MQX Dealloc Test Suite Specification

# Objectives

Objective of this test suite is to confirm proper resource de-allocation.

## Reference documentation

| Document Name | Version |
| --- | --- |
| 1. MQX MFS User Guide | 2.2 |

Table 1. Reference documentation

## Test environment

|  |  |
| --- | --- |
| **Software Item Name** | **Version** |
| CodeWarrior | 10.1 |
| IAR | 6.10.1 |

Table 2. Required software resources

|  |  |
| --- | --- |
| **Hardware Item Name** | **Version** |
| TWR – K60N512 | N/A |
| TWR – MCF51CN | N/A |

Table 3. Required hardware resources

# General API

|  |  |
| --- | --- |
| *API name* | *Test app containing API* |
| \_mem\_create\_pool | memory |
| \_mem\_alloc | memory |
| \_mem\_alloc\_system | memory, lwevent, lwmsgq, lwsem, mutex |
| \_mem\_alloc\_from | memory |
| \_event\_create, \_event\_create\_fast, \_event\_create\_component, \_event\_destroy, \_event\_destroy\_fast, \_event\_open, \_event\_open\_fast | event |
| \_log\_create, \_log\_destroy, \_log\_create\_component | log |
| \_lwevent\_create, lwevent\_destroy | lwevent |
| \_lwlog\_create, \_lwlog\_destroy, \_lwlog\_create\_component | lwlog |
| \_lwmsgq\_init, \_lwmsgq\_destroy1 | lwmsgq |
| \_lwsem\_create, \_lwsem\_destroy | lwsem |
| \_msg\_create\_component, \_msg\_pool\_create, \_msg\_open, \_msg\_send, \_msg\_pool\_destroy, | message |
| \_mutatr\_init, \_mutex\_create\_component, \_mutex\_init, \_mutex\_destroy | mutex |
| \_partition\_create\_component, \_partition\_get\_free\_blocks, \_partition\_alloc, \_partition\_destroy | part |
| \_sem\_create\_component, \_sem\_create, \_sem\_open, \_sem\_destroy | sem |

# specifications

Tests are designed to use both MEM and LWMEM allocator. So there are 2 sets of tests created: dealloc\_mem and dealloc\_lwmem. Since both sets are sharing same code, code is located in dealloc\_lwmem folder. Dealloc\_mem contains only user\_config.h for various boards. Difference between user configs for MEM and LWMEM are as follows:

MEM:

* MQX\_USE\_MEM = 1
* MQX\_USE\_LWMEM = 0
* MQX\_USE\_LWMEM\_ALLOCATOR = 0

LWMEM:

* MQX\_USE\_MEM = 0
* MQX\_USE\_LWMEM = 1
* MQX\_USE\_LWMEM\_ALLOCATOR = 1

## Test app – EVENT

### Test case #1 – Testing event deallocation after task exits

At first an event component is created. This component holds information of all created events. Amount of free memory and count of already created events are stored. TEST\_TASK is called. This task creates and opens 4 event (2 using create/open and 2 using create\_fast/open\_fast). Since events are crated in system memory pool, size of created components must be saved for later calculations. After TEST\_TASK exits created events should be saved in KERNEL\_DATA and amount of free memory should decrease by value saved in TEST\_TASK as *system\_owned*. CLEAN\_TASK is executed. This task destroys events that were created in TEST\_TASK.

Expected result: After TEST\_TASK exits, amount of free memory should decrease only by amount occupied by created events. That means memory allocated for task descriptor, task stack and event connections (crated by open functions) should be released. All created events should be saved in KERNEL\_DATA. After CLEAN\_TASK is finished amount of free memory and count of created events should be the same as at the beginning.

API used: \_event\_create\_component, \_event\_create, \_event\_create\_fast, \_event\_open, \_event\_open\_fast, \_event\_destroy, \_event\_destroy\_fast

## Test app - LOG

### Test case # 1 - Testing log deallocation after task exits

At first a log component is created. This component holds information of all created logs. Amount of free memory and count of already created logs are stored. TEST\_TASK is called. This task creates several logs. Exact count is in macro LOGS\_CREATED. Since logs are crated in system memory pool, size of created components must be saved for later calculations. After TEST\_TASK exits created logs should be saved in KERNEL\_DATA and amount of free memory should decrease by value saved in TEST\_TASK as *system\_owned*. CLEAN\_TASK is executed. This task destroys logs that were created in TEST\_TASK.

Expected result: After TEST\_TASK exits, amount of free memory should decrease only by amount occupied by created logs. That means memory allocated for task descriptor and task stack should be released. All created logs should be saved in KERNEL\_DATA. After CLEAN\_TASK is finished amount of free memory and count of created logs should be the same as at the beginning.

API used: \_log\_create\_component, \_log\_create, \_log\_destroy

## Test app – LWEVENT

### Test case # 1 – Testing lwevent deallocation after task exits

Amount of free memory and count of already created lightweight events are stored. TEST\_TASK is called. This task creates several lightweight events. Exact count is in macro LWEVENTS\_CREATED. Since creation of lightweight event doesn’t allocated memory, this is allocated in system memory pool. Size of allocated memory is saved for later calculations. After TEST\_TASK exits created lightweight events should be saved in KERNEL\_DATA and amount of free memory should decrease by value saved in TEST\_TASK as *system\_owned*. CLEAN\_TASK is executed. This task destroys lightweight events that were created, and de-allocate memory from TEST\_TASK.

Expected result: After TEST\_TASK exits, amount of free memory should decrease only by amount occupied by created lightweight events. That means memory allocated for task descriptor and task stack should be released. All created lightweight events should be saved in KERNEL\_DATA. After CLEAN\_TASK is finished amount of free memory and count of created lightweight events should be the same as at the beginning.

API used: \_lwevent\_create, \_lwevent\_destroy, \_mem\_alloc\_system

## Test app – LWLOG

### Test case #1 – Testing lightweight log deallocation after task exits

At first a lightweight log component is created. This component holds information of all created lightweight logs. Amount of free memory and count of already created lightweight logs are stored. TEST\_TASK is called. This task creates several lightweight logs. Exact count is in macro LOGS\_CREATED. Since lightweight logs are crated in system memory pool, size of created components must be saved for later calculations. After TEST\_TASK exits created logs should be saved in KERNEL\_DATA and amount of free memory should decrease by value saved in TEST\_TASK as *system\_owned*. CLEAN\_TASK is executed. This task destroys logs that were created in TEST\_TASK.

Expected result: After TEST\_TASK exits, amount of free memory should decrease only by amount occupied by created lightweight logs. That means memory allocated for task descriptor and task stack should be released. All created logs should be saved in KERNEL\_DATA. After CLEAN\_TASK is finished amount of free memory and count of created lightweight logs should be the same as at the beginning.

API used: \_lwlog\_create\_component, \_lwlog\_create, \_lwlog\_destroy

## Test app – LWMSGQ

### Test case #1 – Testing lightweight message queue deallocation after task exits

Amount of free memory and count of already created lightweight message queues are stored. TEST\_TASK is called. This task creates several lightweight message queues. Exact count is in macro LWMSGQ\_CREATED + 2 (memory for these 2 is not allocated dynamically). Since creation of lightweight event doesn’t allocated memory, this is allocated in system memory pool. Size of created components must be saved for later calculations. After TEST\_TASK exits created lightweight message queues should be saved in KERNEL\_DATA and amount of free memory should decrease by value saved in TEST\_TASK as *system\_owned*. CLEAN\_TASK is executed. This task destroys lightweight message queues that were created in TEST\_TASK.

Expected result: After TEST\_TASK exits, amount of free memory should decrease only by amount occupied by created lightweight message queues. That means memory allocated for task descriptor and task stack should be released. All created logs should be saved in KERNEL\_DATA. After CLEAN\_TASK is finished amount of free memory and count of created lightweight message queues should be the same as at the beginning.

API used: \_lwmsgq\_init, \_lwmsgq\_destroy, \_mem\_alloc\_system

## Test app – LWSEM

### Test case #1 – Testing lightweight semaphore deallocation after task exits

Amount of free memory and count of already created lightweight semaphores are stored. TEST\_TASK is called. This task creates several lightweight semaphores. Exact count is in macro LWSEM\_CREATED+ 2 (memory for these 2 is not allocated dynamically). Since creation of lightweight semaphores doesn’t allocated memory, this is allocated in system memory pool. Size of allocated memory is saved for later calculations. After TEST\_TASK exits created lightweight semaphores should be saved in KERNEL\_DATA and amount of free memory should decrease by value saved in TEST\_TASK as *system\_owned*. CLEAN\_TASK is executed. This task destroys lightweight semaphores that were created, and de-allocate memory from TEST\_TASK.

Expected result: After TEST\_TASK exits, amount of free memory should decrease only by amount occupied by created lightweight semaphores. That means memory allocated for task descriptor and task stack should be released. All created lightweight semaphores should be saved in KERNEL\_DATA. After CLEAN\_TASK is finished amount of free memory and count of created lightweight semaphores should be the same as at the beginning.

API used: \_lwsem\_create, \_lwsem\_destroy, \_mem\_alloc\_system

## Test app – MEMORY

### Test case #1 – Testing memory deallocation after task exits

Memory pool with size MEM\_POOL\_SIZE is created. TEST\_TASK allocate memory with size TEST\_ALLOC\_SIZE. Both size macros are defined in test.h. Condition: TEST\_ALLOC\_SIZE > MEM\_POOL\_SIZE/2. After memory is allocated, TETS\_TASK is blocked. Main task tries to allocate memory from pool. After that main task destroys TEST\_TASK and tries to allocate memory from pool once more.

Expected results: When TEST\_TASK is blocked, attempt to allocate memory from pool should fail. After TEST\_TASK is destroyed, allocated memory should be released, and allocation should pass.

API used: \_mem\_create\_pool, \_mem\_alloc\_from

### Test case #2 – Dealloc private memory

Amount of free memory is saved. RUN\_DRY task allocate all available memory from private memory pool. After this task is finished, amount of free memory is captured.

Expected result: amount of free memory after RUN\_DRY task should be same as prior calling this task.

API used: \_mem\_alloc

### Test case #3 – Dealloc system memory

Amount of free memory is saved. RUN\_DRY task is blocked at beginning, so main task can calculate amount of memory used for task (task descriptor, stack) RUN\_DRY then allocate all available memory from system pool and block itself. Main task tries to allocate memory from system pool. Main task destroys RUN\_DRY task.

Expected result: Allocated memory from RUN\_DRY task should stay allocated after task is finished.

API used: \_mem\_alloc\_system

## Test app – MESSAGE

### Test case #1 – Testing message deallocation after task exits

At first a message component is created. This component holds information of all created message pools. Amount of free memory is stored. TEST\_TASK is called. This task creates two message pools. Since message pools are crated in system memory pool, size of created components must be saved for later calculations. TEST\_TASK opens one pool, allocate memory for one message and send one message into queue. After TEST\_TASK exits amount of free memory should decrease by value saved in TEST\_TASK as *system\_owned*. CLEAN\_TASK is executed. This task destroys message pools that were created in TEST\_TASK.

Expected result: After TEST\_TASK exits, amount of free memory should decrease only by amount occupied by created message pools. That means memory allocated for task descriptor and task stack should be released. All created message pools should be saved in KERNEL\_DATA. After CLEAN\_TASK is finished amount of free memory should be the same as at the beginning.

API used: \_msg\_create\_component, \_msg\_create\_pool, \_msg\_open, \_msg\_alloc, \_msg\_send, \_msgpool\_destroy

## Test app – MUTEX

### Test case #1 – Testing mutex deallocation after task exits

At first mutex attribute structure is initialized and mutex component is created. This component holds information of all created mutexes. Amount of free memory and count of already created mutexes are stored. TEST\_TASK is called. This task creates several mutexes. Exact count is in macro MUTEX\_CREATED. Since creation of mutex doesn’t allocated memory, this is allocated in system memory pool. Size of allocated memory is saved for later calculations. After TEST\_TASK exits created mutexes should be saved in KERNEL\_DATA and amount of free memory should decrease by value saved in TEST\_TASK as *system\_owned*. CLEAN\_TASK is executed. This task destroys mutexes that were created, and de-allocate memory from TEST\_TASK.

Expected result: After TEST\_TASK exits, amount of free memory should decrease only by amount occupied by created mutexes. That means memory allocated for task descriptor and task stack should be released. All created mutexes should be saved in KERNEL\_DATA. After CLEAN\_TASK is finished amount of free memory and count of created mutexes should be the same as at the beginning.

API used: \_mutatr\_init, \_mutex\_create\_component, \_mutex\_init, \_mutex\_destroy, \_mem\_alloc\_system, \_mem\_free

## Test app – PART

### Test case #1 – Testing partition deallocation after task exits

At first partition component is created. This component holds information of all created partitions. Amount of free memory is stored. TEST\_TASK is called. This task creates partition with BLOCK\_CNT blocks with BLOCK\_SIZE bytes each. Since partitions are created in system memory, size of allocated memory is saved for later calculations. TEST\_TASK allocate two blocks from created partition and checks for free blocks count. CLEAN\_TASK is executed. This task destroys partition that was created in TEST\_TASK.

Expected result: After TEST\_TASK exits, amount of free memory should decrease only by amount occupied by created partition. Blocks allocated in TEST\_TASK from partitions should be released. After CLEAN\_TASK is finished amount of free memory should be the same as at the beginning.

API used: \_partition\_create\_component, \_partition\_create, \_partition\_alloc, \_partition\_get\_free\_blocks, \_partition\_destroy

## Test app – SEM

### Test case #1 – Testing semaphore deallocation after task exits

At first a semaphore component is created. This component holds information of all created semaphores. Amount of free memory and count of already created semaphores are stored. TEST\_TASK is called. This task creates and opens 4 semaphores (2 using create/open and 2 using create\_fast/open\_fast). Since semaphores are crated in system memory pool, size of created components must be saved for later calculations. After TEST\_TASK exits created semaphores should be saved in KERNEL\_DATA and amount of free memory should decrease by value saved in TEST\_TASK as *system\_owned*. CLEAN\_TASK is executed. This task destroys semaphores that were created in TEST\_TASK.

Expected result: After TEST\_TASK exits, amount of free memory should decrease only by amount occupied by created semaphores. That means memory allocated for task descriptor, task stack and semaphore connections (crated by open functions) should be released. All created semaphores should be saved in KERNEL\_DATA. After CLEAN\_TASK is finished amount of free memory and count of created semaphores should be the same as at the beginning.

API used: \_sem\_create\_component, \_sem\_create, \_sem\_create\_fast, \_sem\_open, \_sem\_open\_fast, \_sem\_destroy, \_sem\_destroy\_fast

**Revision SHEET**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision Date** | **Author** | **Description of Revision & Writer** | **Spec Coord.** |
| 24.10.2011 | Michal Starecek | Initial version |  |