# CMSIS-RTOS API (RTX)

## API

```
#include "cmsis_os.h"
```

## 1. Kernel Information and Control

**Macros**

```
#define osKernelSysTickFrequency os_tickfreq
```
The RTOS kernel system timer frequency in Hz.
```
#define osKernelSysTickMicroSec(microsec) ((microsec *os_tickus_i) + ((microsec
  *os_tickus_f) >> 16))
```
Convert a microseconds value to a RTOS kernel system timer value.

**Functions**

```
osStatus osKernelInitialize(void)
```
Initialize the RTOS Kernel for creating objects.
```
osStatus osKernelStart(void)
```
Start the RTOS Kernel.
```
int32_t osKernelRunning(void)
```
Check if the RTOS kernel is already started.
```
uint32_t osKernelSysTick(void)
```
Get the RTOS kernel system timer counter.

## 2. Thread Management

**Macros**

```
#define osThreadDef(name, priority, instances, stacksz)
```
Create a Thread Definition with function, priority, and stack requirements.
```
#define osThread(name) &os_thread_def_##name
```
Access a Thread definition.

**Enumerations**

```
enum osPriority {
  osPriorityIdle = -3,
  osPriorityLow = -2,
  osPriorityBelowNormal = -1,
  osPriorityNormal = 0,
  osPriorityAboveNormal = +1,
  osPriorityHigh = +2,
  osPriorityRealtime = +3,
  osPriorityError = 0x84
}
```

**Functions**

```
osThreadId osThreadCreate(const osThreadDef_t *thread_def, void *argument)
```
Create a thread and add it to Active Threads and set it to state READY.
```
osThreadId osThreadGetId(void)
```
Return the thread ID of the current running thread.
```
osStatus osThreadTerminate(osThreadId thread_id)
```
Terminate execution of a thread and remove it from Active Threads.
```
osStatus osThreadSetPriority(osThreadId thread_id, osPriority priority)
```
Change priority of an active thread.
```
osPriority osThreadGetPriority(osThreadId thread_id)
```
Get current priority of an active thread.
```
osStatus osThreadYield(void)
```
Pass control to next thread that is in state READY.

## 3. Timer Management

**Macros**

```
#define osTimerDef(name, function)
```
Define a Timer object.
```
#define osTimer(name) &os_timer_def_##name
```
Access a Timer definition.

**Enumerations**

```
enum os_timer_type {
  osTimerOnce = 0,
  osTimerPeriodic = 1
}
```

**Functions**

```
osTimerId osTimerCreate(const osTimerDef_t *timer_def, os_timer_type type, void
  *argument)
```
Create a timer.
```
osStatus osTimerStart(osTimerId timer_id, uint32_t millisec)
```
Start or restart a timer.
```
osStatus osTimerStop(osTimerId timer_id)
```
Stop a timer.
```
osStatus osTimerDelete(osTimerId timer_id)
```
Delete a timer that was created by osTimerCreate.

## 4. Signal Management

**Functions**

```
int32_t osSignalSet(osThreadId thread_id, int32_t signals)
```
Set the specified Signal Flags of an active thread.
```
int32_t osSignalClear(osThreadId thread_id, int32_t signals)
```
Clear the specified Signal Flags of an active thread.
```
os_InRegs osEvent osSignalWait(int32_t signals, uint32_t millisec)
```
Wait for one or more Signal Flags to become signaled for the current RUNNING thread.

## 5. Mutex Management

**Macros**

```
#define osMutexDef(name)
```
Define a Mutex.
```
#define osMutex(name) &os_mutex_def_##name
```
Access a Mutex definition.

**Functions**

```
osMutexId osMutexCreate(const osMutexDef_t *mutex_def)
```
Create and Initialize a Mutex object.
```
osStatus osMutexWait(osMutexId mutex_id, uint32_t millisec)
```
Wait until a Mutex becomes available.
```
osStatus osMutexRelease(osMutexId mutex_id)
```
Release a Mutex that was obtained by osMutexWait.
```
osStatus osMutexDelete(osMutexId mutex_id)
```
Delete a Mutex that was created by osMutexCreate.

## 6. Semaphore Management

**Macros**

```
#define osSemaphoreDef(name)
```
Define a Semaphore object.
```
#define osSemaphore(name) &os_semaphore_def_##name
```
Access a Semaphore definition.

**Functions**
```
osSemaphoreId osSemaphoreCreate(const osSemaphoreDef_t *semaphore_def, int32_t count)
```
Create and Initialize a Semaphore object used for managing resources.
```
int32_t osSemaphoreWait(osSemaphoreId semaphore_id, uint32_t millisec)
```
Wait until a Semaphore token becomes available.
```
osStatus osSemaphoreRelease(osSemaphoreId semaphore_id)
```
Release a Semaphore token.
```
osStatus osSemaphoreDelete (osSemaphoreId semaphore_id)
```
Delete a Semaphore that was created by osSemaphoreCreate.

## 7. Memory Pool Management

**Macros**
```
#define osPoolDef(name, no, type)
```
Define a Memory Pool.
```
#define osPool(name) &os_pool_def_##name
```
Access a Memory Pool definition.

**Functions**
```
osPoolId osPoolCreate(const osPoolDef_t *pool_def)
```
Create and Initialize a memory pool.
```
void *osPoolAlloc(osPoolId pool_id)
```
Allocate a memory block from a memory pool.
```
void *osPoolCAlloc(osPoolId pool_id)
```
Allocate a memory block from a memory pool and set memory block to zero.
```
osStatus osPoolFree (osPoolId pool_id, void *block)
```
Return an allocated memory block back to a specific memory pool.

## 8. Message Queue Management

**Macros**
```
#define osMessageQDef(name, queue_sz, type)
```
Create a Message Queue Definition.
```
#define osMessageQ(name) &os_messageQ_def_##name
```
Access a Message Queue Definition.

**Functions**
```
osMessageQId osMessageCreate(const osMessageQDef_t *queue_def, osThreadId thread_id)
```
Create and Initialize a Message Queue.
```
osStatus osMessagePut(osMessageQId queue_id, uint32_t info, uint32_t millisec)
```
Put a Message to a Queue.
```
os_InRegs osEvent osMessageGet(osMessageQId queue_id, uint32_t millisec)
```
Get a Message or Wait for a Message from a Queue.

## 9. Mail Queue Management

**Macros**
```
#define osMailQDef(name, queue_sz, type)
```
Create a Mail Queue Definition.
```
#define osMailQ(name) &os_mailQ_def_##name
```
Access a Mail Queue Definition.

**Functions**
```
osMailQId osMailCreate(const osMailQDef_t *queue_def, osThreadId thread_id)
```
Create and Initialize mail queue.
```
void *osMailAlloc(osMailQId queue_id, uint32_t millisec)
```
Allocate a memory block from a mail.
```
void *osMailCAlloc(osMailQId queue_id, uint32_t millisec)
```
Allocate a memory block from a mail and set memory block to zero.
```
osStatus osMailPut(osMailQId queue_id, void *mail)
```
Put a mail to a queue.
```
os_InRegs osEvent osMailGet(osMailQId queue_id, uint32_t millisec)
```
Get a mail from a queue.
```
osStatus osMailFree(osMailQId queue_id, void *mail)
```
Free a memory block from a mail.

## 10. Generic Wait Function

```
osStatus osDelay(uint32_t millisec)
```
Wait for Timeout (Time Delay).

## 11. RTX Global Functions

```
void os_idle_demon(void)
```
The idle demon is running when no other thread is ready to run.
```
void os_error(uint32_t error_code)
```
Called when a runtime error is detected.

## 12. Status and Error Codes

**Enumerations**
```
Enum osStatus {
  osOK = 0,
  osEventSignal = 0x08,
  osEventMessage = 0x10,
  osEventMail = 0x20,
  osEventTimeout = 0x40,
  osErrorParameter = 0x80,
  osErrorResource = 0x81,
  osErrorTimeoutResource = 0xC1,
  osErrorISR = 0x82,
  osErrorISRRecursive = 0x83,
  osErrorPriority = 0x84,
  osErrorNoMemory = 0x85,
  osErrorValue = 0x86,
  osErrorOS = 0xFF,
  os_status_reserved = 0x7FFFFFFF
}
```

## 13. Constants

**Macro Definitions**
```
#define osWaitForever 0xFFFFFFFFU
```
Wait forever timeout value

## 14. Other Definitions

```
typedef struct {
  osStatus status;              // status code: event or error information
  union {
    uint32_t v;                 // message as 32-bit value
    void *p;                    // message or mail as void pointer
    int32_t signals;            // signal flags
  } value;                      // event value
  union {
    osMailQId mail_id;          // mail id obtained by \ref osMailCreate
    osMessageQId message_id;    // message id obtained by \ref osMessageCreate
  } def;                        // event definition
} osEvent;
```