



**WITTENSTEIN**

**SAFERTOS APPLICATION NOTE: #34-172-AN-001**

---

# **UPGRADING FROM FreeRTOS TO SAFERTOS**

---

**WITTENSTEIN high integrity systems is a trading name of WITTENSTEIN aerospace & simulation ltd**

**Proprietary to WITTENSTEIN aerospace & simulation ltd.**

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROPRIETARY INFORMATION AND ALL INFORMATION, TECHNICAL DATA, DESIGNS, INCLUDING BUT NOT LIMITED TO DATA DISCLOSED AND/OR PROVIDED HEREIN, IS AND REMAINS THE EXCLUSIVE PROPERTY OF WITTENSTEIN aerospace & simulation ltd. IT IS STRICTLY PROHIBITED TO DISCLOSE ANY INFORMATION TO THIRD PARTIES WITHOUT THE PRIOR WRITTEN CONSENT OF WITTENSTEIN aerospace & simulation ltd. THE RECIPIENT OF THIS DOCUMENT, BY IT'S RETENTION AND USE AGREES TO HOLD IN CONFIDENCE ALL PROPRIETARY INFORMATION PROVIDED WITHIN THIS DOCUMENT.

Copyright WITTENSTEIN aerospace & simulation ltd date as document, all rights reserved.



WITTENSTEIN

## CONTENTS

CONTENTS .....	2
REFERENCED DOCUMENTS .....	2
CHAPTER 1 INTRODUCTION.....	3
1.1 KEY DIFFERENCES .....	4
CHAPTER 2 MEMORY ALLOCATION .....	5
2.1 MEMORY ALLOCATION SCHEME.....	6
2.2 CHANGES TO XTASKCREATE().....	6
2.3 CHANGES TO XQUEUECREATE().....	7
2.4 CHANGES TO XSEMAPHORECREATEBINARY() .....	8
2.5 CHANGES TO XSEMAPHORECREATECOUNTING().....	8
2.6 CHANGES TO XTIMERCREATE() .....	8
2.7 CHANGES TO XEVENTGROUPCREATE() .....	9
2.8 CHANGES TO XMUTEXCREATE().....	10
CHAPTER 3 FUNCTIONALITY.....	11
3.1 RESTRICTED FUNCTIONALITY .....	12
3.1.1 Co-routines .....	12
3.1.2 Stream Buffers.....	12
3.1.3 Removed API Functions .....	12
CHAPTER 4 FUNCTION AND MACRO NAMES .....	13
4.1 FUNCTION AND MACRO NAMES .....	14
CHAPTER 5 ERROR REPORTING .....	32
5.1 SIDE EFFECTS OF INCREASED ERROR REPORTING .....	33
CHAPTER 6 MISCELLANEOUS.....	34
6.1 HEADER FILES.....	35
6.2 HOOK FUNCTIONS .....	35
6.3 FILE NAMES .....	35

## REFERENCED DOCUMENTS

Ref #	Document	Description
1	34-172-MAN-1-ccc-aaa	SAFERTOS Product Variant User Manual, Author WHIS.



**WITTENSTEIN**

# **CHAPTER 1**

# **INTRODUCTION**

---



## 1.1 KEY DIFFERENCES

---

FreeRTOS and **SAFERTOS** share a similar usage model but are not direct drop in replacements for each other. This document highlights the areas requiring modification when moving an application from FreeRTOS to **SAFERTOS**. These differences primarily arise from:

### 1. Memory allocation

Each task and queue created consumes a small amount of RAM. Under FreeRTOS the required RAM is automatically dynamically allocated at run time. **SAFERTOS** does not permit dynamic memory allocation so the required RAM must instead be statically allocated at compile time, then manually passed into the create functions for the various kernel objects (i.e. `xTaskCreate()`, `xQueueCreate`, `xSemaphoreCreateBinary()`, `xSemaphoreCreateCounting()`, `xTimerCreate()`, `xEventGroupCreate()` and `xMutexCreate()` API functions).

### 2. Function parameter checking

FreeRTOS contains very little in the way of API function input parameter checking. As a result many FreeRTOS API functions either just return a simple pass or fail result, or do not return any status information at all.

**SAFERTOS** checks the validity of every appropriate input parameter. This means not only do more **SAFERTOS** API functions return status information but the status information returned is also more detailed. The **SAFERTOS** function naming convention states that API functions shall be prefixed with their return type – so changing the type returned by a function also requires a small change to the function name.

### 3. Internal data checking

**SAFERTOS** performs validity and consistency tests on its key internal data – calling an application defined error hook function should such a test fail. FreeRTOS does not require an equivalent hook function so one must be provided when upgrading to instead use **SAFERTOS**.

### 4. Restricted Functionality

**SAFERTOS** supports only the core components of FreeRTOS, therefore some FreeRTOS functionality has been restricted. Refer to Section 3.1 for more details.

To facilitate the upgrade processes a small **SAFERTOS** demonstration project is provided for your reference. This project is the **SAFERTOS** equivalent to a sub-set of the standard FreeRTOS demonstration application – allowing the two to be compared.



**WITTENSTEIN**

# **CHAPTER 2**

# **MEMORY ALLOCATION**

---



## 2.1 MEMORY ALLOCATION SCHEME

---

FreeRTOS projects have to include implementations for the dynamic memory allocation functions `pvPortMalloc()` and `vPortFree()`. The implementation would normally be contained within a file called `heap_1.c`, `heap_2.c`, `heap_3.c` or `heap_4.c`. **SAFERTOS** does not permit any form of dynamic memory allocation so this file should be removed from the project.

## 2.2 CHANGES TO `xTASKCREATE()`

---

The lack of dynamic memory allocation necessitates that the **SAFERTOS** version of `xTaskCreate()` requires more parameters than its FreeRTOS counterpart. In addition, a **SAFERTOS** product variant may require additional parameters when creating a task if, for example, a Memory Protection Unit (MPU) or Memory Management Unit (MMU) is supported. Consequently, the **SAFERTOS** version of `xTaskCreate()` accepts just 2 parameters – `pxTaskParameters` which is a pointer to an `xTaskParameters` structure and `pxCreatedTask` which is used to pass back the handle of the created task. This is similar to the FreeRTOS API function `xTaskCreateRestricted()`.

The members of the `xTaskParameter` structure are specific to each product variant and are described fully in the **SAFERTOS** Product Variant User Manual [Reference 1]. As a minimum, all product variants require the following members to be present:

1. `pvTaskCode`  
A pointer to the function that implements the task.
2. `pcTaskName`  
A descriptive name for the task. This is mainly used to facilitate debugging.
3. `pxTCB`  
A pointer to a statically declared `xTCB` used by the kernel to hold the task data structures.
4. `pcStackBuffer`  
A pointer to the statically declared buffer to be used by the kernel to hold the task stack. Care must be taken to ensure that the stack buffer is aligned correctly.
5. `ulStackDepthBytes`  
The size of the buffer pointed to by the `pcStackBuffer` parameter. **Note that this is the size in bytes.**



WITTENSTEIN

6. pvParameters

A pointer that will be used as the parameter for the task being created.

7. uxPriority

The priority at which the task will run.

Refer to the API reference section of the **SAFERTOS** Product Variant User Manual [Reference 1] for a full description of the xTaskCreate() function, and the accompanying demo application for some usage examples.

## 2.3 CHANGES TO XQUEUECREATE()

---

The lack of dynamic memory allocation and increased error checking necessitates that the **SAFERTOS** version of xQueueCreate() requires three more parameters than its FreeRTOS counterpart.

1. pcQueueMemory

Points to the statically declared buffer to be used by the kernel to hold the queue data structures and storage area.

2. uxBufferLength

The size of the buffer pointed to by the pcQueueMemory parameter.

3. pxQueue

Used to return a handle to the queue being created.

The FreeRTOS xQueueCreate() function returns either a handle to the created queue, or NULL should the function be unable to create the queue for any reason. The application need therefore only check the return value against NULL to know whether a valid queue handle was returned or not. The improved error detection and reporting provided by **SAFERTOS** replaces this binary pass/fail return value with a set of descriptive status codes – necessitating that the queue handle instead be passed out of the function using this new reference parameter.

Refer to the API reference section in the **SAFERTOS** Product Variant User Manual [Reference 1] for a full description and the accompanying demo application for some usage examples.



WITTENSTEIN

## 2.4 CHANGES TO XSEMAPHORECREATEBINARY()

---

The lack of dynamic memory allocation and increased error checking necessitates that the **SAFERTOS** version of `xSemaphoreCreateBinary()` requires different parameters to its FreeRTOS counterpart.

1. `pcSemaphoreBuffer`

Points to the statically declared buffer to be used by the kernel to hold the semaphore data structure. The buffer must have a size of `portQUEUE_OVERHEAD_BYTES` and be 8 byte aligned.

2. `pxSemaphore`

Used to return a handle to the semaphore being created.

Refer to the API reference section in the **SAFERTOS** Product Variant User Manual [Reference 1] for a full description and a list of the possible error codes returned by **SAFERTOS**. Refer to the accompanying demo application for some usage examples.

## 2.5 CHANGES TO XSEMAPHORECREATECOUNTING()

---

The lack of dynamic memory allocation and increased error checking necessitates that the **SAFERTOS** version of `xSemaphoreCreateCounting()` requires two more parameters than its FreeRTOS counterpart.

1. `pcSemaphoreBuffer`

Points to the statically declared buffer to be used by the kernel to hold the semaphore data structure. The buffer must have a size of `portQUEUE_OVERHEAD_BYTES` and be 8 byte aligned.

2. `pxSemaphore`

Used to return a handle to the semaphore being created.

Refer to the API reference section in the **SAFERTOS** Product Variant User Manual [Reference 1] for a full description and a list of the possible error codes returned by **SAFERTOS**. Refer to the accompanying demo application for some usage examples.

## 2.6 CHANGES TO XTIMERCREATE()

---

The lack of dynamic memory allocation and increased error checking necessitates that the **SAFERTOS** version of `xTimerCreate()` requires more parameters than its FreeRTOS counterpart.

1. `pxNewTimer`





**WITTENSTEIN**

Points to the statically declared buffer to be used by the kernel to hold the timer data block. The buffer must be of type `timerControlBlockType`.

2. `pxTimerHandle`

Used to return a handle to the timer being created.

3. `pxTimerInstance`

**SAFERTOS** supports multiple instance of the timer module. This parameter identifies which instance the timer belongs to.

Refer to the API reference section in the **SAFERTOS** Product Variant User Manual [Reference 1] for a full description and a list of the possible error codes returned by **SAFERTOS**. Refer to the accompanying demo application for some usage examples.

## 2.7 CHANGES TO `xEVENTGROUPCREATE()`

---

The lack of dynamic memory allocation and increased error checking necessitates that the **SAFERTOS** version of `xEventGroupCreate()` requires two more parameters than its FreeRTOS counterpart.

1. `pxEventGroup`

Points to the statically declared `eventGroupType` structure buffer to be used by the kernel to hold the created event group.

2. `pxEventGroupHandle`

Used to return a handle to the event group being created.

The FreeRTOS `xEventGroupCreate()` function returns either a handle to the created event group, or `NULL` should the function be unable to create the event group because of insufficient heap. The application need therefore only check the return value against `NULL` to know whether a valid event group handle was returned or not. The improved error detection and reporting provided by **SAFERTOS** replaces this binary pass/fail return value with a set of descriptive status codes – necessitating that the event group handle instead be passed out of the function using this new reference parameter.

Refer to the API reference section in the **SAFERTOS** Product Variant User Manual [Reference 1] for a full description and the accompanying demo application for some usage examples.



WITTENSTEIN

## 2.8 CHANGES TO XMUTEXCREATE()

---

The lack of dynamic memory allocation and increased error checking necessitates that the **SAFERTOS** function `xMutexCreate()` requires two more parameters than its FreeRTOS counterparts (`xSemaphoreCreateMutex()` and `xSemaphoreCreateRecursiveMutex()`).

1. `pcMutexBuffer`

Points to the statically declared buffer to be used by the kernel to hold the data required to manage the mutex.

2. `pxMutex`

Used to return a handle to the mutex being created.

The FreeRTOS `xSemaphoreCreateMutex()` and `xSemaphoreCreateRecursiveMutex()` functions return either a handle to the created mutex, or NULL should the function be unable to create the mutex. The application need therefore only check the return value against NULL to know whether a valid mutex handle was returned or not. The improved error detection and reporting provided by **SAFERTOS** replaces this with a set of descriptive status codes – necessitating that the mutex handle instead be passed out of the function using this new reference parameter.

Refer to the API reference section in the **SAFERTOS** Product Variant User Manual [Reference 1] for a full description and the accompanying demo application for some usage examples.



**WITTENSTEIN**

# **CHAPTER 3**

# **FUNCTIONALITY**

---



## 3.1 RESTRICTED FUNCTIONALITY

---

The **SAFERTOS** functionality has been restricted to include only the core components necessary. This is standard practice for source code that is intended to undergo formal audit or certification as it greatly reduces the test burden.

### 3.1.1 Co-routines

**SAFERTOS** does not provide a co-routine implementation.

### 3.1.2 Stream Buffers

FreeRTOS v10 introduced a new feature, stream buffers. **SAFERTOS** does not provide an implementation of this feature.

### 3.1.3 Removed API Functions

Refer to Table 4-1 for a FreeRTOS to **SAFERTOS** API function cross reference.



**WITTENSTEIN**

# **CHAPTER 4**

# **FUNCTION AND MACRO NAMES**

---



## 4.1 FUNCTION AND MACRO NAMES

The **SAFERTOS** function naming convention requires each function name to be prefixed by a character that describes its return type. In particular functions that return a value of type `portBASE_TYPE` are prefixed by an 'x', and functions that return a void are prefixed with a 'v'. A greater number of **SAFERTOS** functions than FreeRTOS functions return non void values, necessitating that the prefixes assigned to these functions be updated accordingly.

Table 4-1 provides a cross reference between FreeRTOS API functions and macros, and their **SAFERTOS** equivalents. This analysis was conducted using v10.0.1 of the FreeRTOS API and v6.2 of the **SAFERTOS** API.

Table 4-1 FreeRTOS to **SAFERTOS** function name mapping

FreeRTOS Function	<b>SAFERTOS</b> Equivalent	Notes
Task and scheduler API		
<code>portSWITCH_TO_USER_MODE()</code>	<code>vMPUTaskExecuteInUnprivilegedMode()</code>	This API function is only provided for <b>SAFERTOS</b> product variants that support use of an MPU.
<code>vTaskAllocateMPURegions()</code>	<code>xMPUSetTaskRegions()</code>	This API function is only provided for <b>SAFERTOS</b> product variants that support use of an MPU.
<code>xTaskAbortDelay()</code>	Not implemented.	
<code>xTaskCallApplicationTaskHook()</code>	Not implemented.	
<code>xTaskCheckForTimeOut()</code>	Not implemented.	



**WITTENSTEIN**

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
xTaskCreate()	xTaskCreate()	The SAFERTOS version requires a pointer to a structure to be passed in place of the task parameters, as detailed within this application note. FreeRTOS returns pdPASS or errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY. SAFERTOS also returns either pdPASS, or an error code should it not be possible to create the task. The error codes errNULL_PARAMETER_SUPPLIED, errINVALID_TASK_CODE_POINTER, errINVALID_PRIORITY, errINVALID_BYTE_ALIGNMENT and errTASK_STACK_ALREADY_IN_USE are returned by all product variants. Other error codes may be returned depending on the product variant; refer to the SAFERTOS Product Variant User Manual [Reference 1].
xTaskCreateStatic()	xTaskCreate()	FreeRTOS provides this function to enable tasks to be created with a statically allocated stack buffer. SAFERTOS tasks are by default created with statically allocated stack buffers.
xTaskCreateRestricted()	xTaskCreate()	FreeRTOS only provides this API function for systems that include an MPU implementation. The parameters supplied to the SAFERTOS function are discussed within this application note.
vTaskDelay()	xTaskDelay()	SAFERTOS returns pdPASS or errSCHEDULER_IS_SUSPENDED.
vTaskDelayUntil()	xTaskDelayUntil()	SAFERTOS returns pdPASS, errSCHEDULER_IS_SUSPENDED, errNULL_PARAMETER_SUPPLIED or errDID_NOT_YIELD.
vTaskDelete()	xTaskDelete()	SAFERTOS returns pdPASS or errINVALID_TASK_HANDLE.
taskDISABLE_INTERRUPTS()	Not implemented.	For SAFERTOS, use taskENTER_CRITICAL().
taskENABLE_INTERRUPTS()	Not implemented.	For SAFERTOS, use taskEXIT_CRITICAL().



WITTENSTEIN

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
taskENTER_CRITICAL()	taskENTER_CRITICAL()	
taskENTER_CRITICAL_FROM_ISR()	Not implemented.	For SAFERTOS, use taskSET_INTERRUPT_MASK_FROM_ISR(), which returns a value that should be passed to the corresponding call to taskCLEAR_INTERRUPT_MASK_FROM_ISR(). The behavior of these macros depends on the product variant. These macros will not affect the critical nesting count and are not a precise equivalent to the FreeRTOS functions.
taskEXIT_CRITICAL()	taskEXIT_CRITICAL()	
taskEXIT_CRITICAL_FROM_ISR()	Not implemented.	For SAFERTOS, use taskCLEAR_INTERRUPT_MASK_FROM_ISR(), which takes as its parameter the value returned by the corresponding taskSET_INTERRUPT_MASK_FROM_ISR(). The behavior of these macros depends on the product variant. These macros will not affect the critical nesting count and are not a precise equivalent to the FreeRTOS functions.
xTaskGetApplicationTaskTag()	Not implemented.	This function is used to access a “tag” value within the task TCB. It’s a form of thread local storage with application-defined meaning. For SAFERTOS, it should be possible instead to make use of pvTaskTLSObjectGet(), depending on application requirements. The TLS Object is an arbitrary void pointer associated with a task, defined at task creation time. (Note that FreeRTOS also provides an array of thread local storage pointers, in addition to the “task tag”.)
xTaskGetCurrentTaskHandle()	xTaskGetCurrentTaskHandle()	
xTaskGetIdleTaskHandle()	Not implemented.	
xTaskGetHandle()	Not implemented.	
uxTaskGetNumberOfTasks()	Not implemented.	





**WITTENSTEIN**

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
vTaskGetRunTimeStats()	Not implemented.	SAFERTOS can provide run time statistics on a task by task basis – refer to the discussion of xCalculateCPUUsage() in the SAFERTOS Product Variant User Manual [Reference 1].
xTaskGetSchedulerState()	xTaskIsSchedulerStarted() xTaskIsSchedulerStartedFromISR()	The SAFERTOS functions return pdTRUE if the scheduler has been started.
uxTaskGetStackHighWaterMark()	Not implemented.	
eTaskGetState()	Not implemented.	
uxTaskGetSystemState()	Not implemented.	
vTaskGetTaskInfo()	Not implemented.	
pvTaskGetThreadLocalStoragePointer()	pvTaskTLSObjectGet()	FreeRTOS provides for an array of TLS pointers, the size of which is determined at compile time. SAFERTOS provides a single TLS object, which is a void pointer that may be used to reference an arbitrary structure defined, populated and maintained according to application requirements.
pcTaskGetTaskName()	Not implemented.	
xTaskGetTickCount()	xTaskGetTickCount()	
xTaskGetTickCountFromISR()	xTaskGetTickCountFromISR()	
vTaskList()	Not implemented.	
xTaskNotify()	xTaskNotifySend()	Note that SAFERTOS xTaskNotifySend() has differently ordered parameters compared with FreeRTOS xTaskNotify().
xTaskNotifyAndQuery()	Not implemented.	
xTaskNotifyAndQueryFromISR()	Not implemented.	
xTaskNotifyFromISR()	xTaskNotifySendFromISR()	Note that SAFERTOS xTaskNotifySendFromISR() has differently ordered parameters compared with FreeRTOS xTaskNotifyFromISR().



**WITTENSTEIN**

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
xTaskNotifyGive()	Not implemented.	For SAFERTOS, use xTaskNotifySend() to implement the FreeRTOS “lightweight semaphore” task notification use case if required.
xTaskNotifyGiveFromISR()	Not implemented.	For SAFERTOS, use xTaskNotifySendFromISR() to implement the FreeRTOS “lightweight semaphore” task notification use case if required.
xTaskNotifyStateClear()	Not implemented.	
ulTaskNotifyTake()	Not implemented.	For SAFERTOS, use xTaskNotifyWait() to implement the FreeRTOS “lightweight semaphore” task notification use case if required.
xTaskNotifyWait()	xTaskNotifyWait()	
uxTaskPriorityGet()	xTaskPriorityGet()	Whereas the FreeRTOS version returns the priority, the SAFERTOS version passes the priority out using a reference parameter. This is to permit the return value to provide more detailed error reporting. SAFERTOS returns pdPASS, errNULL_PARAMETER_SUPPLIED or errINVALID_TASK_HANDLE.
vTaskPrioritySet()	xTaskPrioritySet()	SAFERTOS returns pdPASS, errINVALID_PRIORITY or errINVALID_TASK_HANDLE.
vTaskResume()	xTaskResume()	SAFERTOS returns pdPASS, errNULL_PARAMETER_SUPPLIED, errINVALID_TASK_HANDLE or errTASK_WAS_NOT_SUSPENDED.
xTaskResumeAll()	xTaskResumeScheduler()	The name was changed to be more descriptive. FreeRTOS returns a portBASE_TYPE that simply indicates whether or not a yield was performed within the function call; SAFERTOS does the same, but can also return the error code errSCHEDULER_WAS_NOT_SUSPENDED.



**WITTENSTEIN**

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
xTaskResumeFromISR()	Not implemented.	This function is often misused under FreeRTOS and is considered unsafe.
vTaskSetApplicationTaskTag()	Not implemented.	For SAFERTOS, arbitrary thread local storage may be implemented using the TLS object during task creation along with the pvTaskTLSObjectGet() function. The TLS object may be used to access thread (task) local storage structures of arbitrary size and complexity according to application requirements.
vTaskSetThreadLocalStoragePointer()	Not implemented.	For SAFERTOS, arbitrary thread local storage may be implemented using the TLS object during task creation along with the pvTaskTLSObjectGet() function. The TLS object may be used to access thread (task) local storage structures of arbitrary size and complexity according to application requirements.
vTaskSetTimeOutState()	Not implemented.	
vTaskStartScheduler()	xTaskStartScheduler()	The SAFERTOS version requires a parameter that indicates whether or not configuration checks should be performed before starting the scheduler. SAFERTOS also returns a number of error codes should it not be possible to start the scheduler. The error codes errNO_TASKS_CREATED and errSCHEDULER_ALREADY_RUNNING are returned by all product variants. Other error codes may be returned depending on the product variant; refer to the SAFERTOS Product Variant User Manual [Reference 1].
vTaskStepTick()	Not implemented.	
vTaskSuspend()	xTaskSuspend()	SAFERTOS returns pdPASS, errSCHEDULER_IS_SUSPENDED, errINVALID_TASK_HANDLE or errTASK_ALREADY_SUSPENDED.
vTaskSuspendAll()	vTaskSuspendScheduler()	The name was changed to be more descriptive.
taskYIELD()	taskYIELD()	



WITTENSTEIN

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
Queues API		
vQueueAddToRegistry()	Not implemented.	Provided as a separate queue_register.c/h on request
xQueueAddToSet()	Not implemented.	
xQueueCreate()	xQueueCreate()	The SAFERTOS version requires extra parameters as detailed within this application note. FreeRTOS returns the handle of the created queue, or NULL if the queue could not be created; SAFERTOS returns pdPASS, errINVALID_BYTE_ALIGNMENT, errINVALID_QUEUE_LENGTH, errINVALID_BUFFER_SIZE or errNULL_PARAMETER_SUPPLIED.
xQueueCreateSet()	Not implemented.	
xQueueCreateStatic()	xQueueCreate()	The SAFERTOS version requires parameters as detailed within this application note. FreeRTOS returns the handle of the created queue, or NULL if the queue could not be created; SAFERTOS returns pdPASS, errINVALID_BYTE_ALIGNMENT, errINVALID_QUEUE_LENGTH, errINVALID_BUFFER_SIZE or errNULL_PARAMETER_SUPPLIED
vQueueDelete()	Not implemented.	The FreeRTOS version just frees the memory allocated to the queue, whereas SAFERTOS does not dynamically allocate or free memory. It is not considered safe to delete queues.
pcQueueGetName()	Not implemented.	
xQueueIsQueueEmptyFromISR()	Not implemented.	
xQueueIsQueueFullFromISR()	Not implemented.	



**WITTENSTEIN**

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
uxQueueMessagesWaiting()	xQueueMessagesWaiting()	Whereas the FreeRTOS version returns the number of messages, the SAFERTOS version passes the number of messages out using a reference parameter. This is to permit the return value to provide more detailed error reporting. SAFERTOS returns pdPASS, errNULL_PARAMETER_SUPPLIED or errINVALID_QUEUE_HANDLE.
uxQueueMessagesWaitingFromISR()	Not implemented.	
xQueueOverwrite()	Not implemented.	
xQueueOverwriteFromISR()	Not implemented.	
xQueuePeek()	xQueuePeek()	FreeRTOS returns either pdPASS or errQUEUE_EMPTY; SAFERTOS returns pdPASS, errSCHEDULER_IS_SUSPENDED, errINVALID_QUEUE_HANDLE, errNULL_PARAMETER_SUPPLIED or errQUEUE_EMPTY.
xQueuePeekFromISR()	Not implemented.	
xQueueReceive()	xQueueReceive()	FreeRTOS returns either pdPASS or errQUEUE_EMPTY; SAFERTOS returns pdPASS, errSCHEDULER_IS_SUSPENDED, errINVALID_QUEUE_HANDLE, errNULL_PARAMETER_SUPPLIED or errQUEUE_EMPTY.
xQueueReceiveFromISR()	xQueueReceiveFromISR()	FreeRTOS returns either pdPASS or pdFAIL; SAFERTOS returns pdPASS, errNULL_PARAMETER_SUPPLIED, errINVALID_QUEUE_HANDLE or errQUEUE_EMPTY.
xQueueRemoveFromSet()	Not implemented.	
xQueueReset()	Not implemented.	
xQueueSelectFromSet()	Not implemented.	
xQueueSelectFromSetFromISR()	Not implemented.	



**WITTENSTEIN**

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
xQueueSend()	xQueueSend()	FreeRTOS returns either pdPASS or errQUEUE_FULL; SAFERTOS returns pdPASS, errSCHEDULER_IS_SUSPENDED, errNULL_PARAMETER_SUPPLIED, errINVALID_QUEUE_HANDLE or errQUEUE_FULL.
xQueueSendToFront()	xQueueSendToFront()	V6.2 onwards. FreeRTOS returns either pdPASS or errQUEUE_FULL; SAFERTOS returns pdPASS, errSCHEDULER_IS_SUSPENDED, errNULL_PARAMETER_SUPPLIED, errINVALID_QUEUE_HANDLE or errQUEUE_FULL.
xQueueSendToBack()	xQueueSend()	FreeRTOS returns either pdPASS or errQUEUE_FULL; SAFERTOS returns pdPASS, errSCHEDULER_IS_SUSPENDED, errNULL_PARAMETER_SUPPLIED, errINVALID_QUEUE_HANDLE or errQUEUE_FULL.
xQueueSendToBackFromISR()	xQueueSendFromISR()	FreeRTOS returns either pdPASS or errQUEUE_FULL; SAFERTOS returns pdPASS, errNULL_PARAMETER_SUPPLIED, errINVALID_QUEUE_HANDLE or errQUEUE_FULL.
xQueueSendFromISR()	xQueueSendFromISR()	FreeRTOS returns either pdPASS or errQUEUE_FULL; SAFERTOS returns pdPASS, errNULL_PARAMETER_SUPPLIED, errINVALID_QUEUE_HANDLE or errQUEUE_FULL.
xQueueSendToFrontFromISR()	xQueueSendToFrontFromISR()	V6.2 onwards. FreeRTOS returns either pdPASS or errQUEUE_FULL; SAFERTOS returns pdPASS, errNULL_PARAMETER_SUPPLIED, errINVALID_QUEUE_HANDLE or errQUEUE_FULL.
uxQueueSpacesAvailable()	Not implemented.	



**WITTENSTEIN**

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
Semaphores API		
vSemaphoreCreateBinary()	xSemaphoreCreateBinary()	The SAFERTOS version requires different parameters as detailed within this application note. SAFERTOS returns pdPASS, errINVALID_BYTE_ALIGNMENT or errNULL_PARAMETER_SUPPLIED.
xSemaphoreCreateBinary()	xSemaphoreCreateBinary()	The SAFERTOS version requires different parameters as detailed within this application note. SAFERTOS returns pdPASS, errINVALID_BYTE_ALIGNMENT or errNULL_PARAMETER_SUPPLIED.
xSemaphoreCreateBinaryStatic()	xSemaphoreCreateBinary()	The SAFERTOS version requires different parameters as detailed within this application note. SAFERTOS returns pdPASS, errINVALID_BYTE_ALIGNMENT or errNULL_PARAMETER_SUPPLIED.
xSemaphoreCreateCounting()	xSemaphoreCreateCounting()	The SAFERTOS version requires different parameters as detailed within this application note. FreeRTOS returns the handle of the created semaphore, or NULL if the semaphore could not be created; SAFERTOS returns pdPASS, errINVALID_BYTE_ALIGNMENT, errINVALID_INITIAL_SEMAPHORE_COUNT, errINVALID_QUEUE_LENGTH or errNULL_PARAMETER_SUPPLIED.
xSemaphoreCreateCountingStatic()	xSemaphoreCreateCounting()	The SAFERTOS version requires different parameters as detailed within this application note. FreeRTOS returns the handle of the created semaphore, or NULL if the semaphore could not be created; SAFERTOS returns pdPASS, errINVALID_BYTE_ALIGNMENT, errINVALID_INITIAL_SEMAPHORE_COUNT, errINVALID_QUEUE_LENGTH or errNULL_PARAMETER_SUPPLIED.



WITTENSTEIN

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
xSemaphoreCreateMutex()	xMutexCreate()	<p>The <b>SAFERTOS</b> version requires parameters as detailed within this application note.</p> <p>FreeRTOS returns the handle of the created mutex, or NULL if the mutex could not be created; <b>SAFERTOS</b> returns pdPASS, errQUEUE_ALREADY_IN_USE, errINVALID_BYTE_ALIGNMENT or errNULL_PARAMETER_SUPPLIED</p> <p>Mutexes always support recursion in <b>SAFERTOS</b>, FreeRTOS provides different APIs for recursive and non-recursive mutexes.</p>
xSemaphoreCreateMutexStatic()	xMutexCreate()	<p>The <b>SAFERTOS</b> version requires parameters as detailed within this application note.</p> <p>FreeRTOS returns the handle of the created mutex, or NULL if the mutex could not be created; <b>SAFERTOS</b> returns pdPASS, errQUEUE_ALREADY_IN_USE, errINVALID_BYTE_ALIGNMENT or errNULL_PARAMETER_SUPPLIED</p> <p>Mutexes always support recursion in <b>SAFERTOS</b>, FreeRTOS provides different APIs for recursive and non-recursive mutexes.</p>
xSemaphoreCreateRecursiveMutex()	xMutexCreate()	<p>The <b>SAFERTOS</b> version requires parameters as detailed within this application note.</p> <p>FreeRTOS returns the handle of the created mutex, or NULL if the mutex could not be created; <b>SAFERTOS</b> returns pdPASS, errQUEUE_ALREADY_IN_USE, errINVALID_BYTE_ALIGNMENT or errNULL_PARAMETER_SUPPLIED</p>





**WITTENSTEIN**

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
xSemaphoreCreateRecursiveMutexStatic() ( )	xMutexCreate()	The SAFERTOS version requires parameters as detailed within this application note. FreeRTOS returns the handle of the created mutex, or NULL if the mutex could not be created; SAFERTOS returns pdPASS, errQUEUE_ALREADY_IN_USE, errINVALID_BYTE_ALIGNMENT or errNULL_PARAMETER_SUPPLIED
vSemaphoreDelete()	Not implemented.	
uxSemaphoreGetCount()	xSemaphoreGetCountDepth()	Whereas the FreeRTOS version returns the semaphore count, the SAFERTOS version passes the count out using a reference parameter. This is to permit the return value to provide more detailed error reporting. SAFERTOS returns pdPASS, errNULL_PARAMETER_SUPPLIED or errINVALID_QUEUE_HANDLE
xSemaphoreGetMutexHolder()	Not implemented.	
xSemaphoreGive()	xSemaphoreGive()	FreeRTOS returns either pdPASS or errQUEUE_FULL; SAFERTOS returns pdPASS, errSCHEDULER_IS_SUSPENDED, errINVALID_SEMAPHORE_HANDLE or errSEMAPHORE_ALREADY_GIVEN.  In FreeRTOS, xSemaphoreGive() can be used interchangeably when addressing semaphores or mutexes whereas SAFERTOS enforces the use of xSemaphoreGive() with semaphores only.
xSemaphoreGiveFromISR()	xSemaphoreGiveFromISR()	FreeRTOS returns either pdPASS or errQUEUE_FULL; SAFERTOS returns pdPASS, errNULL_PARAMETER_SUPPLIED, errINVALID_SEMAPHORE_HANDLE or errSEMAPHORE_ALREADY_GIVEN.
xSemaphoreGiveRecursive()	xMutexGive()	



**WITTENSTEIN**

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
xSemaphoreTake()	xSemaphoreTake()	FreeRTOS returns either pdPASS or errQUEUE_EMPTY; SAFERTOS returns pdPASS, errSCHEDULER_IS_SUSPENDED, errINVALID_SEMAPHORE_HANDLE, errSEMAPHORE_ALREADY_TAKEN.  In FreeRTOS, xSemaphoreTake() can be used interchangeably when addressing semaphores or mutexes whereas SAFERTOS enforces the use of xSemaphoreTake() with semaphores only
xSemaphoreTakeFromISR()	xSemaphoreTakeFromISR()	FreeRTOS returns either pdPASS or pdFAIL; SAFERTOS returns pdPASS, errNULL_PARAMETER_SUPPLIED, errINVALID_SEMAPHORE_HANDLE or errSEMAPHORE_ALREADY_TAKEN.
xSemaphoreTakeRecursive()	xMutexTake()	
Software Timer API		
xTimerChangePeriod()	xTimerChangePeriod()	FreeRTOS returns either pdPASS or pdFAIL; SAFERTOS returns pdPASS, errINVALID_PARAMETERS, errINVALID_TIMER_HANDLE or a return code from xQueueSend().
xTimerChangePeriodFromISR()	xTimerChangePeriodFromISR()	FreeRTOS returns either pdPASS or pdFAIL; SAFERTOS returns pdPASS, errINVALID_PARAMETERS, errINVALID_TIMER_HANDLE or a return code from xQueueSendFromISR().
xTimerCreate()	xTimerCreate()	FreeRTOS returns the timer handle or NULL; SAFERTOS returns pdPASS, errINVALID_PARAMETERS, errINVALID_TIMER_HANDLE, errNULL_PARAMETER_SUPPLIED, errTIMER_ALREADY_IN_USE, errINVALID_TIMER_TASK_INSTANCE or a return code from xTaskCreate().  SAFERTOS uses a structure to pass the initialisation parameters to xTimerCreate() rather than the list of individual parameters used in FreeRTOS.



**WITTENSTEIN**

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
xTimerCreateStatic()	xTimerCreate()	FreeRTOS returns the timer handle or NULL; SAFERTOS returns pdPASS, errINVALID_PARAMETERS, errINVALID_TIMER_HANDLE, errNULL_PARAMETER_SUPPLIED, errTIMER_ALREADY_IN_USE, errINVALID_TIMER_TASK_INSTANCE or a return code from xTaskCreate().  SAFERTOS uses a structure to pass the initialisation parameters to xTimerCreate() rather than the list of individual parameters used in FreeRTOS.
xTimerDelete()	xTimerDelete()	FreeRTOS returns pdPASS or pdFAIL; SAFERTOS returns pdPASS, errINVALID_TIMER_HANDLE or a return code from xQueueSend().
xTimerGetExpiryTime()	Not implemented.	
pcTimerGetName()	Not implemented.	For SAFERTOS, an arbitrary “timer local” storage object, present for each software timer, may be initialized when a timer is created, and is accessible via the pvTimerTLSObjectGet() function. This object is of type void pointer, and could be used to point to any arbitrary timer-specific data structure, so timer names may be implemented as part of such a structure, if required.
xTimerGetPeriod()	Not implemented.	
xTimerGetTimerDaemonTaskHandle()	Not implemented.	
pvTimerGetTimerID()	xTimerGetTimerID()	FreeRTOS returns the timer ID, SAFERTOS returns pdPASS, errNULL_PARAMETER_SUPPLIED or errINVALID_TIMER_HANDLE
xTimerIsTimerActive()	xTimerIsTimerActive()	FreeRTOS returns either pdPASS or pdFAIL; SAFERTOS returns pdTRUE, pdFALSE or errINVALID_TIMER_HANDLE
xTimerPendFunctionCall()	Not implemented.	
xTimerPendFunctionCallFromISR()	Not implemented.	



**WITTENSTEIN**

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
xTimerReset()	xTimerReset()	FreeRTOS returns pdPASS or pdFAIL; SAFERTOS returns pdPASS, errINVALID_TIMER_HANDLE or a return code from xQueueSend().
xTimerResetFromISR()	xTimerResetFromISR()	FreeRTOS returns pdPASS or pdFAIL; SAFERTOS returns pdPASS, errINVALID_TIMER_HANDLE or a return code from xQueueSendFromISR().
vTimerSetTimerID()	Not implemented.	In SAFERTOS, the timer ID is set when the timer is created. Arbitrary timer-specific storage may be implemented via the timer local storage object, accessible using pvTimerTLSObjectGet ()
xTimerStart()	xTimerStart()	FreeRTOS returns pdPASS or pdFAIL; SAFERTOS returns pdPASS, errINVALID_TIMER_HANDLE or a return code from xQueueSend().
xTimerStartFromISR()	xTimerStartFromISR()	FreeRTOS returns pdPASS or pdFAIL; SAFERTOS returns pdPASS, errINVALID_TIMER_HANDLE or a return code from xQueueSendFromISR().
xTimerStop()	xTimerStop()	FreeRTOS returns pdPASS or pdFAIL; SAFERTOS returns pdPASS, errINVALID_TIMER_HANDLE or a return code from xQueueSend().
xTimerStopFromISR()	xTimerStopFromISR()	FreeRTOS returns pdPASS or pdFAIL; SAFERTOS returns pdPASS, errINVALID_TIMER_HANDLE or a return code from xQueueSendFromISR().
Event Groups API		
xEventGroupClearBits()	xEventGroupClearBits()	FreeRTOS returns pdPASS or pdFALSE, SAFERTOS returns pdPASS, errNULL_PARAMETER_SUPPLIED, errINVALID_EVENT_GROUP_HANDLE or errINVALID_PARAMETERS



**WITTENSTEIN**

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
xEventGroupClearBitsFromISR()	xEventGroupClearBitsFromISR()	FreeRTOS returns pdPASS or pdFALSE, <b>SAFERTOS</b> returns pdPASS, errNULL_PARAMETER_SUPPLIED, errINVALID_EVENT_GROUP_HANDLE or errINVALID_PARAMETERS
xEventGroupCreate()	xEventGroupCreate()	The <b>SAFERTOS</b> version requires different parameters as detailed within this application note.  FreeRTOS returns an event group handle or NULL. <b>SAFERTOS</b> returns pdPASS, errNULL_PARAMETER_SUPPLIED or errINVALID_EVENT_GROUP_HANDLE.
xEventGroupCreateStatic()	xEventGroupCreate()	The <b>SAFERTOS</b> version requires different parameters as detailed within this application note.  FreeRTOS returns an event group handle or NULL. <b>SAFERTOS</b> returns pdPASS, errNULL_PARAMETER_SUPPLIED or errINVALID_EVENT_GROUP_HANDLE.
vEventGroupDelete()	xEventGroupDelete()	FreeRTOS returns no value (void function), <b>SAFERTOS</b> returns pdPASS or errNULL_PARAMETER_SUPPLIED.
xEventGroupGetBits()	xEventGroupGetBits()	FreeRTOS returns the event group bits that are set, or NULL. <b>SAFERTOS</b> returns pdPASS, errNULL_PARAMETER_SUPPLIED or errINVALID_EVENT_GROUP_HANDLE
xEventGroupGetBitsFromISR()	xEventGroupGetBitsFromISR()	FreeRTOS returns the event group bits that are set, or NULL. <b>SAFERTOS</b> returns pdPASS, errNULL_PARAMETER_SUPPLIED or errINVALID_EVENT_GROUP_HANDLE
xEventGroupSetBits()	xEventGroupSetBits()	FreeRTOS returns pdPASS or pdFALSE. <b>SAFERTOS</b> returns psPASS, errNULL_PARAMETER_SUPPLIED, errINVALID_PARAMETERS or errINVALID_EVENT_GROUP_HANDLE



**WITTENSTEIN**

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
xEventGroupSetBitsFromISR()	xEventGroupSetBitsFromISR()	FreeRTOS returns pdPASS or pdFALSE. SAFERTOS returns psPASS, errNULL_PARAMETER_SUPPLIED, errINVALID_PARAMETERS or errINVALID_EVENT_GROUP_HANDLE
xEventGroupSync()	Not implemented.	
xEventGroupWaitBits()	xEventGroupWaitBits()	FreeRTOS returns the value of the event group at the time the function returns. SAFERTOS returns pdPASS, errSCHEDULER_IS_SUSPENDED, errNULL_PARAMETER_SUPPLIED, errINVALID_PARAMETERS, errINVALID_EVENT_GROUP_HANDLE, errEVENT_GROUP_DELETED or errEVENT_GROUP_BITS_NOT_SET
Stream Buffer API		
xStreamBufferBytesAvailable()	Not implemented.	
xStreamBufferCreate()	Not implemented.	
xStreamBufferCreateStatic()	Not implemented.	
vStreamBufferDelete()	Not implemented.	
xStreamBufferIsEmpty()	Not implemented.	
xStreamBufferIsFull()	Not implemented.	
xStreamBufferReceive()	Not implemented.	
xStreamBufferReceiveFromISR()	Not implemented.	
xStreamBufferReset()	Not implemented.	
xStreamBufferSend()	Not implemented.	
xStreamBufferSendFromISR()	Not implemented.	
xStreamBufferSetTriggerLevel()	Not implemented.	
xStreamBufferSpacesAvailable()	Not implemented.	



**WITTENSTEIN**

Table 4-1 FreeRTOS to SAFERTOS function name mapping

FreeRTOS Function	SAFERTOS Equivalent	Notes
Message Buffer API		
xMessageBufferCreate()	Not implemented.	
xMessageBufferCreateStatic()	Not implemented.	
vMessageBufferDelete()	Not implemented.	
xMessageBufferIsEmpty()	Not implemented.	
xMessageBufferIsFull()	Not implemented.	
xMessageBufferReceive()	Not implemented.	
xMessageBufferReceiveFromISR()	Not implemented.	
xMessageBufferReset()	Not implemented.	
xMessageBufferSend()	Not implemented.	
xMessageBufferSendFromISR()	Not implemented.	
xMessageBufferSpacesAvailable()	Not implemented.	



**WITTENSTEIN**

# **CHAPTER 5**

# **ERROR REPORTING**

---





## 5.1 SIDE EFFECTS OF INCREASED ERROR REPORTING

---

SAFERTOS functions generally return an error code or pdPASS, and those that need to have an extra reference parameter in which to return data.

Care must be taken that application code is updated to correctly interpret the increased number of function return values. For example, under FreeRTOS the code depicted by Listing 1 is valid because xQueueSend() only returns one of two values indicating either success or failure. Under SAFERTOS the code must be changed to that depicted by Listing 2 as only a value of pdPASS would indicate success – whereas one of a number of values are used to indicate the cause of failure.

---

```
if( xQueueSend( xQueue, &ucItemToSend, 0 ) == pdFAIL )
{
    /* Could not send to the queue. */
}
```

---

Listing 1 Checking a function return value for an error under FreeRTOS

---

```
if( xQueueSend( xQueue, &ucItemToSend, 0 ) != pdPASS )
{
    /* Could not send to the queue. */
}
```

---

Listing 2 Checking a function return value for an error under SAFERTOS



**WITTENSTEIN**

# **CHAPTER 6**

# **MISCELLANEOUS**

---



## 6.1 HEADER FILES

---

SAFERTOS uses SafeRTOS.h and SafeRTOSConfig.h in place of FreeRTOS.h and FreeRTOSConfig.h

In order for an application to call a SAFERTOS API function, the source file simply needs to include the SafeRTOS\_API.h header file – so long as the compiler can locate the other SAFERTOS header files, the required header files will be automatically included.

## 6.2 HOOK FUNCTIONS

---

The host application (the application that uses SAFERTOS) is required to provide an "Error Hook" (or callback) function. In addition, the host application can optionally supply an "Idle Hook", "Task Delete Hook", "SVC Hook", "Tick Hook" and a function to set up the tick interrupt timer.

SAFERTOS calls the Error Hook function upon the detection of a fatal error – either a corruption within the scheduler data structures or a potential stack overflow while performing a context switch. The purpose of the Error Hook function is to allow the host application to place the system into a 'safe state'.

Pointers to the hook functions are passed to SAFERTOS as part of the xPORT\_INIT\_PARAMETERS structure referenced in the call to xTaskInitializeScheduler(). Refer to the SAFERTOS Product Variant User Manual [Reference 1] for a full description, and SafeRTOSConfig.c of the accompanying demonstration application for further information.

## 6.3 FILE NAMES

---

The FreeRTOS file tasks.c is called task.c under SAFERTOS.



**WITTENSTEIN**

## **CONTACT INFORMATION**

User feedback is essential to the continued maintenance and development of **SAFERTOS**. Please provide all software and documentation comments and suggestions to the most convenient contact point listed below.

Address:	WITTENSTEIN high integrity systems Brown's Court, Long Ashton Business Park Yanley Lane, Long Ashton Bristol, BS41 9LB England
Phone:	+44 (0)1275 395 600
Fax:	+44 (0)1275 393 630
Email:	<a href="mailto:support@HighIntegritySystems.com">support@HighIntegritySystems.com</a>
Website	<a href="http://www.HighIntegritySystems.com">www.HighIntegritySystems.com</a>

All Trademarks acknowledged.