.)
pointer	stack_ptr	**input**	Pointer to where the stack and TD are to be created.
_mem_size	stack_size	**input**	The size of the stack.

Returns :

- Task ID of the child task (Success.)
- MQX_NULL_TASK_ID (Failure.)

See also :

- _task_abort
- _task_destroy
- _task_get_parameter
- _task_get_parameter_for
- _task_set_parameter
- _task_set_parameter_for

- _task_ready
- _task_set_error

CAUTION

If the child is on another processor, blocks the creator until the child is created. On failure, _task_set_error is called to set the following task error codes:

Task Error Codes

- MQX_INVALID_PROCESSOR_NUMBER (Processor_number is not one of the allowed processor numbers.)

- MQX_NO_TASK_TEMPLATE (Template_index is not in the task template list.)

- MQX_OUT_OF_MEMORY (MQX cannot allocate memory for the task data structures.)

2.11.4 _task_destroy

Destroys the task.

Source :

Prototype : `_mqx_uint _task_destroy(_task_id);`

Table 2-78. _task_destroy arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	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

See also :

- _task_create_at
- _task_get_creator
- _task_get_id
- _task_abort

Description :

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.

CAUTION

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.

2.11.5 _task_get_creator

Gets parent's task ID to the calling task.

Source :

Prototype : `_task_id _task_get_creator(void);`

Returns :

- Task ID of the parent task.

See also :

- _task_get_id

2.11.6 _task_get_environment

Gets a pointer to the application-specific environment data for the task.

Source :

Prototype : `pointer _task_get_environment(_task_id);`

Table 2-79. _task_get_environment arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	Task ID of the task whose environment is to be obtained.

Returns :

- Environment data (Success.)
- NULL (Failure.)

See also :
- _task_set_environment
- _task_get_parameter
- _task_get_parameter_for
- _task_set_parameter
- _task_set_parameter_for
- _task_set_error

CAUTION

On failure, calls _task_set_error to set the task error code to MQX_INVALID_TASK_ID.

2.11.7 _task_get_error

Gets the task error code.

Source :

Prototype : `_mqx_uint _task_get_error(void);`

Returns :
- Task error code for the active task.

See also :
- _task_get_error_ptr
- _task_set_error

2.11.8 _task_get_error_ptr

Gets a pointer to the task error code.

Source :

Prototype : `_mqx_uint_PTR_ _task_get_error_ptr(void);`

Returns :
- Pointer to the task error code.

See also :

- _task_get_error
- _task_set_error

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.

2.11.9 _task_get_exception_handler

Gets a pointer to the task exception handler.

Source :

Prototype : `TASK_EXCEPTION_FPTR _task_get_exception_handler(_task_id);`

Table 2-80. _task_get_exception_handler arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	Task ID of the task whose exception handler is to be obtained.

Returns :

- Pointer to the task exception handler for the task (might be NULL).
- NULL (Task ID is not valid.)

See also :

- _task_set_exception_handler
- _task_get_exit_handler
- _task_set_exit_handler
- _task_set_error

CAUTION

On failure, calls _task_set_error to set the task error code to MQX_INVALID_TASK_ID.

2.11.10 _task_get_exit_handler

Gets a pointer to the task exit handler for the task.

Source :

Prototype : `TASK_EXIT_FPTR _task_get_exit_handler(_task_id);`

Table 2-81. _task_get_exit_handler arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	Task ID of the task whose exit handler is to be obtained.

Returns :

- Pointer to the exit handler (might be NULL).
- NULL (Task_id is not valid.)

See also :

- _task_set_exit_handler
- _mqx_exit
- _task_get_exception_handler
- _task_set_exception_handler
- _task_abort
- _task_set_error

Description :

MQX 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).

CAUTION

On failure, calls _task_set_error to set the task error code to MQX_INVALID_TASK_ID.

2.11.11 _task_get_id

Gets the task ID of the active task.

Source :

Prototype : `_task_id _task_get_id(void);`

Returns :

- Task ID of the active task.

See also :

- _task_get_creator
- _task_get_id_from_name

2.11.12 _task_get_id_from_name

Gets the task ID that is associated with the task name.

Source :

Prototype : `_task_id _task_get_id_from_name(char_ptr);`

Table 2-82. _task_get_id_from_name arguments

Name	Type	Direction	Description
char_ptr	name_ptr	**input**	Pointer to the name to find in the task template list.

Returns :

- Task ID that is associated with the first match of name_ptr.
- MQX_NULL_TASK_ID (Name is not in the task template list.)

See also :

- _task_get_creator
- _task_get_id

Description :

This function uses a task name (from its task template) to find a task id. Only the first task found with the provided name is found.

2.11.13 _task_get_id_from_td

Gets the task ID out of the task descriptor.

Source :

Prototype : `_task_id _task_get_id_from_td(pointer);`

Table 2-83. _task_get_id_from_td arguments

Name	Type	Direction	Description
pointer	td_ptr	**input**	Pointer to the task descriptor.

Returns :

- TASK_ID Task ID
- MQX_NULL_TASK_ID

2.11.14 _task_get_index_from_id

Gets the task template index for the task ID.

Source :

Prototype : `_mqx_uint _task_get_index_from_id(_task_id);`

Table 2-84. _task_get_index_from_id arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	Task ID to look up.

Returns :

- Task template index.
- 0 (Task ID was not found.)

See also :

- _task_get_template_index

2.11.15 _task_get_parameter

Gets the task creation parameter of the active task.

Source :

Prototype : `uint_32 _task_get_parameter(void);`

Returns :

- Creation parameter (might be NULL).

See also :

- _task_get_parameter_for
- _task_set_parameter
- _task_set_parameter_for
- _task_create_at

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.

2.11.16 _task_get_parameter_for

Gets the task creation parameter of the specified task.

Source :

Prototype : `uint_32 _task_get_parameter_for(_task_id);`

Table 2-85. _task_get_parameter_for arguments

Name	Type	Direction	Description
_task_id	tid	**input**	Task ID of the task to get creation parameter from.

Returns :

- Creation parameter (might be NULL).

See also :

- _task_get_parameter
- _task_set_parameter
- _task_set_parameter_for
- _task_create_at

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.

2.11.17 _task_get_priority

Gets the priority of the task.

Source :

Prototype : `_mqx_uint _task_get_priority(_task_id, _mqx_uint_ptr);`

Table 2-86. _task_get_priority arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	One of the following: - Task ID of the task for which to set or get info. - MQX_NULL_TASK_ID (Use the calling task.)
_mqx_uint_ptr	priority_ptr	**output**	Pointer to the priority.

Returns :
- MQX_OK
- MQX_INVALID_TASK_ID (Task_id does not represent a currently valid task.)

See also :
- _task_set_priority
- _task_get_creator
- _mutatr_get_sched_protocol
- _mutatr_set_sched_protocol
- _mutex_lock

CAUTION

Might dispatch a task.

2.11.18 _task_get_td

Gets a pointer to the task descriptor for the task ID.

Source :

Prototype :`pointer _task_get_td(_task_id);`

Table 2-87. _task_get_td arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	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.
- NULL (Task_id is not valid for this processor.)

See also :
- _task_ready

2.11.19 _task_get_template_index

Gets the task template index that is associated with the task name.

Source :

Prototype : `_mqx_uint _task_get_template_index(char_ptr);`

Table 2-88. _task_get_template_index arguments

Name	Type	Direction	Description
char_ptr	name_ptr	**input**	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.
- MQX_NULL_TASK_ID (Name is not in the task template list.)

See also :

- _task_get_id_from_name
- _task_get_index_from_id

2.11.20 _task_get_template_ptr

Gets the pointer to the task template for the task ID.

Source :

Prototype : `TASK_TEMPLATE_STRUCT_PTR _task_get_template_ptr(_task_id);`

Table 2-89. _task_get_template_ptr arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	Task ID for the task for which to get pointer.

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
- TASK_TEMPLATE_STRUCT

2.11.21 _task_ready

Makes the task ready to run by putting it in its ready queue.

Source :

Prototype : `void _task_ready(pointer);`

Table 2-90. _task_ready arguments

Name	Type	Direction	Description
pointer	td	**input**	Pointer to the task descriptor of the task (on this processor) to be made ready.

See also :

- _time_dequeue

Description :

The function is the only way to make ready a task that called _task_block().

CAUTION

If the new ready task has higher priority than the calling task, MQX makes the new ready task active. Might set one of the following task error codes:

- MQX_INVALID_TASK_ID (Task_id is not valid for this processor.)

- MQX_INVALID_TASK_STATE (Task is already in its ready queue.)

2.11.22 _task_restart

Restart the specified task.

Source :

Prototype : `_mqx_uint _task_restart(_task_id, uint_32_ptr, boolean);`

Table 2-91. _task_restart arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	Task ID of the task to restart.
uint_32_ptr	param_ptr	**input**	One of the following: - Pointer to a new task creation parameter. - NULL
boolean	blocked	**input**	Whether the task should be restarted in the blocked state or not.

Returns :

- MQX_OK

- MQX_CANNOT_CALL_FUNCTION_FROM_ISR (Function cannot be called from an ISR.)
- MQX_INVALID_TASK_ID (Task_id is invalid.)
- MQX_OUT_OF_MEMORY (Not enough memory to restart function.)

See also :

- _task_create_at

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 (Reserve stack for Stack Start Structure and call the _task_restart_func.).

CAUTION

Cannot be called from an ISR.

2.11.23 _task_restart_func

Restart the specified task.

Source :

Prototype : `_mqx_uint _task_restart_func(_task_id task_id, uint_32_ptr param_ptr, boolean blocked);`

Table 2-92. _task_restart_func arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	Task ID of the task to restart.
uint_32_ptr	param_ptr	**input**	One of the following: - Pointer to a new task creation parameter. - NULL (victim's creation parameter used).
boolean	blocked	**input**	Whether the task should be restarted in the blocked state or not.

Returns :

- MQX_OK
- MQX_CANNOT_CALL_FUNCTION_FROM_ISR (Function cannot be called from an ISR.)
- MQX_INVALID_TASK_ID (Task_id is invalid.)
- MQX_OUT_OF_MEMORY (Not enough memory to rebuild stack.)

See also :

- _task_create_at

Description :

Restart the task specified by the given task_id (the victim).

All of the victim's resources are released, specifically all queues are closed and all memory is freed.

Component cleanup functions are called to free any component resources owned by this task.

CAUTION

Cannot be called from an ISR.

2.11.24 _task_set_environment

Sets the address of the application-specific environment data for the task.

Source :

Prototype :`pointer _task_set_environment(_task_id, pointer);`

Table 2-93. _task_set_environment arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	Task ID of the task whose environment is to be set.
pointer	environment_ptr	**input**	Pointer to the environment data.

Returns :

- Previous environment data (Success.)
- NULL (Failure.)

See also :

- _task_get_environment
- _task_get_parameter
- _task_get_parameter_for
- _task_set_parameter
- _task_set_parameter_for
- _task_set_error

CAUTION

On failure, calls _task_set_error to set the task error code to MQX_INVALID_TASK_ID.

2.11.25 _task_set_error

Sets the task error code.

Source :

Prototype : `_mqx_uint _task_set_error(_mqx_uint);`

Table 2-94. _task_set_error arguments

Name	Type	Direction	Description
_mqx_uint	new_error_code	**input**	New task error code.

Returns :

- Previous task error code.

See also :

- _task_check_stack
- _task_get_error
- _task_get_error_ptr

Description :

MQX uses the function to indicate an error. MQX never sets the task error code to MQX_OK; that is, MQX 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 detected since the last time the application reset the task error code.

If the current task error code is:	Function changes the task error code:
MQX_OK	To new_error_code.
Not MQX_OK	To new_error_code if new_error_code is MQX_OK.

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

2.11.26 _task_set_exception_handler

Sets the address of the task exception handler.

Source :

Prototype : `TASK_EXCEPTION_FPTR _task_set_exception_handler(_task_id, TASK_EXCEPTION_FPTR);`

Table 2-95. _task_set_exception_handler arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	Task ID of the task whose exception handler is to be set.
TASK_EXCEPTION_FPTR	handler_address	**input**	Pointer to the task exception handler.

Returns :

- Pointer to the previous task exception handler (might be NULL).
- NULL (Task ID is not valid.)

See also :

- _task_get_exception_handler
- _task_get_exit_handler
- _task_set_exit_handler
- _task_set_error

CAUTION

On failure, calls _task_set_error to set the task error code to MQX_INVALID_TASK_ID.

2.11.27 _task_set_exit_handler

Sets the address of the task exit handler for the task.

Source :

Prototype : `TASK_EXIT_FPTR _task_set_exit_handler(_task_id, TASK_EXIT_FPTR);`

Table 2-96. _task_set_exit_handler arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	Task ID of the task whose exit handler is to be set.
TASK_EXIT_FPTR	exit_handler_address	**input**	Pointer to the exit handler for the task.

Returns :

- Pointer to the previous exit handler (might be NULL).
- NULL (Task_id is not valid.)

See also :

- _task_get_exit_handler
- _mqx_exit
- _task_get_exception_handler
- _task_set_exception_handler
- _task_abort
- _task_set_error

Description :

MQX 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).

CAUTION

On failure, calls _task_set_error to set the task error code to MQX_INVALID_TASK_ID.

2.11.28 _task_set_parameter

Sets the task creation parameter of the active task.

Source :

Prototype : `uint_32 _task_set_parameter(uint_32);`

Table 2-97. _task_set_parameter arguments

Name	Type	Direction	Description
uint_32	new_value	**input**	Value to set the task parameter to.

Returns :

- Previous creation parameter (might be NULL).

See also :

- _task_get_parameter
- _task_get_parameter_for
- _task_set_parameter_for
- _task_create_at

2.11.29 _task_set_parameter_for

Sets the task creation parameter of the specified task.

Source :

Prototype : `uint_32 _task_set_parameter_for(uint_32, _task_id);`

Table 2-98. _task_set_parameter_for arguments

Name	Type	Direction	Description
uint_32	new_value	**input**	Value to set the task parameter to.
_task_id	tid	**input**	Task ID of the task to set.

Returns :

- Previous creation parameter (might be NULL).

See also :

- _task_get_parameter
- _task_get_parameter_for
- _task_set_parameter
- _task_create_at

2.11.30 _task_set_priority

Sets the priority of the task.

Source :

Prototype : `_mqx_uint _task_set_priority(_task_id, _mqx_uint, _mqx_uint_ptr);`

Table 2-99. _task_set_priority arguments

Name	Type	Direction	Description
_task_id	task_id	**input**	One of the following: - Task ID of the task for which to set or get info. - MQX_NULL_TASK_ID (Use the calling task.)
_mqx_uint	new_priority	**input**	New task priority.
_mqx_uint_ptr	priority_ptr	**output**	Pointer to the previous task priority.

Returns :

- MQX_OK

- MQX_INVALID_TASK_ID (Task_id does not represent a currently valid task.)
- MQX_INVALID_PARAMETER (New_priority is numerically greater than the lowest-allowable priority of an application task.)

See also :

- _task_get_priority
- _task_get_creator
- _mutatr_get_sched_protocol
- _mutatr_set_sched_protocol
- _mutex_lock

Description :

MQX might boost the priority of a task that waits for a semaphore or locks a mutex. If MQX has boosted the priority of the task that is specified by task_id, _task_set_priority will raise but not lower the task's priority.

If the task is in the blocked state, priority change takes place when task is ready.

When the task is in the ready state, priority change takes place immediately.

CAUTION

Might dispatch a task.

2.11.31 _task_start_preemption

Enables preemption of the current task.

Source :

Prototype : `void _task_start_preemption(void);`

See also :

- _task_stop_preemption
- _task_ready

Description :

The _task_start_preemption function enables preemption of the active task after _task_stop_preemption was called.

CAUTION

Changes the preemption ability of tasks. Interrupts are still handled.

2.11.32 _task_stop_preemption

Disables preemption of the current task.

Source :

Prototype : `void _task_stop_preemption(void);`

See also :

- _task_start_preemption
- _task_ready

Description :

The _task_stop_preemption function disables preemption of the active task unless the task blocks explicitly (_task_block()) or calls _task_start_preemption.

CAUTION

Changes the preemption ability of tasks. Interrupts are still handled.

2.12 Timing

2.12.1 _time_delay_for

Suspends the active task for the number of ticks (in tick time).

Source :

Prototype : `void _time_delay_for(register MQX_TICK_STRUCT_PTR ticks);`

Table 2-100. _time_delay_for arguments

Name	Type	Direction	Description
register MQX_TICK_STRUCT_PTR	ticks	**input**	Pointer to the minimum number of ticks to suspend the task.

See also :

- _time_delay_until

- _time_dequeue
- MQX_TICK_STRUCT

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.

CAUTION

Blocks the calling task.

2.12.2 _time_delay_until

Suspends the active task until the specified time (in tick time).

Source :

Prototype : `void _time_delay_until(register MQX_TICK_STRUCT_PTR ticks);`

Table 2-101. _time_delay_until arguments

Name	Type	Direction	Description
register MQX_TICK_STRUCT_PTR	ticks	**input**	Pointer to the time (in tick time) until which to suspend the task.

See also :

- _time_delay_for
- _time_dequeue
- MQX_TICK_STRUCT

Description :

The functions put the active task in the timeout queue until the specified tick count is reached.

Before the time expires, any task can remove the task from the timeout queue by calling _time_dequeue.

CAUTION

Blocks the calling task.

2.12.3 _time_dequeue

Removes the task (specified by task ID) from the timeout queue.

Source :

Prototype : `void _time_dequeue(_task_id tid);`

Table 2-102. _time_dequeue arguments

Name	Type	Direction	Description
_task_id	tid	**input**	Task ID of the task to be removed from the timeout queue.

See also :

- _task_ready
- _time_delay_for
- _time_delay_until
- _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.

CAUTION

Removes the task from the timeout queue, but does not put it in the task's ready queue.

2.12.4 _time_dequeue_td

Removes the task (specified by task descriptor) from the timeout queue.

Source :

Prototype : `void _time_dequeue_td(pointer td);`

Table 2-103. _time_dequeue_td arguments

Name	Type	Direction	Description
pointer	td	**input**	Pointer to the task descriptor of the task to be removed from the timeout queue.

See also :

- _task_ready
- _time_delay_for
- _time_delay_until
- _time_dequeue

CAUTION

Removes the task from the timeout queue; does not put it in the task's ready queue.

2.12.5 _time_diff_ticks

Get the difference between two tick times.

Source :

Prototype : `_mqx_uint _time_diff_ticks(MQX_TICK_STRUCT_PTR, MQX_TICK_STRUCT_PTR, MQX_TICK_STRUCT_PTR);`

Table 2-104. _time_diff_ticks arguments

Name	Type	Direction	Description
MQX_TICK_STRUCT_PTR	end_tick_ptr	**input**	Pointer to the normalized end time, which must be greater than the start time.
MQX_TICK_STRUCT_PTR	start_tick_ptr	**input**	Pointer to the normalized start time in tick time.
MQX_TICK_STRUCT_PTR	diff_tick_ptr	**output**	Pointer to the time difference (the time is normalized).

Returns :

- MQX_OK
- MQX_INVALID_PARAMETER (One or more pointers are NULL.)

See also :

- MQX_TICK_STRUCT

2.12.6 _time_diff_ticks_int32

Get the difference between two tick times and clamps result into int_32 interval.

Source :

Prototype : `int_32 _time_diff_ticks_int32(MQX_TICK_STRUCT_PTR, MQX_TICK_STRUCT_PTR, boolean _PTR_);`

Table 2-105. _time_diff_ticks_int32 arguments

Name	Type	Direction	Description
MQX_TICK_STRUCT_PTR	end_tick_ptr	**input**	Pointer to the normalized end time (in ticks), which must be greater than the start time.
MQX_TICK_STRUCT_PTR	start_tick_ptr	**input**	Pointer to the normalized start time (in ticks).
boolean	overflow_ptr	**output**	Set to TRUE if overflow occurs.

Returns :

- Difference between the times as int_32 (<-(MAX_INT_32 + 1), MAX_INT_32>).

See also :

- _time_diff_ticks
- MQX_TICK_STRUCT

2.12.7 _time_get_elapsed_ticks

Get the time elapsed since MQX started in tick time.

Source :

Prototype : `void _time_get_elapsed_ticks(MQX_TICK_STRUCT_PTR);`

Table 2-106. _time_get_elapsed_ticks arguments

Name	Type	Direction	Description
MQX_TICK_STRUCT_PTR	tick_ptr	**input, output**	Where to store the elapsed tick time.

See also :

- _time_get_elapsed_ticks_fast
- MQX_TICK_STRUCT

Description :

The function always returns elapsed time; it is not affected by _time_set() or _time_set_ticks().

2.12.8 _time_get_elapsed_ticks_fast

Get the time elapsed since MQX started in tick time.

Source :

Prototype : `void _time_get_elapsed_ticks_fast(MQX_TICK_STRUCT_PTR);`

Table 2-107. _time_get_elapsed_ticks_fast arguments

Name	Type	Direction	Description
MQX_TICK_STRUCT_PTR	tick_ptr	**input, output**	Where to store the elapsed tick time.

See also :

- _time_get_elapsed_ticks
- MQX_TICK_STRUCT

Description :

The function always returns elapsed time; it is not affected by _time_set() or _time_set_ticks().

The only difference between _time_get_elapsed_ticks_fast and _time_get_elapsed_ticks is that this one is supposed to be called from code with interrupts DISABLED. Do not use this function with interrupts ENABLED.

2.12.9 _time_get_hwticks

Gets the number of hardware ticks since the last tick.

Source :

Prototype : `uint_32 _time_get_hwticks(void);`

Returns :

- Number of hardware ticks since the last tick.

See also :

- _time_get_hwticks_per_tick
- _time_set_hwticks_per_tick

2.12.10 _time_get_hwticks_per_tick

Gets the number of hardware ticks per tick.

Source :

Prototype : `uint_32 _time_get_hwticks_per_tick(void);`

Returns :

- Number of hardware ticks per tick.

See also :

- _time_set_hwticks_per_tick
- _time_get_hwticks

2.12.11 _time_init_ticks

Initializes a tick time structure with the number of ticks.

Source :

Prototype : `_mqx_uint _time_init_ticks(MQX_TICK_STRUCT_PTR, _mqx_uint);`

Table 2-108. _time_init_ticks arguments

Name	Type	Direction	Description
MQX_TICK_STRUCT_PTR	tick_ptr	**input, output**	Pointer to the tick time structure to initialize.
_mqx_uint	ticks	**input**	Number of ticks with which to initialize the structure.

Returns :

- MQX_OK
- MQX_INVALID_PARAMETER (Tick_ptr is NULL.)

See also :

- MQX_TICK_STRUCT

2.12.12 _time_notify_kernel

The BSP periodic timer ISR calls the function when a periodic timer interrupt occurs.

Source :

Prototype : `void _time_notify_kernel(void);`

See also :

- _time_get_elapsed_ticks

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 components that use time will not operate.

CAUTION

See description.

2.12.13 _time_set_hwtick_function

Set the fields in kernel data to get the hardware ticks.

Source :

Prototype : `void _time_set_hwtick_function(MQX_GET_HWTICKS_FPTR, pointer);`

Table 2-109. _time_set_hwtick_function arguments

Name	Type	Direction	Description
MQX_GET_HWTICKS_FPTR	hwtick_function_ptr	**input**	Pointer to the function that returns hw tick, to be executed by the kernel.
pointer	parameter	**input**	Parameter of the function that returns hw tick.

See also :

- _time_set_hwticks_per_tick
- _time_get_hwticks

2.12.14 _time_set_hwticks_per_tick

Sets the number of hardware ticks per tick.

Source :

Prototype : `void _time_set_hwticks_per_tick(uint_32);`

Table 2-110. _time_set_hwticks_per_tick arguments

Name	Type	Direction	Description
uint_32	new_val	**input**	New number of hardware ticks per tick.

See also :

- _time_get_hwticks_per_tick
- _time_get_hwticks

2.12.15 _time_set_timer_vector

Sets the periodic timer interrupt vector number that MQX uses.

Source :

Prototype : `void _time_set_timer_vector(_mqx_uint);`

Table 2-111. _time_set_timer_vector arguments

Name	Type	Direction	Description
_mqx_uint	vector	**input**	Periodic timer interrupt vector to use.

Description :

The BSP should call the function during initialization.

Chapter 3
MQX Lite Data Types

3.1 LOG_ENTRY_STRUCT

Header of an entry in a user log.

Description :

The length of the entry depends on the SIZE field.

Source :

Declaration :

```
typedef struct
{
  _mqx_uint SIZE,
  _mqx_uint SEQUENCE_NUMBER,
  uint_32   SECONDS,
  uint_16   MILLISECONDS,
  uint_16   MICROSECONDS
} LOG_ENTRY_STRUCT;
```

Table 3-1. Structure LOG_ENTRY_STRUCT member description

Member	Description
SIZE	The size of this entry in _mqx_uints.
SEQUENCE_NUMBER	The sequence number for the entry.
SECONDS	The time (in seconds) at which MQX wrote the entry.
MILLISECONDS	The time (in milliseconds) at which MQX wrote the entry.
MICROSECONDS	The time (in microseconds) at which MQX wrote the entry.

3.2 LWEVENT_STRUCT

This structure defines a lightweight event.

Description :

Tasks can wait on and set event bits.

Source :

Declaration :

```
typedef struct
{
  QUEUE_ELEMENT_STRUCT LINK,
  QUEUE_STRUCT         WAITING_TASKS,
  _mqx_uint            VALID,
  _mqx_uint            VALUE,
  _mqx_uint            FLAGS,
  _mqx_uint            AUTO
} LWEVENT_STRUCT;
```

See also :

- _lwevent_clear
- _lwevent_create
- _lwevent_destroy
- _lwevent_set
- _lwevent_set_auto_clear
- _lwevent_wait_for
- _lwevent_wait_ticks
- _lwevent_wait_until

Table 3-2. Structure LWEVENT_STRUCT member description

Member	Description
LINK	Queue data structures.
WAITING_TASKS	Queue of tasks waiting for event bits to be set.
VALID	Validation stamp.
VALUE	Current bit value of the lightweight event.
FLAGS	Flags associated with the light weight event.
AUTO	Mask specifying lightweight event bits that are configured as auto-clear.

3.3 LWLOG_ENTRY_STRUCT

Entry in kernel log or a lightweight log.

Description :

Source :

Declaration :

```
typedef struct
{
  _mqx_uint                       SEQUENCE_NUMBER,
  MQX_TICK_STRUCT                 TIMESTAMP,
  _mqx_max_type                   DATA[LWLOG_MAXIMUM_DATA_ENTRIES],
  struct lwlog_entry_struct_PTR_ NEXT_PTR
} LWLOG_ENTRY_STRUCT;
```

Table 3-3. Structure LWLOG_ENTRY_STRUCT member description

Member	Description
SEQUENCE_NUMBER	Sequence number for the entry.
TIMESTAMP	The time in tick time at which the entry was written **if MQX is configured at compile time to timestamp in ticks.**
DATA	Data for the entry.
NEXT_PTR	Pointer to the next lightweight-log entry.

3.4 LWMSGQ_STRUCT

This structure is used to store a circular long word queue.

Description :

The structure must be the LAST if it is included into another data structure, as the queue falls off of the end of this structure.

Source :

Declaration :

```
typedef struct
{
  QUEUE_ELEMENT_STRUCT LINK,
  QUEUE_STRUCT         WAITING_WRITERS,
  QUEUE_STRUCT         WAITING_READERS,
  _mqx_uint            VALID,
  _mqx_uint            MSG_SIZE,
  _mqx_uint            MAX_SIZE,
  _mqx_uint            CURRENT_SIZE,
  _mqx_max_type_ptr    MSG_WRITE_LOC,
  _mqx_max_type_ptr    MSG_READ_LOC,
  _mqx_max_type_ptr    MSG_START_LOC,
  _mqx_max_type_ptr    MSG_END_LOC
} LWMSGQ_STRUCT;
```

Table 3-4. Structure LWMSGQ_STRUCT member description

Member	Description
LINK	Queue data structures.
WAITING_WRITERS	A Queue of task descriptors waiting to write.
WAITING_READERS	A Queue of task descriptors waiting to read.
VALID	The validity check field.

Table continues on the next page...

Table 3-4. Structure LWMSGQ_STRUCT member description (continued)

Member	Description
MSG_SIZE	The size of the message chunk in the queue in _mqx_max_type's.
MAX_SIZE	The maximum number of msgs for the queue, as specified in queue's initialization.
CURRENT_SIZE	The current number of messages in the queue.
MSG_WRITE_LOC	Next message location to write to.
MSG_READ_LOC	Next message location to read from.
MSG_START_LOC	Starting location of messages.
MSG_END_LOC	Location past end of messages.

3.5 LWSEM_STRUCT

Lightweight semaphore.

Description :

This structure defines a lightweight semaphore.

These sempahores implement a simple counting semaphore.

Tasks wait on these semaphores in a FIFO manner.

Priority inheritance is NOT implemented for these semaphores.

The semaphores can be embedded into data structures similarly to mutexes.

Source :

Declaration :

```
typedef struct
{
  struct lwsem_struct_PTR_ NEXT,
  struct lwsem_struct_PTR_ PREV,
  QUEUE_STRUCT             TD_QUEUE,
  _mqx_uint                VALID,
  _mqx_int                 VALUE
} LWSEM_STRUCT;
```

See also :

- _lwsem_create
- _lwsem_create_hidden
- _lwsem_destroy
- _lwsem_poll
- _lwsem_post

- _lwsem_wait
- _lwsem_wait_for
- _lwsem_wait_ticks
- _lwsem_wait_until

Table 3-5. Structure LWSEM_STRUCT member description

Member	Description
NEXT	Pointer to the next lightweight semaphore in the list of lightweight semaphores.
PREV	Pointer to the previous lightweight semaphore in the list of lightweight semaphores.
TD_QUEUE	Manages the queue of tasks that are waiting for the lightweight semaphore. The NEXT and PREV fields in the task descriptors link the tasks.
VALID	When MQX creates the lightweight semaphore, it initializes the field. When MQX destroys the lightweight semaphore, it clears the field.
VALUE	Count of the semaphore. MQX 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 puts the task in the lightweight semaphore queue until the count is a non-zero value.the semaphore value.

3.6 LWTIMER_PERIOD_STRUCT

Lightweight timer queue.

Description :

This structure controls any number of lightweight timers wishing to be executed at the periodic rate defined by this structure. The periodic rate will be a multiple of the BSP_ALARM_RESOLUTION.

Source :

Declaration :

```
typedef 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;
```

See also :

- _lwtimer_add_timer_to_queue

- _lwtimer_cancel_period
- _lwtimer_create_periodic_queue
- LWTIMER_STRUCT

Table 3-6. Structure LWTIMER_PERIOD_STRUCT member description

Member	Description
LINK	Queue of lightweight timers.
PERIOD	The period of this group of timers (in ticks), a multiple of BSP_ALARM_RESOLUTION.
EXPIRY	Number of ticks that have elapsed in this period.
WAIT	Number of ticks to wait before starting to process this queue.
TIMERS	A queue of timers to expire at this periodic rate.
TIMER_PTR	Pointer to the last timer on the queue that was processed.
VALID	When the timer queue is created, MQX initializes the field. When the queue is cancelled, MQX clears the field.

3.7 LWTIMER_STRUCT

Lightweight timer.

Description :

This structure defines a light weight timer. These timers implement a system where the specified function will be called at a periodic interval.

Source :

Declaration :

```
typedef struct
{
  QUEUE_ELEMENT_STRUCT LINK,
  _mqx_uint            RELATIVE_TICKS,
  _mqx_uint            VALID,
  LWTIMER_ISR_FPTR     TIMER_FUNCTION,
  pointer              PARAMETER,
  pointer              PERIOD_PTR
} LWTIMER_STRUCT;
```

See also :

- _lwtimer_add_timer_to_queue
- _lwtimer_cancel_timer
- LWTIMER_PERIOD_STRUCT

Table 3-7. Structure LWTIMER_STRUCT member description

Member	Description
LINK	Queue data structures.
RELATIVE_TICKS	The relative number of ticks until this timer is to expire.
VALID	When the timer is added to the timer queue, MQX initializes the field. When the timer or the timer queue that the timer is in is cancelled, MQX clears the field.
TIMER_FUNCTION	Function that is called when the timer expires.
PARAMETER	Parameter that is passed to the timer function.
PERIOD_PTR	Pointer to the lightweight timer queue to which the timer is attatched.

3.8 MQX_TICK_STRUCT

This structure defines how time is maintained in the system.

Description :

Time is kept internally in the form of ticks. This is a 64 bit field which is maintained in an array whose size is dependent upon the PSP. HW_TICKS is used to track time between ticks (timer interrupts).

Source :

Declaration :

```
typedef struct
{
  _mqx_uint TICKS[MQX_NUM_TICK_FIELDS],
  uint_32   HW_TICKS
} MQX_TICK_STRUCT;
```

See also :

- _lwevent_wait_for
- _lwevent_wait_until
- _lwmsgq_receive
- _lwsem_wait_for
- _lwsem_wait_until
- _time_delay_for
- _time_delay_until
- _time_diff_ticks
- _time_diff_ticks_int32
- _time_get_elapsed_ticks
- _time_get_elapsed_ticks_fast
- _time_init_ticks

Table 3-8. Structure MQX_TICK_STRUCT member description

Member	Description
TICKS	Ticks since MQX started. The field is a minimum of 64 bits; the exact size depends on the PSP.
HW_TICKS	Hardware ticks (timer counter increments) between ticks. The field increases the accuracy over counting the time simply in ticks.

3.9 MQXLITE_INITIALIZATION_STRUCT

MQX initialization structure for each processor.

Description :

This structure defines the information required to be passed to MQX Lite at initialization time.

When an application starts MQX on each processor, it calls _mqx() (or _mqxlite_init in case of MQX Lite) with the MQX initialization structure.

Source :

Declaration :

```
typedef struct
{
  _mqx_uint                PROCESSOR_NUMBER,
  pointer                  START_OF_KERNEL_MEMORY,
  pointer                  END_OF_KERNEL_MEMORY,
  _mqx_uint                INTERRUPT_STACK_SIZE,
  TASK_TEMPLATE_STRUCT_PTR TASK_TEMPLATE_LIST,
  _mqx_uint                MQX_HARDWARE_INTERRUPT_LEVEL_MAX,
  pointer                  INTERRUPT_STACK_LOCATION,
  _mqx_uint                RESERVED[3]
} MQXLITE_INITIALIZATION_STRUCT;
```

See also :

- _mqxlite_init
- TASK_TEMPLATE_STRUCT

Table 3-9. Structure MQXLITE_INITIALIZATION_STRUCT member description

Member	Description
PROCESSOR_NUMBER	Application-unique processor number of the processor. Minimum is 1, maximum is 255. (Processor number 0 is reserved and is used by tasks to indicate their local processor.)

Table continues on the next page...

Table 3-9. Structure MQXLITE_INITIALIZATION_STRUCT member description (continued)

Member	Description
START_OF_KERNEL_MEMORY	Lowest address from which MQX allocates dynamic memory and task stacks.
END_OF_KERNEL_MEMORY	Highest address from which MQX allocates dynamic memory and task stacks. It is the application's responsibility to allocate enough memory for all tasks.
INTERRUPT_STACK_SIZE	The size of the interrupt stack. This is the maximum amount of stack space used by all interrupt handlers.
TASK_TEMPLATE_LIST	Pointer to the task template list for the processor. The default name for the list is MQX_template_list[].
MQX_HARDWARE_INTERRUPT_LEVEL_MAX	The maximum hardware interrupt priority level of MQX. All tasks and interrupts run at lower priority (Applicable to CPUs with multiple interrupt levels only).
INTERRUPT_STACK_LOCATION	The location of the interrupt stack (if not NULL).
RESERVED	Reserved field. Reserved for future enhancements to MQX; each element of the array must be initialized to 0.

3.10 MUTEX_ATTR_STRUCT

Mutex attributes, which are used to initialize a mutex.

Description :

Source :

Declaration :

```
typedef struct
{
  _mqx_uint SCHED_PROTOCOL,
  _mqx_uint VALID,
  _mqx_uint PRIORITY_CEILING,
  _mqx_uint COUNT,
  _mqx_uint WAIT_PROTOCOL
} MUTEX_ATTR_STRUCT;
```

See also :

- _mutatr_destroy
- _mutatr_init
- _mutatr_get_priority_ceiling
- _mutatr_set_priority_ceiling
- _mutatr_get_sched_protocol
- _mutatr_set_sched_protocol

- _mutatr_get_spin_limit
- _mutatr_set_spin_limit
- _mutatr_get_wait_protocol
- _mutatr_set_wait_protocol
- _mutex_init

Table 3-10. Structure MUTEX_ATTR_STRUCT member description

Member	Description
SCHED_PROTOCOL	Scheduling protocol. A task using this mutex should follow when it owns the mutex. One of the following: - MUTEX_NO_PRIO_INHERIT - MUTEX_PRIO_INHERIT - MUTEX_PRIO_PROTECT - MUTEX_PRIO_INHERIT | MUTEX_PRIO_PROTECT
VALID	A validation field for mutexes. When a task calls _mutatr_init, MQX sets the field to MUTEX_VALID and does not change it. If the field changes, MQX considers the attributes invalid. The function _mutatr_init sets the field to TRUE; _mutatr_destroy sets it to FALSE.
PRIORITY_CEILING	Priority of the mutex; applicable only if the scheduling protocol is priority protect.
COUNT	Number of spins to use if the waiting protocol is limited spin.
WAIT_PROTOCOL	The waiting protocol a task using this mutex should follow when a mutex is not available. One of the following: - MUTEX_SPIN_ONLY If the mutex is already locked, MQX timeslices the task until another task unlocks the mutex. - MUTEX_LIMITED_SPIN If the mutex is already locked, MQX timeslices the task for a number of times before the lock attempt fails. If this is set, the spin limit should be set. - MUTEX_QUEUEING If the mutex is already locked, MQX blocks the task until another task unlocks the mutex, at which time MQX gives the mutex to the first task that requested it. - MUTEX_PRIORITY_QUEUEING If the mutex is already locked, MQX blocks the task until another task unlocks the mutex, at which time MQX gives the mutex to the highest-priority task that is waiting for it.

3.11 MUTEX_STRUCT

This structure defines the mutual exclusion (MUTEX) data structure.

Description :

Source :

Declaration :

```
typedef struct
{
  QUEUE_ELEMENT_STRUCT LINK,
  _mqx_uint            PROTOCOLS,
  _mqx_uint            VALID,
  _mqx_uint            PRIORITY_CEILING,
  _mqx_uint            COUNT,
  uint_16              DELAYED_DESTROY,
  uchar                LOCK,
  uchar                FILLER,
  QUEUE_STRUCT         WAITING_TASKS,
  pointer              OWNER_TD,
  _mqx_uint            BOOSTED
} MUTEX_STRUCT;
```

See also :

- _mutex_destroy
- _mutex_get_priority_ceiling
- _mutex_get_wait_count
- _mutex_init
- _mutex_lock
- _mutex_set_priority_ceiling
- _mutex_try_lock
- _mutex_unlock
- MUTEX_ATTR_STRUCT

Table 3-11. Structure MUTEX_STRUCT member description

Member	Description
LINK	Link pointers to maintain a list of mutexes in the kernel.
PROTOCOLS	Waiting protocol (most significant word) and scheduling protocol (least significant word) for the mutex.
VALID	A validation field for mutexes. When a task calls _mutex_init, MQX sets the field to MUTEX_VALID and does not change it. If the field changes, MQX considers the mutex invalid.
PRIORITY_CEILING	Priority of the mutex. If the scheduling protocol is priority protect, MQX grants the mutex only to tasks with at least this priority.

Table continues on the next page...

Table 3-11. Structure MUTEX_STRUCT member description (continued)

Member	Description
COUNT	Maximum number of spins. The field is used only if the waiting protocol is limited spin.
DELAYED_DESTROY	TRUE if the mutex is being destroyed.
LOCK	Most significant bit is set when the mutex is locked.
FILLER	An alignment filler.
WAITING_TASKS	A queue of tasks waiting for the mutex. If PRIORITY_INHERITANCE is set, the queue is in priority order; otherwise it is in FIFO order.
OWNER_TD	Task descriptor of the task that has locked the mutex.
BOOSTED	Number of times that MQX has boosted the priority of the task that has locked the mutex.

3.12 QUEUE_ELEMENT_STRUCT

Header for a queue element.

Description :

This structure is required in each queue element. The address of this structure is used to enqueue, dequeue elements.

Source :

Declaration :

```
typedef struct
{
  struct queue_element_struct_PTR_ NEXT,
  struct queue_element_struct_PTR_ PREV
} QUEUE_ELEMENT_STRUCT;
```

See also :

- QUEUE_STRUCT

Table 3-12. Structure QUEUE_ELEMENT_STRUCT member description

Member	Description
NEXT	Pointer to the next element in queue.
PREV	Pointer to the previous element in queue.

3.13 QUEUE_STRUCT

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

Description :

This structure represents a generic queue head structure. Each queue element is made up of a data structure consisting of a NEXT pointer followed by a PREV pointer. Thus any type of element may be queued onto this queue.

Source :

Declaration :

```
typedef struct
{
  struct queue_element_struct_PTR_ NEXT,
  struct queue_element_struct_PTR_ PREV,
  uint_16                          SIZE,
  uint_16                          MAX
} QUEUE_STRUCT;
```

See also :

- _queue_test
- QUEUE_ELEMENT_STRUCT

Table 3-13. Structure QUEUE_STRUCT member description

Member	Description
NEXT	Pointer to the next element in queue. If there are no elements in the queue, the field is a pointer to the structure itself.
PREV	Pointer to the previous element in queue. If there are no elements in the queue, the field is a pointer to the structure itself.
SIZE	Current number of elements in the queue.
MAX	Maximum number of elements that queue can hold. If the field is 0, the number is unlimited.

3.14 TASK_TEMPLATE_STRUCT

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

Description :

The task template list is an array of these structures, terminated by a zero-filled element. The MQX initialization structure contains a pointer to the list.

Source :

Declaration :

```
typedef 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,
  uint_32    CREATION_PARAMETER
} TASK_TEMPLATE_STRUCT;
```

See also :

- _task_get_template_ptr

Table 3-14. Structure TASK_TEMPLATE_STRUCT member description

Member	Description
TASK_TEMPLATE_INDEX	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.
TASK_ADDRESS	Pointer to the root function for the task. This function will be called when a task is created with the task template index above. The task is deleted when this function returns.
TASK_STACKSIZE	The amount of stack space required by this task.
TASK_PRIORITY	Software priority of the task. Priorities start at 0, which is the highest priority. 1, 2, 3, and so on are progressively lower priorities.
TASK_NAME	Pointer to a string name for tasks that MQX creates from the template.
TASK_ATTRIBUTES	Possible attributes for the task. Possible bit values are: MQX_AUTO_START_TASK - When MQX starts, it creates one instance of the task. MQX_FLOATING_POINT_TASK - task uses the floating point co-processor. MQX also saves floating-point registers as part of the task's context. MQX_TIME_SLICE_TASK - MQX uses round robin scheduling for the task (the default is FIFO scheduling).
CREATION_PARAMETER	Parameter passed to tasks that MQX creates from the template.

Chapter 4
Version history

Table 4-1. Version history table

Revision	Date	Features
1.0	20.9.2012	Initial release.
1.1	1.10.2012	MQXLite non-relevant functions removed.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

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

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor 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 Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claims alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-complaint and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.

For information on Freescale's Environmental Products program, go to http://www.freescale.com/epp.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© 2012 Freescale Semiconductor, Inc.

Document Number: MQXLITE_RM
Rev. Version 1.1