uMT Reference Manual

This documents is the reference manual for the uMT callable primitives (implemented as C++ methods). This is a preliminary version which might not be including all the calls: please refer to ***uMT.h*** for the full list.

A Getting Started document is available as well.

uMT is a young software and, as consequence, no extensive use (and debugging) has been done yet. As a consequence users must be aware of this and they are suggested to contact the Author in case of bugs or issues.

Please also note that this an EDUCATIONAL tool, not designed for industrial or state-of-the-art application (nor life or mission critical application!!!) and not fully optimized.

uMT version 2.0.0 – 01 June 2017
Document version: 2.0

uMT

Micro Multi Tasker (uMT) is a preemptive, timesharing, soft real-time (not deterministic) multi tasker specifically designed for the Arduino environment. uMT currently works with the AVR microprocessor family (***Arduino Uno*** and ***Arduino Mega2560***) and for the SAM architecture (***Arduino Due***). A porting to SAMD architecture (Arduino Zero) has been planned for the future.

uMT is a simplified rewriting of a 30 years old multi tasker developed by the Author in his youth (!) which was originally designed for the Intel 8086 architecture (and then ported to protected mode 80386, M68000, MIPS R3). uMT is significantly simpler than the original and it has been designed for ease of use and power for the small environment of Arduino microprocessors and boards. Last, uMT is written in a simple C++ versus its C language ancestor to hide complexity and increase ease of use.

Table of Contents

KERNEL MANAGEMENT

Kernel management provides the following primitives:

NAME	DESCRIPTION
Kn_Start()	Start uMT Kernel
Kn_Inited()	Return if uMT has been initialized
Kn_PrintInternals()	Print uMT internal kernel tables on Serial
Kn_GetConfiguration()	Return uMT kernel configuration
Kn_PrintConfiguration()	Print uMT kernel configuration on Serial
isr_Kn_GetVersion()	Return uMT version
isr_Kn_FatalError()	Print an optional message to Serial, a fatal error message and reboot
isr_Kn_Reboot()	Reboot (restart) the system
isr_Kn_IntLock()	Disable global interrupts and return previous interrupt mask
isr_Kn_IntUnlock()	Restore previous interrupt mask (possibly enabling interrupts)
isr_Kn_GetKernelTick()	Return internal kernel counter (usually in milliseconds)

NAME

Kn_Start()

SYNTAX

```
Errno_t       Kn_Start();
Errno_t       Kn_Start(Bool_t _TimeSharingEnabled);
Errno_t       Kn_Start(uMTcfg &Cfg);
```

PARAMETERS

NAME	IN/OUT	DESCRIPTION
TimeSharingEnabled	IN	TRUE to enable, FALSE to disable
Cfg	IN	Configuration setting

DESCRIPTION

Initialize the uMT subsystem.

RETURNED VALUE

CODE	DESCRIPTION
E_SUCCESS	Success
E_INVALID_MAX_TASK_NUM	
E_INVALID_MAX_TIMER_NUM	
E_INVALID_MAX_SEM_NUM	
E_NO_MORE_MEMORY	

EXAMPLE

TASK MANAGEMENT

Task management provides the following primitives:

NAME	DESCRIPTION
Tk_CreateTask()	Create a new task
Tk_DeleteTask()	Delete an existing task
Tk_StartTask()	Start a just created task
Tk_GetMyTid	Return my task id number
Tk_Yield()	Release the processor and perform a rescheduling (with a possible task round robin)
Tk_ReStartTask()	Restart a task
Tk_GetActiveTaskNo()	Return the total number of active tasks (excluding IDLE task)
Tk_SetTimeSharing()	Set the Timesharing flag
Tk_GetTimeSharing()	Return the Timesharing status flag
Tk_SetPreemption()	Set preemption flag
Tk_GetPreemption()	Return preemption status flag
Tk_SetBlinkingLED()	Set BlinkingLED flag
Tk_GetBlinkingLED()	Return BlinkingLED status flag
Tk_SetPriority()	Set task priority
Tk_GetPriority()	Return task priority
Tk_SetParam()	Set task launch parameter
Tk_GetParam()	Return own launch parameter

NAME

Tk_CreateTask()

SYNTAX

```
Errno_t Tk_CreateTask(
          FuncAddress_t StartAddress,
          TaskId_t &Tid,
          FuncAddress_t _BadExit = NULL,
          StackSize_t _StackSize = 0);
```

PARAMETERS

NAME	IN/OUT	DESCRIPTION
StartAddress	IN	The start address (a C/C++ function pointer) of the new task.
Tid	OUT	The newly created task ID.
_BadExit	IN	The address (a C/C++ function pointer) of a function which will be called if the task main entry point incorrectly performs a “return”. If missing, the uMT BadExit() routine will be called (rebooting the system).
StackSize	IN	The stack size in bytes (missing or 0 to use default value).

DESCRIPTION

Create a new task in a CREATED state. A subsequent call to Tk_StartTask() is required to put the task in the READY to run list.

RETURNED VALUE

CODE	DESCRIPTION
E_SUCCESS	Success
E_INVALID_STACK_SIZE	
E_NOMORE_TASKS	
E_NO_MORE_MEMORY	

EXAMPLE

NAME

Tk_DeleteTask()

SYNTAX

```
Errno_t Tk_DeleteTask(TaskId_t Tid);
Errno_t Tk_DeleteTask();
```

PARAMETERS

NAME	IN/OUT	DESCRIPTION
Tid	IN	The task ID to be deleted

DESCRIPTION

Delete a task. In the form without a parameter, delete the calling task.
Note that Arduino loop() task (Tid 1) cannot be deleted.
Task's stack area is released and the task slot is available for a new task creating.
Any pending timers is released and the deleted task is removed from any queue.

Any other resource owned by the task (including semaphores) are NOT released. As a consequence, a proper clean up logic must be implemented before/after calling ***Tk_DeleteTask***().

RETURNED VALUE

CODE	DESCRIPTION
E_SUCCESS	Success
E_INVALID_TASKID	

EXAMPLE

NAME

Tk_StartTask()

SYNTAX

```
Errno_t       Tk_StartTask(TaskId_t Tid);
```

PARAMETERS

NAME	IN/OUT	DESCRIPTION
Tid	IN	Task ID to start

DESCRIPTION

Start a task previously created.

RETURNED VALUE

CODE	DESCRIPTION
E_SUCCESS	Success
E_INVALID_TASKID	
E_ALREADY_STARTED	

EXAMPLE

SEMAPHORE MANAGEMENT

Semaphore management provides the following primitives:

NAME	DESCRIPTION
Sm_Claim()	
Sm_Release()	
Sm_SetQueueMode()	
isr_Sm_Claim()	
isr_Sm_Release()	
isr_p_Sm_Release()	

NAME

Sm_Claim()

SYNTAX

```
Errno_t       Sm_Claim(SemId_t Sid, uMToptions_t Options, Timer_t timeout=(Timer_t)0);
Errno_t       Sm_Claim(SemId_t Sid, uMToptions_t Options);
Errno_t       isr_Sm_Claim(SemId_t Sid);
```

PARAMETERS

NAME	IN/OUT	DESCRIPTION
Sid	IN	Semaphore id, in the range 1-kernelCfg.Semaphores_Num if dynamically configured, 1- uMT_MAX_SEM_NUM if not.
Options	IN	uMT_NOWAIT to return immediately if Semaphore not available. uMT_WAIT otherwise
timeout	IN	Number of milliseconds to wait if Semaphore not available

DESCRIPTION

Try to acquire a semaphore.

RETURNED VALUE

CODE	DESCRIPTION
E_SUCCESS	Success
E_INVALID_SEMID	
E_WOULD_BLOCK	Semaphore not available and uMT_NOWAIT specified as “Options”.
E_TIMEOUT	Timeout expired (Semaphore not taken)

EXAMPLE

NAME

Sm_Release()

SYNTAX

```
Errno_t       Sm_Release(SemId_t Sid);
Errno_t       isr_Sm_Release(SemId_t Sid);
Errno_t       isr_p_Sm_Release(SemId_t Sid);
```

PARAMETERS

NAME	IN/OUT	DESCRIPTION
Sid	IN	Semaphore id, in the range 1-kernelCfg.Semaphores_Num if dynamically configured, 1- uMT_MAX_SEM_NUM if not.

DESCRIPTION

Release a Semaphore. Task does NOT need to have previously acquired the semaphore.

RETURNED VALUE

CODE	DESCRIPTION
E_SUCCESS	Success
E_INVALID_SEMID	
E_OVERFLOW_SEM	Semaphore value overflow.

EXAMPLE

EVENT MANAGEMENT

Event management provides the following primitives:

NAME	DESCRIPTION
Ev_Receive()	
Ev_Send()	
isr_Ev_Send()	
isr_p_Ev_Send()	

NAME

Ev_Receive()

SYNTAX

```
Errno_t       Ev_Receive(Event_t   eventin, uMToptions_t flags, Event_t *eventout, Timer_t
timeout=(Timer_t)0);
Errno_t       Ev_Receive(Event_t   eventin, uMToptions_t flags, Event_t *eventout);
```

PARAMETERS

NAME	IN/OUT	DESCRIPTION
eventin	IN	Event mask to be received.
flags	IN	Event Mode: uMT_ANY (any event) uMT_ALL (all events are required) Wait Mode (to be ORed with Event Mode) uMT_NOWAIT (do not wait if events are available, return immediately)
eventout	OUT	Events received.
timeout	IN	If not zero, the maximum number of milliseconds to wait

DESCRIPTION

Try to receive the requested Events.

RETURNED VALUE

CODE	DESCRIPTION
E_SUCCESS	Success
E_WOULD_BLOCK	Events not available and uMT_NOWAIT specified as “flags” (eventout is loaded with received events).
E_TIMEOUT	Timeout expired (eventout is loaded with received events).

EXAMPLE

NAME

Ev_Send()

SYNTAX

```
Errno_t       Ev_Send(TaskId_t Tid, Event_t Event);
Errno_t       isr_Ev_Send(TaskId_t Tid, Event_t Event);
Errno_t       isr_p_Ev_Send(TaskId_t Tid, Event_t Event);
```

PARAMETERS

NAME	IN/OUT	DESCRIPTION
Tid	IN	The task ID which will receive the Event mask.
Event	IN	Event mask to be sent.

DESCRIPTION

Send the Event mask to the specified task Id.

RETURNED VALUE

CODE	DESCRIPTION
E_SUCCESS	Success
E_INVALID_TASKID	

EXAMPLE

TIMER MANAGEMENT

Timer management provides the following primitives:

NAME	DESCRIPTION
Tm_WakeupAfter()	
Tm_EvAfter()	
Tm_EvEvery()	
Tm_Cancel()	

NAME

Tm_WakeupAfter()

SYNTAX

```
Errno_t Tm_WakeupAfter(Timer_t timeout);
```

PARAMETERS

NAME	IN/OUT	DESCRIPTION
timeout	IN	Number of milliseconds to wait.

DESCRIPTION

Initialize the uMT subsystem.

RETURNED VALUE

CODE	DESCRIPTION
E_SUCCESS	Success

EXAMPLE

NAME

Tm_EvAfter()

SYNTAX

```
Errno_t Tm_EvAfter(Timer_t timeout, Event_t Event, TimerId_t &TmId);
```

PARAMETERS

NAME	IN/OUT	DESCRIPTION
timeout	IN	Number of milliseconds (it cannot be zero).
Event	IN	Event mask to sent
TmId	OUT	Timer ID to be used in a subsequent Tm_Cancel().

DESCRIPTION

Send the specified Event mask to the calling task, after a timeout.

RETURNED VALUE

CODE	DESCRIPTION
E_SUCCESS	Success
E_INVALID_TIMEOUT	
E_NOMORE_TIMERS	

EXAMPLE

NAME

Tm_EvEvery()

SYNTAX

```
Errno_t Tm_EvEvery(Timer_t timeout, Event_t Event, TimerId_t &TmId);
```

PARAMETERS

NAME	IN/OUT	DESCRIPTION
timeout	IN	Number of milliseconds (it cannot be zero).
Event	IN	Event mask to sent
TmId	OUT	Timer ID to be used in a subsequent Tm_Cancel().

DESCRIPTION

Send the specified Event mask to the calling task, every milliseconds.

RETURNED VALUE

CODE	DESCRIPTION
E_SUCCESS	Success
E_INVALID_TIMEOUT	
E_NOMORE_TIMERS	

EXAMPLE

NAME

Tm_Cancel()

SYNTAX

```
Errno_t Tm_Cancel(TimerId_t TmId);
```

PARAMETERS

NAME	IN/OUT	DESCRIPTION
TmId	IN	Timer ID returned by previous Tm_xxx() calls.

DESCRIPTION

Cancel a Timer previously created with a Tm_xxx() call.

RETURNED VALUE

CODE	DESCRIPTION
E_SUCCESS	Success
E_INVALID_TIMERID	
E_NOT_OWNED_TIMER	Timer ID is not owned by the caller task.

EXAMPLE