ASR1802 OS API(OSA) User Manual V0.2

2017 年 10 月

ASR team

Directory

About This Document

Purpose

This document describes OS API of ASR1802 - OSA (OS Adaptor).

Product/Sub-Product Overview

Document Structure

Referenced Documents

Table 1: Related Documentation

Ref #	Document Name	Doc Number	Revision
[1]	osa_old_api.h		
[2]	osa_api.h		
[3]	PXA1802 OS API(OSA) User Manual v0.1.pdf		

API Reference

Initialization

OSAInitialize

Prototype:	*OSA_STATUS OSAInitialize(void);*	
Description:	This function initializes the OS software interface.	
Input Parameters:	None	
Output Parameters:	None	
Returns:	OS_SUCCESS	The initialization has completed without error
	OS_FAIL	The initialization did not complete successfully.
Attributes	Synchronous, Non-blocking, no scheduler involvement	
Notes:	After this function returns, tasks and other OS objects may be created. On return from this function, the scheduler is disabled. The scheduler is enabled by calling **OSARun()**.	

Example:

```
OSA_STATUS status;


/*  Initialize the OS Abstraction layer. */


status = OSAInitialize();
```

OSARun()

Prototype: *Void OSARun(void);*

Description: This function completes OS initialization, sets the OS system tick counter to zero, and enables the scheduler.

Input Parameters: None.

Output Parameters:

Returns:

Attributes Synchronous, Non-blocking, involves scheduler

Notes:

Example:

```
/*  Start-up the scheduler. */


status = OSARun();
```

Task Management

OSAContextLock

Prototype: *OSA_STATUS OSAContextLock(void);*

Description: This function disables context switching until the lock is removed.

Input Parameters: None.

Output Parameters: None.

Returns: OS_SUCCESS

Attributes Synchronous, Non-blocking, disables scheduler

Notes: WinCE : Implemented using a critical section object so does not provide the exact functionality desired.

Symbian: Implemented using a semaphore object.

Linux: Not supported and has no affect when used

Example:

```
OSA_STATUS status;


status = OSAContextLock();
```

OSAContextUnlock

Prototype: *OSA_STATUS OSAContextUnlock(void);*

Description: This function enables context switching. If there is a ready task with a priority greater than the priority of the calling task, the task with the higher priority will preempt the calling task.

Input Parameters: None.

Output Parameters: None.

Returns: OS_SUCCESS

Attributes Synchronous, Non-blocking, may involve scheduler

Notes: WinCE: Implemented using a critical section object so does not provide the exact functionality desired.

Symbian: Implemented using a semaphore object.

Linux: Not supported and has no affect when used

Example:

```
OSA_STATUS status;


status = OSAContextUnlock();
```

OSASysContextLock

Prototype:	*OSA_STATUS OSASysContextLock(*`OSASysContext*`*);*
Description:	This function locks context (both task preemption and isr disable). It allows nesting calls (lock, lock - unlock, unlock)and other OS function calls
Input Parameters:	None.
Output Parameters:	PrevContext – where to save previous context as it was before the lock.
Returns:	OS_SUCCESS
Attributes	Synchronous, Non-blocking, disables scheduler, disables interrupts
Notes:	WinCE : Not supported and has no affect when used Symbian: Not supported and has no affect when used Linux: Not supported and has no affect when used
Example:	

```
OSA_STATUS status;


status = OSASysContextLock();
```

OSASysContextUnlock

Prototype: *OSA_STATUS OSAContextUnlock(void);*

Description: This function unconditionally unlocks context switching and enables interrupts. If there is a ready task with a priority higher than the priority of the calling task, the task with the higher priority will preempt the calling task.

Input Parameters: None.

Output Parameters: None.

Returns: OS_SUCCESS

Attributes Synchronous, Non-blocking, may involve scheduler

Notes: WinCE: Not supported and has no affect when used.

Symbian: Not supported and has no affect when used.

Linux: Not supported and has no affect when used

Example:

```
OSA_STATUS status;


status = OSASysContextUnlock();
```

OSASysContextRestore

Prototype:	*OSA_STATUS OSASysContexRestore(*`OSASysContext*`*);*
Description:	This function restores context (both task preemption and isr disable). It allows nesting calls (lock, lock - unlock, unlock)
Input Parameters:	None.
Output Parameters:	PrevContext – previous context .
Returns:	OS_SUCCESS
Attributes	Synchronous, Non-blocking, enables scheduler, enables interrupts
Notes:	WinCE : Not supported and has no affect when used Symbian: Not supported and has no affect when used Linux: Not supported and has no affect when used
Example:	

```
OSA_STATUS status;


status = OSASysContextLock();
```

OSATaskCreate

Prototype: *OSA_STATUS OSATaskCreate(OSATaskRef **taskRef,** **void** stackPtr, UINT32 stackSize, UINT8 priority, CHAR *taskName, void (*taskStart)(void*), void *argv);*

Description: This function requests that a task be created with the specified parameters.

Input Parameters:

[1] *void* stackPtr*
pointer to the low address of the stack

[2] *UINT32 stackSize*
maximum size of the stack

[3] *UINT8 priority*
initial priority of the task. Range 0…31 where 0 is the highest priority and 31 is the lowest priority. If OSA_NO_PRIORITY_CONVERSION is enabled, priority range is defaulted to the old 0 … 255 range.

[4] *CHAR* taskName*
Pointer to an 8 character name for the task. The name does not have to be null-terminated.

[5] *void* taskStart*
entry function of the task

[6] *void *argv*
argument to be passed into task entry function

Output Parameters:

[7] *OSATaskRef* taskRef*
OS assigned reference to the task

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Task reference is NULL.
OS_INVALID_PTR	Task's entry function pointer is NULL.
OS_INVALID_MEMORY	Pointer to stack memory is NULL.
OS_INVALID_SIZE	Stack size is insufficient.
OS_INVALID_PRIORITY	Priority is invalid.
OS_NO_TASKS	No available task references.
OS_FAIL	OS specific error.

Attributes Synchronous, Non-blocking, may involve scheduler

Notes: Symbian: Tasks are created with initial priority of EPriorityNormal as flat priorities are not supported in Symbian.

Example:

```
/*  Assume ''taskRef'' is defined as a global. This is one of
    several ways to allocate a reference. */


OSATaskRef taskRef;

OSA_STATUS status;


/*  Create a task whose entry point is the function
    ''task_entry'' and that has a 2000-byte stack pointed to
    by ''stack_ptr'', a priority of 20 and no argument. */


status = OSATaskCreate(&taskRef, stack_ptr, 2000, 20,
    ''TASK_1'', task_entry, NULL);
```

OSATaskDelete

Prototype: *OSA_STATUS OSATaskDelete(OSATaskRef taskRef);*

Description: This function requests that the specified task be deleted.

Input Parameters: [8] *OSATaskRef taskRef*
OS assigned reference to the task

Output Parameters: [9] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid task reference.
OS_FAIL	Task could not be deleted because it was not in the **Suspended** state or due to other OS specific error.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes: All resources held by the task should be released before this function is called. A task must be in the **Suspended** state or this call will fail.

Example:

```
OSATaskRef taskRef;

OSA_STATUS status;


/*  Delete the task ''taskRef''. Assume ''taskRef'' has
    previously been created with the OSATaskCreate service
    call. */


status = OSATaskDelete(taskRef);
```

OSATaskChangePriority

Prototype: *OSA_STATUS OSATaskChangePriority(OSATaskRef taskRef, UINT8 newPriority, UINT8 *oldPriority);*

Description: This function changes the priority of the specified task.

Input Parameters:

[10] *OSATaskRef taskRef*
OS assigned reference to the task

[11] *UINT8 newPriority*
Specifies a priority value between 0 and 31. The lower the numeric value, the higher the task's priority. If OSA_NO_PRIORITY_CONVERSION is enabled, priority range is defaulted to the old 0 … 255 range.

Output Parameters:

[12] *UINT8 *oldPriority*
Previous priority of the task returned by the service.

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Task reference is NULL.
OS_INVALID_PRIORITY	Priority is invalid.

Attributes Synchronous, Non-blocking, may involve scheduler

Notes: Symbian: Priority of EPriorityNormal is fixed as flat priorities are not supported in Symbian. Must use Symbian specific code to tune task priorities.

Example:

```
OSATaskRef taskRef;
```

```
OSA_STATUS status;

UINT8     old_priority;


/*  Change the priority of the task ‘‘taskRef’’ to the
    priority 10. Assume ‘‘taskRef’’ has previously been
    created with the OSATaskCreate service call. */


status = OSATaskChangePriority(taskRef, 10,
    &old_priority);
```

OSATaskGetPriority

Prototype: *OSA_STATUS OSATaskGetPriority(OSATaskRef taskRef, UINT8 *priority);*

Description: This function retrieves the priority of the specified task.

Input Parameters: [13] *OSATaskRef taskRef*
OS assigned reference to the task

Output Parameters: [14] *UINT8 *priority*
Priority of the task returned by the service.

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Task reference is NULL.

Attributes Synchronous, Non-blocking, may involve scheduler

Notes: Returns a priority value between 0 and 31. The lower the numeric value, the higher the task’s priority. If OSA_NO_PRIORITY_CONVERSION is defined the priority range defaults back to the old 1 … 255.

RTT Support: RTT only supports priorities from 0 – 30.

Example:

```
OSATaskRef taskRef;

OSA_STATUS status;

UINT8     priority;
```

```
/*  Get the priority of the task ‘‘taskRef’’. Assume
    ‘‘taskRef’’ has previously been created with the
    OSATaskCreate service call. */


status = OSATaskGetPriority(taskRef, &priority);
```

OSATaskResume

Prototype: *OSA_STATUS OSATaskResume(OSATaskRef taskRef);*

Description: This function requests that a specified task be resumed.

Input Parameters: [15] *OSATaskRef taskRef*
OS assigned reference to the task

Output Parameters: [16] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Task reference is NULL.
OS_FAIL	Indicates that the task was not in Suspended state or other OS specific error occurred.

Attributes Synchronous, Non-blocking, may involve scheduler

Notes:

Example:

```
OSATaskRef taskRef;

OSA_STATUS status;


/*  Resume the task ‘‘taskRef’’. Assume ‘‘taskRef’’ has
    previously been created with the OSATaskCreate service
    call. */


status = OSATaskResume(taskRef);
```

OSATaskSleep

Prototype: *void OSATaskSleep(UINT32 ticks);*

Description: This function requests that the task be put to sleep for a number of OS clock TICKS. Use OSA_TICK_FREQ_IN_MILLISEC to get the configured tick time for the system.

Input Parameters: [17] UINT32 ticks
number of OS clock ticks to sleep

Output Parameters: [18] None

Returns: None

Attributes Asynchronous, Blocking, involves scheduler

Notes:

Example:

```
/*  Sleep for 200 timer ticks. */


OSATaskSleep(200);
```

OSATaskSuspend

Prototype: *OSA_STATUS OSATaskSuspend(OSATaskRef taskRef);*

Description: This function requests that a specific task be suspended including the current task.

Input Parameters: [19] *OSATaskRef taskRef*
OS assigned reference to the task

Output Parameters: [20] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Task reference is NULL.
OS_FAIL	OS specific error.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes: RTT Support: RTT only supports suspending the current task and not other tasks.

Example:

```
OSATaskRef taskRef;

OSA_STATUS status;


/* Unconditionally suspend the task ‘‘taskRef’’. Assume
   ‘‘taskRef’’ has previously been created with the
   OSATaskCreate service call. */


status = OSATaskSuspend(taskRef);
```

OSATaskGetCurrentRef

Prototype: *OSA_STATUS OSATaskGetCurrentRef(OSATaskRef *taskRef);*

Description: This function retrieves the reference for the current running task.

Input Parameters: [21] None

Output Parameters: [22] *OSATaskRef** *taskRef*
OS assigned reference to the task

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Task reference is NULL.
OS_FAIL	OS specific error.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes:

Example:

```
OSATaskRef taskRef;

OSA_STATUS status;


/* Get the reference for the currently running task. */


status = OSATaskGetCurrentRef(&taskRef);
```

OSATaskYield

Prototype: *OSA_STATUS OSATaskGetCurrentRef(OSATaskRef *taskRef);*

Description: This function retrieves the reference for the current running task.

Input Parameters: [23] None

Output Parameters: [24] *OSATaskRef** *taskRef*
OS assigned reference to the task

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Task reference is NULL.
OS_FAIL	OS specific error.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes:

Example:

```
OSATaskRef taskRef;

OSA_STATUS status;


/* Get the reference for the currently running task. */


status = OSATaskGetCurrentRef(&taskRef);
```

Event Flags

OSAFlagCreate

Prototype: *OSA_STATUS OSAFlagCreate(OSAFlagRef *flagRef);*

Description: This function requests that a flag group be created.

Input Parameters: [25] None

Output Parameters: [26] *OSAFlagRef *flagRef*
OS assigned reference to the flag

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Pointer to "flagRef " is NULL.
OS_NO_FLAGS	No available flags left in the system.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes:

Example:

```
OSAFlagRef flagRef;

OSA_STATUS status;


/*  Assume ''flagRef'' is defined as a global. This is one
    of several ways to allocate a reference. */


status = OSAFlagCreate(&flagRef);
```

OSAFlagCreateGlobal

Prototype: *OSA_STATUS* OSAFlagCreateGlobal *(OSAFlagRef *flagRef);*

Description: This function requests that a global event flag group be created. If the event flag is already created the existing reference is returned allowing tasks in multiple processes to share the same resource (Symbian only).

Input Parameters: [27] None

Output Parameters: [28] *OSAFlagRef *flagRef*
OS assigned reference to the flag

*[29] char *flagName*

[30] name of flag

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Pointer to "flagRef " is NULL.
OS_NO_FLAGS	No available flags left in the system.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes:

Example:

```
OSAFlagRef flagRef;

OSA_STATUS status;

char flag [] = ''flagName'';



status =
   OSAFlagCreateGlobal(&flagRef,&flagName);
```

OSAFlagDelete

Prototype: *OSA_STATUS OSAFlagDelete(OSAFlagRef flagRef);*

Description: This function requests that the specified flag be deleted.

Input Parameters: [31] *OSAFlagRef flagRef*
OS assigned reference to the flag

Output Parameters: [32] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid flag reference.
OS_FAIL	OS specific error.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes: If there are tasks waiting for a **OSAFlagWait()** operation to complete, the waiting tasks will be made ready to run. See **OSAFlagWait()** for details.

RTT Support: OS_FAIL will be returned when trying to delete a flag with users suspended waiting for it.

Example:

```
OSAFlagRef flagRef;

OSA_STATUS status;


/*  Delete the flag ‘‘flagRef’’. Assume ‘‘flagRef’’ has
    previously been created with the OSAFlagCreate service
    call. */


status = OSAFlagDelete(flagRef);
```

OSAFlagPeek

Prototype: *OSA_STATUS OSAFlagPeek(OSAFlagRef flagRef, UINT32* flags);*

Description: This function checks the value of a flag.

Input Parameters: [33] *OSAFlagRef flagRef*
OS assigned reference to the flag

Output Parameters: [34] *UINT32 flags*
current value of event flags referenced by "flagRef"

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Flag reference is NULL.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes:

Example:

```
OSAFlagRef flagRef;

OSA_STATUS status;

UINT32     flags;


/* Check the value of the flag ''flagRef''. Assume
   ''flagRef'' has previously been created with the
   OSAFlagCreate service call. */


status = OSAFlagPeek(flagRef, &flags);
```

OSAFlagSet

Prototype: *OSA_STATUS OSAFlagSet(OSAFlagRef flagRef, UINT32 mask, UINT32 operation);*

Description: This function executes a logical OR or AND of the flag group with the input mask.

Input Parameters: [35] *OSAFlagRef flagRef*
OS assigned reference to the flag.

[36] *UINT32 mask*

mask specifying which bits need to be set. A 1 in a certain bit position will set the same bit position in the flag.

[37] *UINT32 operation*

logical operation to be executed on the flag group. OSA_AND executes a logical AND and OSA_OR executes a logical OR.

Output Parameters: None.

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Flag reference is NULL.
OS_INVALID_MODE	Operation is invalid.

Attributes Synchronous, Non-blocking, may involve scheduler

Notes:

Example:

```
OSAFlagRef flagRef;

OSA_STATUS status;


/*  Set the event flags 7, 4, and 2 in the flag ‘‘flagRef’’.
    Assume ‘‘flagRef’’ has previously been created with the
    OSAFlagCreate service call. */


status = OSAFlagSet(flagRef, 0x00000094, OSA_OR);
```

OSAFlagWait

Prototype: *OSA_STATUS OSAFlagWait(OSAFlagRef flagRef, UINT32 mask, UINT32 operation, UINT32* flags, UINT32 timeout);*

Description: This function waits for the specified operation on a flag group to complete. The operation is defined by the "operation" input parameter. If the timeout input parameter is **OSA_NO_SUSPEND** the operation completes immediately and the current value of the flags is returned in the output parameter "flags". By specifying **OSA_NO_SUSPEND**, an application can read the current value of the flags without blocking.

Input Parameters:

[38] *OSAFlagRef flagRef*
OS assigned reference to the flag

[39] *UINT32 mask*
mask of flags to wait for

[40] *UINT32 operation*
may be one of the following:

- **OSA_FLAG_AND** – Wait for all of the bits in the input mask to be set, don't clear the event flags
- **OSA_FLAG_AND_CLEAR** – Wait for all of the bits in the input mask to be set, clear all event flags on successful completion
- **OSA_FLAG_OR** – Wait for any of the bits in the input mask to be set, don't clear the event flags
- **OSA_FLAG_OR_CLEAR** – Wait for any of the bits in the input mask to be set, clear all event flags of successful completion

[41] *UINT32 timeout*
If timeout is set to **OSA_NO_SUSPEND**, the operation completes immediately and the current value of the flags is returned in the output parameter "flags". If timeout is set to **OSA_SUSPEND**, this call will block until the condition specified by the "mask" and "operation" inputs is satisfied. If a timeout value between 1 and 4,294,967,293 is specified, the call will block until the wait condition is satisfied or until the timeout period, in number of OS clock ticks, elapses.

Output

[42] *UINT32* flags*

Parameters: the current value of all flags

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Flag reference is NULL.
OS_INVALID_MODE	Invalid operation.
OS_INVALID_POINTER	Pointer to 'flags' is NULL.
OS_TIMEOUT	A timeout has occurred while suspended waiting for a Wait operation to complete.
OS_FLAG_NOT_PRESENT	Flag combination not met when using **OSA_NO_SUSPEND**.

Attributes Synchronous, Blocking only when using a timeout or **OSA_NO_SUSPEND**, may involve scheduler

Notes: Only one task may wait for the same flag reference at the same time.

Example:

```
OSAFlagRef flagRef;

OSA_STATUS status;

UINT32     flags;


/* Retrieve event flags 7, 4, and 1 from the flag ''flagRef''.
   All event flags must be present to satisfy the request
   as the operation specified is OSA_FLAG_AND_CLEAR. If they
   are not, the calling task suspends unconditionally. Also,
   event flags 7, 4, and 1 are consumed when this request
   is satisfied. Assume ''flagRef'' has previously been
   created with the OSAFlagCreate service call. */


status = OSAFlagWait(flagRef, 0x0000094,
   OSA_FLAG_AND_CLEAR, &flags, OSA_SUSPEND);
```

Semaphores

OSASemaphoreAcquire

Prototype: *OSA_STATUS OSASemaphoreAcquire(OSASemaRef semaRef, UINT32 timeout);*

Description: This function requests that the specified semaphore be decremented. If the semaphore count is zero before this call, the service cannot be satisfied immediately. In that case, the blocking behavior is specified by the "timeout" parameter.

Input Parameters:

[43] *OSASemaRef semaRef*

OS assigned reference to the semaphore

[44] *UINT32 timeout*

If timeout is set to **OSA_NO_SUSPEND**, this call will not block. If timeout is set to **OSA_SUSPEND**, this call will block until the semaphore count is greater than zero. If a timeout value between 1 and 4,294,967,293 is specified, the call will block until the semaphore count is greater than zero or the timeout period, in number of OS clock ticks, elapses.

Output Parameters: [45] None

Returns:

OS_SUCCESS	Semaphore has been decremented.
OS_INVALID_REF	Invalid semaphore reference.
OS_UNAVAILABLE	The semaphore is not available.
OS_TIMEOUT	A timeout has occurred while waiting for the semaphore count to be greater than zero.
OS_FAIL	OS specific error.

Attributes Synchronous, Blocking only when using a timeout or **OSA_NO_SUSPEND**, may involve scheduler

Notes:

Example:

```
OSASemaRef semaRef;

OSA_STATUS status;
```

```
/* Acquire the semaphore ''semaRef''. If the semaphore is
   unavailable, suspend for a maximum of 200 timer ticks.
   The order of multiple tasks suspending on the same
   semaphore is determined when the semaphore is created.
   Assume ''semaRef'' has previously been created with the
   OSASemaphoreCreate service call. */


status = OSASemaphoreAcquire(semaRef, 200);
```

OSASemaphoreCreate

Prototype: *OSA_STATUS OSASemaphoreCreate(OSASemaRef *semaRef, UINT32 initialCount, UINT32 waitingMode);*

Description: This function requests that a counting semaphore be created.

Input Parameters:

[46] *UINT32 initialCount*

Initial count of the semaphore.

[47] *UINT32 waitingMode*

OSA_FIFO or OSA_PRIORITY. "waitingMode" specifies how tasks are added to a semaphore's Wait queue. They may be added in first-in-first-out order (OSA_FIFO) or in priority order (OSA_PRIORITY) with the highest priority waiting task at the front on the queue.

Output Parameters:

[48] *OSASemaRef *semaRef*

OS assigned reference to the semaphore

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid semaphore reference.
OS_INVALID_MODE	Invalid mode.
OS_NO_SEMAPHORES	No available semaphores

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes: The waiting mode of a semaphore specifies how tasks are added to the Semaphore Wait queue. They can be added in first-in-first-out order or in priority order with the highest priority waiting task at the front on the queue.

WinCE Support: Waiting mode is fixed at priority waiting so ‘waitingMode’ is ignored.

Example:

```
/*  Assume semaphore reference is defined as a global. This
    is one of several ways to allocate a reference. */


OSASemaRef semaRef;

OSA_STATUS status;


/*  Create a semaphore with an initial count of 2 and a FIFO
    order task suspension. */


status = OSASemaphoreCreate(&semaRef, 2, OSA_FIFO);
```

OSASemaphoreCreateGlobal

Prototype: *OSA_STATUS OSASemaphoreCreate(OSASemaRef *semaRef, UINT32 initialCount, UINT32 waitingMode);*

Description: This function requests that a counting semaphore be created. If the semaphore is already created the existing reference is returned allowing tasks in multiple processes to share the same resource (Symbian only).

Input Parameters:

[49] *UINT32 initialCount*

Initial count of the semaphore.

[50] *UINT32 waitingMode*

OSA_FIFO or OSA_PRIORITY. "waitingMode" specifies how tasks are added to a semaphore's Wait queue. They may be added in first-in-first-out order (OSA_FIFO) or in priority order (OSA_PRIORITY) with the highest priority waiting task at the front on the queue.

Output Parameters:

[51] *OSASemaRef *semaRef*

OS assigned reference to the semaphore

`char*semaName`

name of semaphore

[52]

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid semaphore reference.
OS_INVALID_MODE	Invalid mode.
OS_NO_SEMAPHORES	No available semaphores

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes: The waiting mode of a semaphore specifies how tasks are added to the Semaphore Wait queue. They can be added in first-in-first-out order or in priority order with the highest priority waiting task at the front on the queue.

WinCE Support: Waiting mode is fixed at priority waiting so 'waitingMode' is ignored.

Example:

```
/*  Assume semaphore reference is defined as a global. This
```

```
    is one of several ways to allocate a reference. */


OSASemaRef semaRef;

OSA_STATUS status;

char semaName[] = ''semaphoreName'';


/*  Create a semaphore with an initial count of 2 and a FIFO
    order task suspension. */


status = OSASemaphoreCreateGlobal(&semaRef, #semaName, 2,
    OSA_FIFO);
```

OSASemaphoreDelete

Prototype: *OSA_STATUS OSASemaphoreDelete(OSASemaRef semaRef,);*

Description: This function requests that the specified semaphore be deleted.

Input Parameters: [53] *OSASemaRef semaRef*
OS assigned reference to the semaphore

Output Parameters: [54] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid semaphore reference.
OS_FAIL	OS specific error.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes:

Example:

```
OSASemaRef semaRef;

OSA_STATUS status;


/*  Delete the semaphore reference ''semaRef''. Assume
    ''semaRef'' has previously been created with the
    OSASemaphoreCreate service call. */
```

```
status = OSASemaphoreDelete(semaRef);
```

OSASemaphoreRelease

Prototype: *OSA_STATUS OSASemaphoreRelease(OSASemaRef semaRef);*

Description: If there are any tasks waiting for the semaphore, the first waiting task is made ready to run. If there are no tasks waiting, the value of the semaphore is incremented by one.

Input Parameters: [55] *OSASemaRef semaRef*
OS assigned reference to the semaphore

Output Parameters: [56] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid semaphore reference.

Attributes Synchronous, Non-blocking, may involve scheduler

Notes:

Example:

```
OSASemaRef semaRef;

OSA_STATUS status;


/* Release the semaphore ''semaRef''. If other tasks are
   waiting to obtain the same semaphore, this service
   results in a transfer of this instance of the semaphore
   to the first task waiting. Assume ''semaRef'' has
   previously been created with the OSASemaphoreCreate
   service call. */


status = OSASemaphoreRelease(semaRef);
```

OSASemaphorePoll

Prototype: *OSA_STATUS OSASemaphorePoll(OSASemaRef semaRef, UINT32 *count);*

Description: This function requests the current semaphore count.

Input Parameters: [57] *OSASemaRef semaRef*
OS assigned reference to the semaphore

Output Parameters: [58] *UINT32 *count*
Current value of the semaphore.

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid semaphore reference.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes:

Example:

```
OSASemaRef semaRef;

OSA_STATUS status;

UINT32     currentCount;


/* Retrieve the current count of the semaphore ''semaRef''.
   Assume ''semaRef'' has previously been created with the
   OSASemaphoreCreate service call. */


status = OSASemaphorePoll(semaRef, &currentCount);
```

Mutexes

OSAMutexCreate

Prototype: *OSA_STATUS OSAMutexCreate(OSAMutexRef *mutexRef, UINT32 waitingMode);*

Description: This function requests that a mutex be created. Mutexes use the priority inheritance protocol to bound the time spent in priority inversions.

Input Parameters: [59] *UINT32 waitingMode*
OSA_FIFO or OSA_PRIORITY.

Output Parameters: [60] *OSSMutexRef *mutexRef*
OS assigned reference to the semaphore

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid mutex reference.
OS_INVALID_MODE	Invalid mode.
OS_NO_MUTEXES	No available mutexes in the system.
OS_FAIL	Mutex already locked by the task.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes: The waiting mode of a mutex specifies how tasks are added to the Mutex Wait queue. They may be added in first-in-first-out order (OSA_FIFO) or in priority order (OSA_PRIORITY) with the highest priority waiting task at the front on the queue.

Mutexes use the priority inheritance protocol to ensure that time spent in priority inversions is bounded. The priority inheritance protocol ensures that the priority of the mutex owner is always greater than or equal to that of the highest priority task waiting for the mutex. When a task releases ownership of a mutex, its priority is restored to the original value (i.e. the value it had before the mutex was acquired).

The priority inheritance protocol, unlike the priority ceiling protocol, does not prevent deadlocks if a task uses nested locks. (i.e. a task locks “Mutex 1” and then locks “Mutex 2” before unlocking “Mutex 1”)

If a task tries to lock the same mutex twice, **OSAMutexLock()** will

return **OS_FAIL**.

WinCE Support: Waiting mode is fixed at priority waiting so ‘waitingMode’ is ignored. OS_FAIL will not be returned when locking the same mutex twice.

Example:

```
/*  Assume the mutex reference is defined as a global. This
    is one of several ways to allocate a reference. */


OSAMutexRef mutexRef;

OSA_STATUS  status;


/*  Create a mutex with priority order task suspension. */


status = OSAMutexCreate(&mutexRef, OSA_PRIORITY);
```

OSAMutexDelete

Prototype: *OSA_STATUS OSAMutexDelete(OSAMutexRef mutexRef,);*

Description: This function requests that the specified mutex be deleted.

Input Parameters: [61] *OSAMutexRef mutexRef*
OS assigned reference to the mutex

Output Parameters: [62] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid mutex reference.
OS_FAIL	OS specific error.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes: If there are tasks waiting for an **OSAMutexLock()** operation to complete, the waiting tasks will be made ready to run. See **OSAMutexLock()** for details.

Example:

```
OSAMutexRef mutexRef;
```

```
OSA_STATUS  status;


/*  Delete the mutex reference ‘‘mutexRef’’. Assume
    ‘‘mutexRef’’ has previously been created with the
    OSAMutexCreate service call. */


status = OSAMutexDelete(MutexRef);
```

OSAMutexLock

Prototype: *OSA_STATUS OSAMutexLock (OSAMutexRef mutexRef, UINT32 timeout);*

Description: This function requests that the specified mutex be locked. If the mutex is locked by another task before this call, the service cannot be satisfied immediately. In this case, the blocking behavior is specified by the "timeout" parameter.

Input Parameters:

[63] *OSAMutexRef mutexRef*

OS assigned reference to the mutex

[64] *UINT32 timeout*

If timeout is set to **OSA_NO_SUSPEND**, this call will not block. If timeout is set to **OSA_SUSPEND**, this call will block until the semaphore count is greater than zero. If a timeout value between 1 and 4,294,967,293 is specified, the call will block until the mutex is unlocked or the timeout period, in number of OS clock ticks, elapses.

Output Parameters: [65] None

Returns:

OS_SUCCESS	Mutex has been acquired.
OS_INVALID_REF	Invalid mutex reference.
OS_UNAVAIABLE	The mutex is not available. It is locked by another task.
OS_TIMEOUT	A timeout has occurred while waiting for the mutex.
OS_FAIL	The calling task already has locked the mutex.

Attributes Synchronous, Blocking only when using a timeout or **OSA_SUSPEND**, may involve scheduler

Notes: See notes for **OSAMutexCreate()**

Example:

```
OSAMutexRef mutexRef;

OSA_STATUS  status;
```

```
/*  Lock the mutex ‘‘mutexRef’’. If the mutex is unavailable,
    suspend for a maximum of 200 timer ticks. The order of
    multiple tasks suspending on the same mutex is determined
    when the mutex is created. Assume ‘‘mutexRef’’ has
    previously been created with the OSAMutexCreate service
    call. */


status = OSAMutexLock(mutexRef, 200);
```

OSAMutexUnlock

Prototype: *OSA_STATUS OSAMutexUnlock (OSAMutexRef mutexRef);*

Description: If there are any tasks waiting for the mutex, the task at the front of the Wait queue is made ready to run. Only the mutex owner may unlock a mutex.

Input Parameters: [66] *OSAMutexRef mutexRef*
OS assigned reference to the mutex

Output Parameters: [67] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid mutex reference.
OS_FAIL	The calling task is not the mutex owner.

Attributes Synchronous, Non-blocking, may involve scheduler

Notes: See notes for **OSAMutexCreate().**

Example:

```
OSAMutexRef mutexRef;

OSA_STATUS  status;


/*  Unlock the mutex ‘‘mutexRef’’. If other tasks are waiting
    to obtain the same mutex, this service results in a
    transfer of this instance of the mutex to the first task
    waiting. Assume ‘‘mutexRef’’ has previously been created
```

```
                with the OSAMutexCreate service call. */


        status = OSAMutexUnlock(mutexRef);
```

Message Queues

OSAMsgQCreate

Prototype: *OSA_STATUS OSAMsgQCreate(OSAMsgQRef *msgQRef, char *queueName, UINT32 maxSize, UINT32 maxNumber, UINT32 waitingMode);*

Description: This function requests that a message queue be created. All memory used to store messages on the message queue is allocated by the operating system.

Input Parameters:

*[68] char *queueName*

[69] 8 character name of queue. . The name does not have to be null-terminated.

[70] *UINT32 maxSize*
maximum size of a message on the queue. This is used for error checking by **OSAMsgQSend()**.

[71] *UINT32 maxNumber*
maximum number of messages on the queue

[72] *UINT32 waitingMode*
defines scheduling of waiting events: OSA_FIFO, or OSA_PRIORITY.

Output Parameters:

[73] *OSAMsgQRef *msgQRef*
pointer to location to hold message queue reference allocated by the operating system

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid queue reference.
OS_INVALID_MODE	Invalid waiting mode.
OS_INVALID_SIZE	Invalid queue size.
OS_NO_QUEUES	No available queues left in the system.
OS_FAIL	OS specific error.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes: Message queues are implemented using variable messages. Space in a queue versus number of messages is used to

determine whether the call will block or send “queue full/empty”. The waiting mode can be either FIFO or priority wise (according to the priority of the task waiting)

Nucleus Support: Nucleus sends and receives data in word (4 byte) boundaries. Data must be sent and received using long word boundaries. Memory for the queue is configured in the osa_config.h or gbl_config.h files.

RTT Support: RTT only supports full blocking queues and does not support timeouts. Memory for the queue comes from the system and not OSA.

WinCE Support: Waiting mode is fixed at priority waiting so ‘waitingMode’ is ignored. Memory for the queue comes from the process heap and not OSA.

Example:

```
OSAMsgQRef msgQRef;

OSA_STATUS status;


/*  Assume message queue ‘‘msgQRef’’ is defined as a global.
    This is one of several ways to allocate a reference.
    Create a queue with a capacity of 200 elements that can
    each have a maximum size of 32 bytes. */


status = OSAMsgQCreate(&msgQRef, ‘‘testQ’’, 32, 200,
    OSA_PRIORITY);
```

OSAMsgQCreateWithMem

Prototype: *OSA_STATUS OSAMsgQCreateWithMem(OSAMsgQRef *msgQRef, char *queueName, UINT32 maxSize, UINT32 maxNumber, void *qAddr, UINT32 waitingMode);*

Description: This function requests that a message queue be created. Memory used to store messages on the message queue is provided by the user.

Input Parameters:

*[74] char *queueName*

[75] 8 character name of queue. . The name does not have to be null-terminated.

[76] *UINT32 maxSize*

maximum size of a message on the queue. This is used for error checking by **OSAMsgQSend()**.

[77] *UINT32 maxNumber*

maximum number of messages on the queue

[78] Void *qAddr

pointer to memory to be used for the queue

[79] *UINT32 waitingMode*

defines scheduling of waiting events: OSA_FIFO, or OSA_PRIORITY.

Output Parameters:

[80] *OSAMsgQRef *msgQRef*

pointer to location to hold message queue reference allocated by the operating system

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid queue reference.
OS_INVALID_MODE	Invalid waiting mode.
OS_INVALID_SIZE	Invalid queue size.
OS_NO_QUEUES	No available queues left in the system.
OS_FAIL	OS specific error.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes: Message queues are implemented using variable messages. Space in a queue versus number of messages is used to

determine whether the call will block or send “queue full/empty”. The waiting mode can be either FIFO or priority wise (according to the priority of the task waiting)

Nucleus Support: Nucleus sends and receives data in word (4 byte) boundaries. Data must be sent and received using long word boundaries.

RTT Support: RTT only supports blocking queues and does not support timeouts.

WinCE Support: Waiting mode is fixed at priority waiting so ‘waitingMode’ is ignored. Memory for the queue comes from the process heap and not OSA.

Example:

```
OSAMsgQRef msgQRef;

OSA_STATUS status;

UINT8      queue[32 * 200];


/*  Assume message queue ‘‘msgQRef’’ and buffer ‘‘queue’’ are
    defined as a global. This is one of several ways to
    allocate a reference and space for the queue. Create a
    queue with a capacity of 200 elements that can each have
    a maximum size of 32 bytes. */


  status = OSAMsgQCreateWithMem(&msgQRef, ‘‘testQ’’, 32, 200,
      (void*)queue, OSA_PRIORITY);
```

OSAMsgQDelete

Prototype: *OSA_STATUS OSAMsgQDelete(OSAMsgQRef msgQRef);*

Description: This function requests that the specified message queue be deleted.

Input Parameters: [81] *OSAMsgQRef msgQRef*
identifier that uniquely identifies the message queue

Output Parameters: [82] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid queue reference.
OS_FAIL	OS specific error.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes:

Example:

```
OSAMsgQRef msgQRef;

OSA_STATUS status;


/* Delete the queue ‘‘msgQRef’’. Assume ‘‘msgQRef’’ has
   previously been created with the OSAMsgQCreate service
   call. */


status = OSAMsgQDelete(msgQRef);
```

OSAMsg*Q*Poll

Prototype: *OSA_STATUS OSAMsgQPoll(OSAMsgQRef msgQRef, UINT32* msgCount)*

Description: This function checks the number of messages on the message queue.

Input Parameters: [83] *OSAMsgQRef msgQRef*
identifier that uniquely identifies the message queue

Output Parameters: [84] *UINT32* msgCount*
on return from this function, msgCount contains the number of messages on the queue.

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_POINTER	msgCount pointer is NULL.
OS_INVALID_REF	Invalid queue reference

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes:

Example:

```
OSAMsgQRef msgQRef;

OSA_STATUS status;

UINT32     msgCount;


/*  Check for the number of messages on the queue ''msgQRef''.
    Assume ''msgQRef'' has previously been created with the
    OSAMsgQCreate service call. */


status = OSAMsgQPoll(msgQRef, &msgCount);
```

OSAMsgQRecv

Prototype: *OSA_STATUS OSAMsgQRecv(OSAMsgQRef msgQRef, UINT8* recvMsg, UINT32 size, UINT32 timeout);*

Description: This function requests that a message be received from the specified message queue. If the queue is empty, the blocking behavior of the call is determined by the value of the "timeout" argument.

Input Parameters:

[85] *OSAMsgQRef msgQRef*
identifier that uniquely identifies the message queue

[86] *UINT32 size*
size of received message buffer 'recvMsg' in bytes

[87] *UINT32 timeout*
If timeout is set to **OSA_NO_SUSPEND**, this call will not block. If timeout is set to **OSA_SUSPEND**, this call will block until a message is available on the queue. If a timeout value between 1 and 4,294,967,293 is specified, the call will block until a message is available or until the timeout period, in number of OS clock ticks, elapses.

Output Parameters:

[88] *UINT8* recvMsg*
pointer to application supplied buffer to which the received message should be copied

Returns:

OS_SUCCESS	Successful completion of the service. It indicates a message has been copied to the receiving task's "recvMsg" buffer.
OS_INVALID_REF	Invalid queue reference.
OS_INVALID_POINTER	"recvMsg" pointer is NULL.
OS_QUEUE_EMPTY	Indicates the message queue is empty.
OS_TIMEOUT	A timeout has occurred while suspended waiting for a message on an empty queue.
OS_INVALID_SIZE	Indicates that the actual message size is larger than the size of the application receive buffer as indicated by the 'size' parameter

	OS_FAIL	OS specific error.

Attributes Synchronous, may involve scheduler

Blocking – refer to notes for “timeout” argument

Notes: Nucleus Support: Nucleus sends and receives data in word (4 byte) boundaries. Data must be sent and received using long word boundaries.

Example:

```
OSAMsgQRef msgQRef;

OSA_STATUS status;

UINT8     recvMsg[20];


/*  Receive a 20-byte message from the queue ‘‘msgQRef’’. If
    the queue is empty, suspend until the request can be
    satisfied. Assume ‘‘msgQRef’’ has previously been
    created with the OSAMsgQCreate service call. */


status = OSAMsgQRecv(msgQRef, recvMsg, 20, OSA_SUSPEND);
```

OSAMsgQSend

Prototype: *OSA_STATUS OSAMsgQSend(OSmsgQRef msgQRef, UINT32 size, UINT8* msgPtr, UINT32 timeout);*

Description: This function requests that a message be sent to the specified message queue.

Input Parameters:

[89] *OSAMsgQRef msgQRef*
identifier that uniquely identifies the message queue

[90] *UINT32 size*
number of bytes to send

[91] *UINT8 *msgPtr*
starting address of the data

[92] *UINT32 timeout*
If timeout is set to **OSA_NO_SUSPEND**, this call will not block. If timeout is set to **OSA_SUSPEND**, this call will block until there is space available on the queue.

Output Parameters: [93] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid queue reference.
OS_INVALID_POINTER	Message pointer is NULL.
OS_QUEUE_FULL	Indicates the message queue is full.
OS_INVALID_SIZE	Indicates the message size is incompatible with the message size supported by the queue. The maximum size of message that may be sent to the queue is specified in **OSAMsgQCreate().**
OS_FAIL	OS specific error.

Attributes Synchronous, may involve scheduler

Nucleus: Blocking – refer to notes for “timeout” argument

Notes: Nucleus Support: Nucleus sends and receives data in word (4 byte) boundaries. Data must be sent and received using long word boundaries.

RTT Support: RTT does not return OS_QUEUE_FULL as it continues to allocate more space for the queue.

Example:

```
OSAMsgQRef msgQRef;

OSA_STATUS status;

UINT8     msgPtr[20];


/*  Build a 4-byte message to send. */

msgPtr[0] = 0x11;

msgPtr[1] = 0x12;

msgPtr[2] = 0x13;

msgPtr[3] = 0x14;


/*  Send a 4-byte message to the queue ''msgQRef''. Suspend
    the calling task until the message can be sent. Assume
    ''msgQRef'' has previously been created with the
    OSAMsgQCreate service call. */


status = OSAMsgQSend(msgQRef, 4, msgPtr, OSA_SUSPEND);
```

Mailbox Queues

OSAMailboxQCreate

Prototype: *OSA_STATUS OSAMailboxQCreate(OSAMailboxQRef *mboxQRef, char *queueName, UINT32 maxNumber, UINT32 waitingMode);*

Description: This function requests that a mailbox be created.

Input Parameters:

*[94] char *queueName*

[95] 8 character name of queue. . The name does not have to be null-terminated.

[96] *UINT32 maxNumber*
maximum number of messages in the queue

[97] *UINT32 waitingMode*
defines how tasks wait on the queue: OSA_FIFO or OSA_PRIORITY

Output Parameters:

[98] *OSAMailboxQRef *mailboxQRef*
pointer to location to hold mailbox reference allocated by the operating system

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid queue reference.
OS_INVALID_MODE	Invalid waiting mode.
OS_INVALID_SIZE	Invalid queue size.
OS_NO_MBOXES	No available mailboxes left.
OS_FAIL	OS specific error.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes: Mailbox queues are used to hold items that are void pointers.

Nucleus Support: Memory for the queue is configured in the osa_config.h or gbl_config.h files.

RTT Support: RTT only supports full blocking queues and does not support timeouts. Memory comes from the system and not OSA.

WinCE Support: Waiting mode is fixed at priority waiting so ‘waitingMode’ is ignored. Memory for the queue comes from

WinCE and not OSA.

Example:

```
OSAMailboxQRef mboxRef;

OSA_STATUS     status;


/*  Assume mailbox queue ''mboxRef'' is defined as a global.
    This is one of several ways to allocate a reference.
    Create a queue with a capacity of 100 mailboxes. */


status = OSAMailboxQCreate(&mboxRef, ''testQ'', 100,
    OSA_FIFO);
```

OSAMailboxQDelete

Prototype: *OSA_STATUS OSAMailboxQDelete(OSAMailboxQRef mboxQRef);*

Description: This function requests that the specified mailbox queue be deleted.

Input Parameters: [99] *OSAMailboxQRef mboxQRef*
identifier that uniquely identifies the mailbox queue

Output Parameters: [100] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid queue reference.
OS_FAIL	OS specific error.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes:

Example:

```
OSAMailboxQRef mboxRef;

OSA_STATUS     status;


/*  Delete the mailbox queue reference ''mboxRef''. Assume
    ''mboxRef'' has previously been created with the
```

```
    OSAMailboxQCreate service call. */


status = OSAMailboxQDelete(mboxRef);
```

OSAMailboxQPoll

Prototype: *OSA_STATUS OSAMailboxQPoll(OSAMailboxQRef mboxQRef, UINT32* msgCount);*

Description: This function checks the number of messages on the mailbox queue.

Input Parameters: [101] *OSAMailboxQRef mboxQRef*
identifier that uniquely identifies the mailbox queue

Output Parameters: [102] *UINT32* msgCount*
On return from this function, *msgCount* contains the number of message on the queue.

Returns:

OS_SUCCESS	There are one or more items on the queue
OS_INVALID_POINTER	"msgCount" pointer is NULL.
OS_INVALID_REF	Invalid queue reference.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes:

Example:

```
OSAMailboxQRef mboxRef;

OSA_STATUS     status;

UINT32         msgCount;


/*  Check for the number of messages on the queue ''mboxRef''.
    Assume ''mboxRef'' has previously been created with the
    OSAMailboxQCreate service call. */


status = OSAMailboxQPoll(mboxRef, &msgCount);
```

OSAMailboxQRecv

Prototype: *OSA_STATUS OSAMailboxQRecv(OSAMailboxQRef mboxQRef, void **recvMsg, UINT32 timeout);*

Description: This function requests that an item be removed from the specified mailbox queue. If the queue is empty, the blocking behavior of the call is determined by the value of the "timeout" argument.

Input Parameters:

[103] *OSAMailboxQRef mboxQRef*
identifier that uniquely identifies the mailbox queue

[104] *UINT32 timeout*
If timeout is set to **OSA_NO_SUSPEND**, this call will not block. If timeout is set to **OSA_SUSPEND**, this call will block until an item is available on the queue. If a timeout value between 1 and 4,294,967,293 is specified, the call will block until an item is available or until the timeout period, in number of OS clock ticks, elapses.

Output Parameters:

[105] void**
recvMsg will contain the item removed from the queue (i.e. a void*)

Returns:

OS_SUCCESS	Successful completion of the service. It indicates that an item has been removed from the queue.
OS_INVALID_REF	Invalid queue reference.
OS_INVALID_POINTER	"recvMsg" pointer is NULL.
OS_QUEUE_EMPTY	Indicates the mailbox queue is empty.
OS_TIMEOUT	A timeout has occurred while suspended waiting for a message on an empty queue.
OS_FAIL	OS specific error.

Attributes Synchronous, may involve scheduler

Blocking – refer to notes for "timeout" argument

Notes:

Example:

```
OSAMailboxQRef mboxRef;
```

```
OSA_STATUS     status;
void           *recvMsg;

/*  Receive a mailbox message from the queue ‘‘mboxRef’’. If
    the queue is empty, suspend until the request can be
    satisfied. Assume ‘‘mboxRef’’ has previously been
    created with the OSAMailboxQCreate service call. */

status = OSAMailboxQRecv(mboxRef, &recvMsg, OSA_SUSPEND);
```

OSAMailboxQSend

Prototype: *OSA_STATUS OSAMailboxQSend(OSAMailboxQRef mboxQRef, void *msgPtr, UINT32 timeout);*

Description: This function requests that an item be put on the specified mailbox queue.

Input Parameters:

[106] *OSAMailboxQRef mboxQRef*
identifier that uniquely identifies the mailbox queue

[107] *void* msgPtr*
pointer to item (i.e. void*) to put on the mailbox queue

[108] *UINT32 timeout*
If timeout is set to **OSA_NO_SUSPEND**, this call will not block. If timeout is set to **OSA_SUSPEND**, this call will block until an item is available on the queue.

Output Parameters: [109] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid queue reference.
OS_INVALID_POINTER	"msgPtr" pointer is NULL.
OS_QUEUE_FULL	Indicates the mailbox queue is full
OS_FAIL	OS specific error.

Attributes Synchronous, may involve scheduler

Blocking – refer to notes for "timeout" argument

Notes:

Example:

```
OSAMsgQRef mboxRef;

OSA_STATUS status;

void      *msgPtr;

UINT8     *tempBuf;


/*  Allocate a buffer to store the message. Assume the memory
    pool ''memPool'' has previously been created with the
```

```
    OSAMemPoolCreate service call. */


OSAMemPoolAlloc(memPool, 8, &msgPtr, OSA_SUSPEND);

tempBuf = (UINT*)msgPtr;


/*  Build an 8-byte message to send. */

tempBuf[0] = 0x11; tempBuf[1] = 0x12;

tempBuf[2] = 0x13; tempBuf[3] = 0x14;

tempBuf[4] = 0x15; tempBuf[5] = 0x16;

tempBuf[6] = 0x17; tempBuf[7] = 0x18;


/*  Send a mailbox message to the queue ''mboxRef''. Suspend
    the calling task until the message can be sent. Assume
    ''mboxRef'' has previously been created with the
    OSAMailboxQCreate service call. */


status = OSAMailboxQSend(mboxRef, msgPtr, OSA_SUSPEND);
```

Timers

OSATimerCreate

Prototype: *OSA_STATUS OSATimerCreate(OSATimerRef* timerRef);*

Description: This function allocates a timer. The state of the allocated timer is inactive. OSATimerStart() is used to activate the timer.

Input Parameters: [110] *OSATimerRef* timerRef*
address to store a reference to the timer allocated by the operating system

Output Parameters: [111] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_NO_TIMERS	No available timers in the system.
OS_INVALID_REF	Input argument "timerRef" is a NULL pointer.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes: The task that created the timer has exclusive access to the timer until it is deleted. The timer resolution is in units of OS clock ticks. The function OSAGetClockRate() can be used to get the number of milliseconds per OS clock tick.

Example:

```
OSATimerRef timerRef;

OSA_STATUS  status;


/*  Assume timer queue reference is defined as a global. This
    is one of several ways to allocate a reference. */


status = OSATimerCreate(&timerRef);
```

OSATimerStart

Prototype: *OSA_STATUS OSATimerStart(OSATimerRef timerRef, UINT32 initialTime, UINT32 rescheduleTime, void (*callBackRoutine)(UINT32), UINT32 timerArgc);*

Description: This function requests that an inactive timer be started and the callback function executed at the expiration of the timer.

Input Parameters:

[112] *UINT32 initialTime*
initial expiration time in OS clock ticks

[113] *UINT32 rescheduleTime*
If 0, cyclic timing is disabled and the timer only expires once. If not zero, it indicates the period, in OS clock ticks, of a cyclic timer.

[114] *Void callBackRoutine(UINT32)*
Specifies the application routine to execute each time the timer expires. The callback function must not invoke any "blocking" operating system calls.

[115] *UINT32 timerArgc*
argument to be passed to callback routine on expiration

Output Parameters:

[116] *OSATimerRef *timerRef*
OS supplied timer reference

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Input argument "timerRef" is not a valid timer reference.
OS_INVALID_PTR	Input argurment "callBackRoutine" is a NULL pointer
OS_FAIL	Timer is still active.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes: The timer callback function is invoked outside the context of the task that started the timer.

The callback function must not invoke any "blocking" operating system calls.

Example:

```
OSATimerRef timerRef;

OSA_STATUS  status;

void        timerRoutine(UINT32);


/*  Start the timer ''timerRef'' with an expiration function
    ''timerRoutine'', with an initial expiration of 20 timer
    ticks, and pass the argument 0x1234 into the function
    ''timerRoutine'' when the timer expires. After the
    initial expiration, the timer expires every 56 timer
    ticks. Assume ''timerRef'' has previously been created
    with the OSATimerCreate service call. */


status = OSATimerStart(timerRef, 20, 56,
    timerRoutine(UINT32), 0x1234);
```

OSATimerStop

Prototype: *OSA_STATUS OSATimerStop(OSATimerRef timerRef);*

Description: This function requests that the state of an active timer be changed to inactive. Calling this function when the state of the timer is inactive has no effect.

Input Parameters: [117] *OSATimerRef timerRef*
OS supplied timer reference

Output Parameters: [118] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid timer reference.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes:

Example:

```
OSATimerRef timerRef;

OSA_STATUS  status;
```

```
/*  Stop the timer ‘‘timerRef’’. Assume ‘‘timerRef’’ has
    previously been created with the OSATimerCreate service
    call. */


status = OSATimerStop(timerRef);
```

OSATimerDelete

Prototype: *OSA_STATUS OSATimerDelete(OSATimerRef timerRef);*

Description: This function requests that the specified timer be deleted. The timer must be inactive when this function is called.

Input Parameters: [119] *OSATimerRef timerRef*
timer reference that was allocated when the timer was created

Output Parameters: [120] None

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid timer reference.
OS_FAIL	Timer is still active.

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes: The timer must be inactive when this function is called. The application must prevent use of the timer during and after deletion.

Example:

```
OSATimerRef timerRef;

OSA_STATUS  status;


/*  Delete the timer ‘‘timerRef’’. Assume ‘‘timerRef’’ has
    previously been created with the OSATimerCreate service
    call. */


status = OSATimerDelete(timerRef);
```

OSATimerGetStatus

Prototype: *OSA_STATUS OSATimerGetStatus(OSATimerRef timerRef, OSATimerStatus* status);*

Description: This function requests that the status of the specified timer be returned in the *OSATimerStatus* structure.

Input Parameters:

[121] *OSATimerRef timerRef*
reference assigned when the timer was created

[122] *OSATimerStatus* status*
pointer to the structure that will be filled in

Output Parameters: [123] Structure filled by the system:

```
Struct OSATimerStatus{
UINT32 status;      (OSA_ENABLED, OSA_DISABLED)
UINT32 expirations; (number of expirations for
                     cyclic timers)
}
```

Returns:

OS_SUCCESS	Successful completion of the service.
OS_INVALID_REF	Invalid timer reference

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes: The timer status is OSA_ENABLED if the timer is active or OSA_DISABLED if the timer is inactive.

Example:

```
OSATimerRef    timerRef;
OSA_STATUS     status;
OSATimerStatus timerStatus;

/*  Get the status of the timer ''timerRef''. Assume
    ''timerRef'' has previously been created with the
    OSATimerCreate service call. */

status = OSATimerGetStatus(timerRef, &timerStatus);
```

OS Clock

OSAGetTicks

Prototype: *UINT32 OSAGetTicks (void);*

Description: This function requests the elapsed time, in OS clock tick, since the last system start-up.

Input Parameters: [124] None

Output Parameters: [125] None

Returns: UINT32 Time elapsed in OS clock ticks

Attributes Synchronous, Non-blocking, no scheduler involvement

Notes:

Example:

```
UINT32 ticks;


ticks = OSAGetTicks(void);
```

Revision History

Table 2: Revision History

Document No and Revision	Int Rev	Description	Date
0.1		Draft	Aug 26, 2013
0.2		Transferred from 0.1	Oct 26, 2017