CMSIS-RTOS API v2 (RTX 5)

API

```
#include "cmsis_os2.h"
```

1. Kernel Information and Control

Enumerations

```
Enum osKernelState_t {
  osKernelInactive = 0,
  osKernelReady = 1,
  osKernelRunning = 2,
  osKernelLocked = 3,
  osKernelSuspended = 4,
  osKernelError = -1,
  osKernelReserved = 0x7FFFFFFFU
}
```

Kernel state.

Functions

`osStatus_t osKernelInitialize(void)`

Initialize the RTOS Kernel.

`osStatus_t osKernelGetInfo(osVersion_t *version, char *id_buf, uint32_t id_size)`

Get RTOS Kernel Information.

`osKernelState_t osKernelGetState(void)`

Get the current RTOS Kernel state.

`osStatus_t osKernelStart(void)`

Start the RTOS Kernel scheduler.

`int32_t osKernelLock(void)`

Lock the RTOS Kernel scheduler.

`int32_t osKernelUnlock(void)`

Unlock the RTOS Kernel scheduler.

`int32_t osKernelRestoreLock(int32_t lock)`

Restore the RTOS Kernel scheduler lock state.

`uint32_t osKernelSuspend(void)`

Suspend the RTOS Kernel scheduler.

`void osKernelResume(uint32_t sleep_ticks)`

Resume the RTOS Kernel scheduler.

`uint32_t osKernelGetTickCount(void)`

Get the RTOS kernel tick count.

`uint32_t osKernelGetTickFreq(void)`

Get the RTOS kernel tick frequency.

`uint32_t osKernelGetSysTimerCount(void)`

Get the RTOS kernel system timer count.

`uint32_t osKernelGetSysTimerFreq void)`

Get the RTOS kernel system timer frequency.

2. Thread Management

Enumerations

```
enum osThreadState_t {
  osThreadInactive = 0,
  osThreadReady = 1,
  osThreadRunning = 2,
  osThreadBlocked = 3,
  osThreadTerminated = 4,
  osThreadError = -1,
  osThreadReserved = 0x7FFFFFFF
}
```

Thread state.

```
enum osPriority_t {
  osPriorityNone = 0,
  osPriorityIdle = 1,
  osPriorityLow = 8,
  osPriorityLow1 = 8+1,
  osPriorityLow2 = 8+2,
  osPriorityLow3 = 8+3,
  osPriorityLow4 = 8+4,
  osPriorityLow5 = 8+5,
  osPriorityLow6 = 8+6,
  osPriorityLow7 = 8+7,
  osPriorityBelowNormal = 16,
  osPriorityBelowNormal1 = 16+1,
  osPriorityBelowNormal2 = 16+2,
  osPriorityBelowNormal3 = 16+3,
  osPriorityBelowNormal4 = 16+4,
  osPriorityBelowNormal5 = 16+5,
  osPriorityBelowNormal6 = 16+6,
  osPriorityBelowNormal7 = 16+7,
  osPriorityNormal = 24,
  osPriorityNormal1 = 24+1,
  osPriorityNormal2 = 24+2,
  osPriorityNormal3 = 24+3,
  osPriorityNormal4 = 24+4,
  osPriorityNormal5 = 24+5,
  osPriorityNormal6 = 24+6,
  osPriorityNormal7 = 24+7,
  osPriorityAboveNormal = 32,
  osPriorityAboveNormal1 = 32+1,
  osPriorityAboveNormal2 = 32+2,
  osPriorityAboveNormal3 = 32+3,
  osPriorityAboveNormal4 = 32+4,
  osPriorityAboveNormal5 = 32+5,
  osPriorityAboveNormal6 = 32+6,
  osPriorityAboveNormal7 = 32+7,
  osPriorityHigh = 40,
  osPriorityHigh1 = 40+1,
  osPriorityHigh2 = 40+2,
  osPriorityHigh3 = 40+3,
  osPriorityHigh4 = 40+4,
  osPriorityHigh5 = 40+5,
  osPriorityHigh6 = 40+6,
  osPriorityHigh7 = 40+7,
  osPriorityRealtime = 48,
  osPriorityRealtime1 = 48+1,
  osPriorityRealtime2 = 48+2,
  osPriorityRealtime3 = 48+3,
  osPriorityRealtime4 = 48+4,
  osPriorityRealtime5 = 48+5,
  osPriorityRealtime6 = 48+6,
  osPriorityRealtime7 = 48+7,
  osPriorityISR = 56,
  osPriorityError = -1,
  osPriorityReserved = 0x7FFFFFFF
}
```

Priority values.

Typedefs

```
typedef void * osThreadId_t
typedef void(*osThreadFunc_t)(void *argument)
```

Entry point of a thread.

Functions

```
osThreadId_t osThreadNew(osThreadFunc_t func, void *argument, const osThreadAttr_t *attr)
```

Create a thread and add it to Active Threads.

```
const char *osThreadGetName(osThreadId_t thread_id)
```

Get name of a thread.

`osThreadId_t osThreadGetId(void)`
Return the thread ID of the current running thread.
`osThreadState_t osThreadGetState(osThreadId_t thread_id)`
Get current thread state of a thread.
`osStatus_t osThreadSetPriority(osThreadId_t thread_id, osPriority_t priority)`
Change priority of a thread.
`osPriority_t osThreadGetPriority(osThreadId_t thread_id)`
Get current priority of a thread.
`osStatus_t osThreadYield(void)`
Pass control to next thread that is in state READY.
`osStatus_t osThreadSuspend(osThreadId_t thread_id)`
Suspend execution of a thread.
`osStatus_t osThreadResume(osThreadId_t thread_id)`
Resume execution of a thread.
`osStatus_t osThreadDetach(osThreadId_t thread_id)`
Detach a thread (thread storage can be reclaimed when thread terminates).
`osStatus_t osThreadJoin(osThreadId_t thread_id)`
Wait for specified thread to terminate.
`__NO_RETURN void osThreadExit(void)`
Terminate execution of current running thread.
`osStatus_t osThreadTerminate(osThreadId_t thread_id)`
Terminate execution of a thread.
`uint32_t osThreadGetStackSize(osThreadId_t thread_id)`
Get stack size of a thread.
`uint32_t osThreadGetStackSpace(osThreadId_t thread_id)`
Get available stack space of a thread based on stack watermark recording during execution.
`uint32_t osThreadGetCount(void)`
Get number of active threads.
`uint32_t osThreadEnumerate(osThreadId_t *thread_array, uint32_t array_items)`
Enumerate active threads.

3. Thread Flags

Functions

`uint32_t osThreadFlagsSet(osThreadId_t thread_id, uint32_t flags)`
Set the specified Thread Flags of a thread.
`uint32_t osThreadFlagsClear(uint32_t flags)`
Clear the specified Thread Flags of current running thread.
`uint32_t osThreadFlagsGet(void)`
Get the current Thread Flags of current running thread.
`uint32_t osThreadFlagsWait(uint32_t flags, uint32_t options, uint32_t timeout)`
Wait for one or more Thread Flags of the current running thread to become signaled.

4. Event Flags

Typedefs

`typedef void * osEventFlagsId_t`

Functions

`osEventFlagsId_t osEventFlagsNew(const osEventFlagsAttr_t *attr)`
Create and Initialize an Event Flags object.
`uint32_t osEventFlagsSet(osEventFlagsId_t ef_id, uint32_t flags)`
Set the specified Event Flags.
`uint32_t osEventFlagsClear(osEventFlagsId_t ef_id, uint32_t flags)`
Clear the specified Event Flags.
`uint32_t osEventFlagsGet(osEventFlagsId_t ef_id)`
Get the current Event Flags.
`uint32_t osEventFlagsWait(osEventFlagsId_t ef_id, uint32_t flags, uint32_t options, uint32_t timeout)`
Wait for one or more Event Flags to become signaled.
`osStatus_t osEventFlagsDelete(osEventFlagsId_t ef_id)`
Delete an Event Flags object.

```
const char * osEventFlagsGetName(osEventFlagsId_t ef_id)
```
Get name of an Event Flags object.

5. Generic Wait Functions

Functions

```
osStatus_t osDelay(uint32_t ticks)
```
Wait for Timeout (Time Delay).

```
osStatus_t osDelayUntil(uint32_t ticks)
```
Wait until specified time.

6. Timer Management

Typedefs

```
typedef void * osTimerId_t
typedef void(*    osTimerFunc_t )(void *argument)
```
Timer callback function.

Enumerations

```
enum osTimerType_t {
  osTimerOnce = 0,
  osTimerPeriodic = 1
}
```
Timer type.

Functions

```
osTimerId_t osTimerNew(osTimerFunc_t func, osTimerType_t type, void *argument, const
osTimerAttr_t *attr)
```
Create and Initialize a timer.

```
const char * osTimerGetName(osTimerId_t timer_id)
```
Get name of a timer.

```
osStatus_t osTimerStart(osTimerId_t timer_id, uint32_t ticks)
```
Start or restart a timer.

```
osStatus_t osTimerStop(osTimerId_t timer_id)
```
Stop a timer.

```
uint32_t osTimerIsRunning(osTimerId_t timer_id)
```
Check if a timer is running.

```
osStatus_t osTimerDelete(osTimerId_t timer_id)
```
Delete a timer.

7. Mutex Management

Typedefs

```
typedef void * osMutexId_t
```

Functions

```
osMutexId_t osMutexNew(const osMutexAttr_t *attr)
```
Create and Initialize a Mutex object.

```
const char *osMutexGetName(osMutexId_t mutex_id)
```
Get name of a Mutex object.

```
osStatus_t osMutexAcquire(osMutexId_t mutex_id, uint32_t timeout)
```
Acquire a Mutex or timeout if it is locked.

```
osStatus_t osMutexRelease(osMutexId_t mutex_id)
```
Release a Mutex that was acquired by osMutexAcquire.

```
osThreadId_t osMutexGetOwner(osMutexId_t mutex_id)
```
Get Thread which owns a Mutex object.

```
osStatus_t osMutexDelete(osMutexId_t mutex_id)
```
Delete a Mutex object.

8. Semaphores

Typedefs

```
typedef void * osSemaphoreId_t
```

Functions

```
osSemaphoreId_t osSemaphoreNew(uint32_t max_count, uint32_t initial_count, const
osSemaphoreAttr_t *attr)
```

Create and Initialize a Semaphore object.

```
const char *osSemaphoreGetName(osSemaphoreId_t semaphore_id)
```

Get name of a Semaphore object.

```
osStatus_t osSemaphoreAcquire(osSemaphoreId_t semaphore_id, uint32_t timeout)
```

Acquire a Semaphore token or timeout if no tokens are available.

```
osStatus_t osSemaphoreRelease(osSemaphoreId_t semaphore_id)
```

Release a Semaphore token up to the initial maximum count.

```
uint32_t osSemaphoreGetCount(osSemaphoreId_t semaphore_id)
```

Get current Semaphore token count.

```
osStatus_t osSemaphoreDelete(osSemaphoreId_t semaphore_id)
```

Delete a Semaphore object.

9. Memory Pool

Typedefs

```
typedef void * osMemoryPoolId_t
```

Functions

```
osMemoryPoolId_t osMemoryPoolNew(uint32_t block_count, uint32_t block_size, const
osMemoryPoolAttr_t *attr)
```

Create and Initialize a Memory Pool object.

```
const char *osMemoryPoolGetName(osMemoryPoolId_t mp_id)
```

Get name of a Memory Pool object.

```
void *osMemoryPoolAlloc(osMemoryPoolId_t mp_id, uint32_t timeout)
```

Allocate a memory block from a Memory Pool.

```
osStatus_t osMemoryPoolFree(osMemoryPoolId_t mp_id, void *block)
```

Return an allocated memory block back to a Memory Pool.

```
uint32_t osMemoryPoolGetCapacity(osMemoryPoolId_t mp_id)
```

Get maximum number of memory blocks in a Memory Pool.

```
uint32_t osMemoryPoolGetBlockSize(osMemoryPoolId_t mp_id)
```

Get memory block size in a Memory Pool.

```
uint32_t osMemoryPoolGetCount(osMemoryPoolId_t mp_id)
```

Get number of memory blocks used in a Memory Pool.

```
uint32_t osMemoryPoolGetSpace(osMemoryPoolId_t mp_id)
```

Get number of memory blocks available in a Memory Pool.

```
osStatus_t osMemoryPoolDelete(osMemoryPoolId_t mp_id)
```

Delete a Memory Pool object.

10. Message Queue

Functions

```
osMessageQueueId_t osMessageQueueNew(uint32_t msg_count, uint32_t msg_size, const
osMessageQueueAttr_t *attr)
```

Create and Initialize a Message Queue object.

```
const char *osMessageQueueGetName(osMessageQueueId_t mq_id)
```

Get name of a Message Queue object.

```
osStatus_t osMessageQueuePut(osMessageQueueId_t mq_id, const void *msg_ptr, uint8_t
msg_prio, uint32_t timeout)
```

Put a Message into a Queue or timeout if Queue is full.

```
osStatus_t osMessageQueueGet(osMessageQueueId_t mq_id, void *msg_ptr, uint8_t *msg_prio,
uint32_t timeout)
```

Get a Message from a Queue or timeout if Queue is empty.

```
uint32_t osMessageQueueGetCapacity(osMessageQueueId_t mq_id)
```

Get maximum number of messages in a Message Queue.

```
uint32_t osMessageQueueGetMsgSize(osMessageQueueId_t mq_id)
```
Get maximum message size in a Memory Pool.
```
uint32_t osMessageQueueGetCount(osMessageQueueId_t mq_id)
```
Get number of queued messages in a Message Queue.
```
uint32_t osMessageQueueGetSpace(osMessageQueueId_t mq_id)
```
Get number of available slots for messages in a Message Queue.
```
osStatus_t osMessageQueueReset(osMessageQueueId_t mq_id)
```
Reset a Message Queue to initial empty state.
```
osStatus_t osMessageQueueDelete(osMessageQueueId_t mq_id)
```
Delete a Message Queue object.

11. Definitions

Macros

```
#define osWaitForever          0xFFFFFFFFU
```
Wait forever timeout value.
```
#define osFlagsWaitAny         0x00000000U
```
Wait for any flag (default).
```
#define osFlagsWaitAll         0x00000001U
```
Wait for all flags.
```
#define osFlagsNoClear         0x00000002U
```
Do not clear flags which have been specified to wait for.
```
#define osFlagsErrorUnknown    0xFFFFFFFFU
```
osError (-1).
```
#define osFlagsErrorTimeout    0xFFFFFFFEU
```
osErrorTimeout (-2).
```
#define osFlagsErrorResource   0xFFFFFFFDU
```
osErrorResource (-3).
```
#define osFlagsErrorParameter  0xFFFFFFFCU
```
osErrorParameter (-4).
```
#define osFlagsErrorISR        0xFFFFFFFAU
```
osErrorISR (-6).

Enumerations

```
enum osStatus_t {
  osOK = 0,
  osError = -1,
  osErrorTimeout = -2,
  osErrorResource = -3,
  osErrorParameter = -4,
  osErrorNoMemory = -5,
  osErrorISR = -6,
  osStatusReserved = 0x7FFFFFFF
}
```
Status code values returned by CMSIS-RTOS functions.

12. RTX v5 Specific Functions

```
uint32_t osRtxErrorNotify(uint32_t code, void *object_id)
```
OS Error Callback function.
```
void osRtxIdleThread(void *argument)
```
OS Idle Thread.