

# **Mastering the FreeRTOS™ Real Time Kernel**

**A Hands-On Tutorial Guide  
(Table Of Contents Only)**

**Richard Barry**

Pre-release 15114, prior to formal publication.

All text, source code, and diagrams are the exclusive property of Real Time Engineers Ltd. unless otherwise noted inline. Distribution, use in presentations, use in commercial training, or publication in any form is strictly prohibited without prior written authority from Real Time Engineers Ltd.

© Real Time Engineers Ltd. 2105. All rights reserved.

FreeRTOS™, FreeRTOS.org™ and the FreeRTOS logo are trademarks of Real Time Engineers Ltd.

OPENRTOS® and SAFERTOS® are trademarks of WITTENSTEIN Aerospace and Simulation Ltd.

All other brands or product names are the property of their respective holders.

<http://www.FreeRTOS.org>  
<http://www.FreeRTOS.org/plus>  
<http://www.FreeRTOS.org/labs>

ISBN TBD

# Contents

---

Contents .....	ix
List of Figures .....	xvi
List of Code Listings .....	xix
List of Tables .....	xxiii
List of Notation.....	xxvi
<b>Preface</b> .....	1
Multitasking in Small Embedded Systems .....	2
About FreeRTOS .....	2
Value Proposition.....	3
A Note About Terminology .....	3
Why Use a Real-time Kernel? .....	3
FreeRTOS Features .....	5
Licensing, and The FreeRTOS, OpenRTOS, and SafeRTOS Family .....	6
Included Source Files and Projects .....	9
Obtaining the Examples that Accompany this Book .....	9
<b>Chapter 1</b> The FreeRTOS Distribution .....	11
1.1 Chapter Introduction and Scope.....	12
Scope .....	12
1.2 Understanding the FreeRTOS Distribution .....	13
Definition: FreeRTOS Port .....	13
Building FreeRTOS.....	13
FreeRTOSConfig.h .....	13
The Official FreeRTOS Distribution .....	14
The Top Directories in the FreeRTOS Distribution .....	14
FreeRTOS Source Files Common to All Ports .....	14
FreeRTOS Source Files Specific to a Port .....	16
Header Files .....	17
1.3 Demo Applications .....	18
1.4 Creating a FreeRTOS Project .....	20
Adapting One of the Supplied Demo Projects .....	20
Creating a New Project from Scratch .....	21
1.5 Data Types and Coding Style Guide .....	23
Data Types .....	23
Variable Names .....	24
Function Names.....	24
Formatting.....	25

Macro Names.....	25
Rationale for Excessive Type Casting .....	26
<b>Chapter 2</b> Heap Memory Management.....	27
2.1 Chapter Introduction and Scope .....	28
Prerequisites .....	28
Dynamic Memory Allocation and its Relevance to FreeRTOS.....	28
Options for Dynamic Memory Allocation.....	29
Scope.....	30
2.2 Example Memory Allocation Schemes .....	31
Heap_1 .....	31
Heap_2 .....	32
Heap_3 .....	34
Heap_4 .....	34
Setting a Start Address for the Array Used By Heap_4 .....	36
Heap_5 .....	37
The vPortDefineHeapRegions() API Function .....	38
2.3 Heap Related Utility Functions .....	43
The xPortGetFreeHeapSize() API Function.....	43
The xPortGetMinimumEverFreeHeapSize() API Function .....	43
Malloc Failed Hook Functions .....	44
<b>Chapter 3</b> Task Management .....	46
3.1 Chapter Introduction and Scope .....	47
Scope.....	47
3.2 Task Functions.....	48
3.3 Top Level Task States.....	49
3.4 Creating Tasks .....	50
The xTaskCreate() API Function .....	50
Example 1. Creating tasks .....	52
Example 2. Using the task parameter.....	56
3.5 Task Priorities .....	59
3.6 Time Measurement and the Tick Interrupt .....	61
Example 3. Experimenting with priorities .....	63
3.7 Expanding the 'Not Running' State .....	65
The Blocked State.....	65
The Suspended State.....	66
The Ready State .....	66
Completing the State Transition Diagram .....	66
Example 4. Using the Blocked state to create a delay .....	67
The vTaskDelayUntil() API Function.....	71
Example 5. Converting the example tasks to use vTaskDelayUntil() .....	72
Example 6. Combining blocking and non-blocking tasks .....	73
3.8 The Idle Task and the Idle Task Hook .....	76

Idle Task Hook Functions.....	76
Limitations on the Implementation of Idle Task Hook Functions.....	77
Example 7. Defining an idle task hook function.....	77
3.9 Changing the Priority of a Task.....	80
The vTaskPrioritySet() API Function.....	80
The uxTaskPriorityGet() API Function.....	80
Example 8. Changing task priorities.....	81
3.10 Deleting a Task.....	86
The vTaskDelete() API Function.....	86
Example 9. Deleting tasks.....	87
3.11 Thread Local Storage.....	90
3.12 Scheduling Algorithms.....	91
A Recap of Task States and Events.....	91
Configuring the Scheduling Algorithm.....	91
Prioritized Pre-emptive Scheduling with Time Slicing.....	92
Prioritized Pre-emptive Scheduling (without Time Slicing).....	96
Co-operative Scheduling.....	98
<b>Chapter 4 Queue Management.....</b>	<b>102</b>
4.1 Chapter Introduction and Scope.....	103
Scope.....	103
4.2 Characteristics of a Queue.....	104
Data Storage.....	104
Access by Multiple Tasks.....	107
Blocking on Queue Reads.....	107
Blocking on Queue Writes.....	107
Blocking on Multiple Queues.....	108
4.3 Using a Queue.....	109
The xQueueCreate() API Function.....	109
The xQueueSendToBack() and xQueueSendToFront() API Functions.....	110
The xQueueReceive() API Function.....	112
The uxQueueMessagesWaiting() API Function.....	114
Example 10. Blocking when receiving from a queue.....	115
4.4 Receiving Data From Multiple Sources.....	120
Example 11. Blocking when sending to a queue, and sending structures on a queue..	121
4.5 Working with Large or Variable Sized Data.....	127
Queuing Pointers.....	127
Using a Queue to Send Different Types and Lengths of Data.....	129
4.6 Receiving From Multiple Queues.....	132
Queue Sets.....	132
The xQueueCreateSet() API Function.....	133
The xQueueAddToSet() API Function.....	135
The xQueueSelectFromSet() API Function.....	136
Example 12. Using a Queue Set.....	138

More Realistic Queue Set Use Cases .....	142
4.7 Using a Queue to Create a Mailbox.....	144
The xQueueOverwrite() API Function.....	145
The xQueuePeek() API Function.....	146
<b>Chapter 5</b> Software Timer Management.....	148
5.1 Chapter Introduction and Scope .....	149
Scope.....	149
5.2 Software Timer Callback Functions .....	150
5.3 Attributes and States of a Software Timer .....	151
Period of a Software Timer.....	151
One-shot and Auto-reload Timers .....	151
Software Timer States.....	152
5.4 The Context of a Software Timer.....	154
The RTOS Daemon (Timer Service) Task.....	154
The Timer Command Queue.....	154
Daemon Task Scheduling .....	155
5.5 Creating and Starting a Software Timer.....	159
The xTimerCreate() API Function.....	159
The xTimerStart() API Function.....	160
Example 13. Creating one-shot and auto-reload timers.....	164
5.6 The Timer ID .....	167
The vTimerSetTimerID() API Function .....	167
The pvTimerGetTimerID() API Function .....	167
Example 14. Using the callback function parameter and the software timer ID.....	168
5.7 Changing the Period of a Timer.....	171
The xTimerChangePeriod() API Function.....	171
5.8 Resetting a Software Timer .....	175
The xTimerReset() API Function .....	175
Example 15. Resetting a software timer .....	177
<b>Chapter 6</b> Interrupt Management.....	182
6.1 Chapter Introduction and Scope .....	183
Events.....	183
Scope.....	184
6.2 Using the FreeRTOS API from an ISR .....	185
The Interrupt Safe API.....	185
The Benefits of Using a Separate Interrupt Safe API.....	185
The Disadvantages of Using a Separate Interrupt Safe API .....	186
The xHigherPriorityTaskWoken Parameter .....	186
The portYIELD_FROM_ISR() and portEND_SWITCHING_ISR() Macros.....	188
6.3 Deferred Interrupt Processing.....	190
6.4 Binary Semaphores Used for Synchronization .....	192
The xSemaphoreCreateBinary() API Function.....	194

The xSemaphoreTake() API Function .....	195
The xSemaphoreGiveFromISR() API Function .....	197
Example 16. Using a binary semaphore to synchronize a task with an interrupt .....	199
Improving the Implementation of the Task Used in Example 16.....	203
6.5 Counting Semaphores .....	209
The xSemaphoreCreateCounting() API Function .....	211
Example 17. Using a counting semaphore to synchronize a task with an interrupt.....	212
6.6 Deferring Work to the RTOS Daemon Task .....	214
The xTimerPendFunctionCallFromISR() API Function .....	215
Example 18. Centralized deferred interrupt processing .....	217
6.7 Using Queues within an Interrupt Service Routine .....	221
The xQueueSendToFrontFromISR() and xQueueSendToBackFromISR() API Functions .....	221
Considerations When Using a Queue From an ISR .....	223
Example 19. Sending and receiving on a queue from within an interrupt .....	223
6.8 Interrupt Nesting .....	229
A Note to ARM Cortex-M and ARM GIC Users .....	231
<b>Chapter 7</b> Resource Management .....	234
7.1 Chapter Introduction and Scope.....	235
Mutual Exclusion.....	237
Scope .....	238
7.2 Critical Sections and Suspending the Scheduler .....	239
Basic Critical Sections .....	239
Suspending (or Locking) the Scheduler .....	241
The vTaskSuspendAll() API Function.....	242
The xTaskResumeAll() API Function .....	242
7.3 Mutexes (and Binary Semaphores).....	244
The xSemaphoreCreateMutex() API Function.....	246
Example 20. Rewriting vPrintString() to use a semaphore .....	246
Priority Inversion .....	250
Priority Inheritance .....	251
Deadlock (or Deadly Embrace) .....	252
Recursive Mutexes .....	253
Mutexes and Task Scheduling .....	256
7.4 Gatekeeper Tasks.....	260
Example 21. Re-writing vPrintString() to use a gatekeeper task.....	260
<b>Chapter 8</b> Event Groups.....	266
8.1 Chapter Introduction and Scope.....	267
Scope .....	267
8.2 Characteristics of an Event Group.....	269
Event Groups, Event Flags and Event Bits.....	269
More About the EventBits_t Data Type .....	270

Access by Multiple Tasks .....	270
A Practical Example of Using an Event Group .....	270
8.3 Event Management Using Event Groups.....	272
The xEventGroupCreate() API Function .....	272
The xEventGroupSetBits() API Function .....	272
The xEventGroupSetBitsFromISR() API Function .....	273
The xEventGroupWaitBits() API Function.....	276
Example 22. Experimenting with event groups.....	280
8.4 Task Synchronization Using an Event Group .....	286
The xEventGroupSync() API Function.....	288
Example 23. Synchronizing tasks.....	290
<b>Chapter 9</b> Task Notifications.....	294
9.1 Chapter Introduction and Scope .....	295
Communicating Through Intermediary Objects.....	295
Task Notifications—Direct to Task Communication .....	295
Scope.....	296
9.2 Task Notifications; Benefits and Limitations.....	297
Performance Benefits of Task Notifications .....	297
RAM Footprint Benefits of Task Notifications .....	297
Limitations of Task Notifications .....	297
9.3 Using Task Notifications .....	299
Task Notification API Options.....	299
The xTaskNotifyGive() API Function .....	299
The vTaskNotifyGiveFromISR() API Function .....	300
The ulTaskNotifyTake() API Function .....	301
Example 24. Using a task notification in place of a semaphore, method 1.....	303
Example 25. Using a task notification in place of a semaphore, method 2.....	306
The xTaskNotify() and xTaskNotifyFromISR() API Functions .....	308
The xTaskNotifyWait() API Function.....	311
Task Notifications Used in Peripheral Device Drivers: UART Example.....	314
Task Notifications Used in Peripheral Device Drivers: ADC Example .....	321
Task Notifications Used Directly Within an Application .....	323
<b>Chapter 10</b> Low Power Support.....	328
<b>Chapter 11</b> Developer Support .....	329
11.1 Chapter Introduction and Scope .....	330
11.2 configASSERT() .....	331
Example configASSERT() definitions .....	331
11.3 FreeRTOS+Trace.....	333
11.4 Debug Related Hook (Callback) Functions .....	337
Malloc failed hook .....	337
11.5 Viewing Run-time and Task State Information.....	338



Task Run-Time Statistics .....	338
The Run-Time Statistics Clock .....	338
Configuring an Application to Collect Run-Time Statistics .....	339
The uxTaskGetSystemState() API Function .....	340
The vTaskList() Helper Function .....	343
The vTaskGetRunTimeStats() Helper Function .....	345
Generating and Displaying Run-Time Statistics, a Worked Example .....	346
11.6 Trace Hook Macros .....	349
Available Trace Hook Macros .....	349
Defining Trace Hook Macros .....	353
FreeRTOS Aware Debugger Plug-ins .....	354
<b>Chapter 12</b> Trouble Shooting .....	356
12.1 Chapter Introduction and Scope .....	357
12.2 Interrupt Priorities .....	358
12.3 Stack Overflow .....	360
The uxTaskGetStackHighWaterMark() API Function .....	360
Run Time Stack Checking—Overview .....	361
Run Time Stack Checking—Method 1 .....	361
Run Time Stack Checking—Method 2 .....	362
12.4 Inappropriate Use of printf() and sprintf() .....	363
Printf-stdarg.c .....	363
12.5 Other Common Sources of Error .....	365
Symptom: Adding a simple task to a demo causes the demo to crash .....	365
Symptom: Using an API function within an interrupt causes the application to crash ...	365
Symptom: Sometimes the application crashes within an interrupt service routine .....	365
Symptom: The scheduler crashes when attempting to start the first task .....	366
Symptom: Interrupts are unexpectedly left disabled, or critical sections do not nest correctly .....	366
Symptom: The application crashes even before the scheduler is started .....	366
Symptom: Calling API functions while the scheduler is suspended, or from inside a critical section, causes the application to crash .....	367
INDEX .....	369

## List of Figures

Figure 1. Top level directories within the FreeRTOS distribution .....	14
Figure 2. Core FreeRTOS source files within the FreeRTOS directory tree.....	15
Figure 3. Port specific source files within the FreeRTOS directory tree .....	16
Figure 4. The demo directory hierarchy.....	19
Figure 5. RAM being allocated from the heap_1 array each time a task is created .....	32
Figure 6. RAM being allocated and freed from the heap_2 array as tasks are created and deleted.....	33
Figure 7. RAM being allocated and freed from the heap_4 array .....	35
Figure 8 Memory Map .....	39
Figure 9. Top level task states and transitions.....	49
Figure 10. The output produced when Example 1 is executed .....	54
Figure 11. The actual execution pattern of the two Example 1 tasks .....	55
Figure 12. The execution sequence expanded to show the tick interrupt executing .....	62
Figure 13. Running both tasks at different priorities .....	64
Figure 14. The execution pattern when one task has a higher priority than the other .....	64
Figure 15. Full task state machine.....	67
Figure 16. The output produced when Example 4 is executed .....	69
Figure 17. The execution sequence when the tasks use vTaskDelay() in place of the NULL loop.....	70
Figure 18. Bold lines indicate the state transitions performed by the tasks in Example 4 .....	71
Figure 19. The output produced when Example 6 is executed .....	75
Figure 20. The execution pattern of Example 6.....	75
Figure 21. The output produced when Example 7 is executed .....	79
Figure 22. The sequence of task execution when running Example 8.....	84
Figure 23. The output produced when Example 8 is executed .....	85
Figure 24. The output produced when Example 9 is executed .....	88
Figure 25. The execution sequence for example 9.....	89
Figure 26. Execution pattern highlighting task prioritization and pre-emption in a hypothetical application in which each task has been assigned a unique priority.....	93
Figure 27 Execution pattern highlighting task prioritization and time slicing in a hypothetical application in which two tasks run at the same priority .....	95
Figure 28 The execution pattern for the same scenario as shown in Figure 27, but this time with configIDLE_SHOULD_YIELD set to 0.....	96
Figure 29 Execution pattern that demonstrates how tasks of equal priority can receive hugely different amounts of processing time when time slicing is not used .....	97
Figure 30 Execution pattern demonstrating the behavior of the co-operative scheduler .....	99
Figure 31. An example sequence of writes to, and reads from a queue .....	105
Figure 32. The output produced when Example 10 is executed .....	119
Figure 33. The sequence of execution produced by Example 10 .....	119
Figure 34. An example scenario where structures are sent on a queue .....	120
Figure 35 The output produced by Example 11 .....	124

Figure 36. The sequence of execution produced by Example 11 .....	125
Figure 37 The output produced when Example 12 is executed.....	142
Figure 38 The difference in behavior between one-shot and auto-reload software timers.....	151
Figure 39 Auto-reload software timer states and transitions.....	153
Figure 40 One-shot software timer states and transitions .....	153
Figure 41 The timer command queue being used by a software timer API function to communicate with the RTOS daemon task.....	155
Figure 42 The execution pattern when the priority of a task calling xTimerStart() is above the priority of the daemon task.....	155
Figure 43 The execution pattern when the priority of a task calling xTimerStart() is below the priority of the daemon task.....	157
Figure 44 The output produced when Example 13 is executed.....	166
Figure 45 The output produced when Example 14 is executed.....	170
Figure 46 Starting and resetting a software timer that has a period of 6 ticks.....	175
Figure 47 The output produced when Example 15 is executed.....	180
Figure 48 Completing interrupt processing in a high priority task .....	191
Figure 49. Using a binary semaphore to implement deferred interrupt processing.....	192
Figure 50. Using a binary semaphore to synchronize a task with an interrupt .....	194
Figure 51. The output produced when Example 16 is executed.....	202
Figure 52. The sequence of execution when Example 16 is executed.....	203
Figure 53. The scenario when one interrupt occurs before the task has finished processing the first event.....	205
Figure 54 The scenario when two interrupts occur before the task has finished processing the first event.....	206
Figure 55. Using a counting semaphore to 'count' events .....	210
Figure 56. The output produced when Example 17 is executed.....	213
Figure 57. The output produced when Example 18 is executed.....	219
Figure 58 The sequence of execution when Example 18 is executed.....	220
Figure 59. The output produced when Example 19 is executed.....	227
Figure 60. The sequence of execution produced by Example 19.....	228
Figure 61. Constants affecting interrupt nesting behavior .....	231
Figure 62 How a priority of binary 101 is stored by a Cortex-M microcontroller that implements four priority bits.....	232
Figure 63. Mutual exclusion implemented using a mutex.....	245
Figure 64. The output produced when Example 20 is executed.....	249
Figure 65. A possible sequence of execution for Example 20 .....	250
Figure 66. A worst case priority inversion scenario .....	251
Figure 67. Priority inheritance minimizing the effect of priority inversion .....	252
Figure 68 A possible sequence of execution when tasks that have the same priority use the same mutex.....	256
Figure 69 A sequence of execution that could occur if two instances of the task shown by Listing 125 are created at the same priority .....	258
Figure 70. The output produced when Example 21 is executed.....	265
Figure 71 Event flag to bit number mapping in a variable of type EventBits_t .....	269

Figure 72 An event group in which only bits 1, 4 and 7 are set, and all the other event flags are clear, making the event group's value 0x92.....	269
Figure 73 The output produced when Example 22 is executed with xWaitForAllBits set to pdFALSE .....	284
Figure 74 The output produced when Example 22 is executed with xWaitForAllBits set to pdTRUE.....	285
Figure 75 The output produced when Example 23 is executed .....	293
Figure 76 A communication object being used to send an event from one task to another....	295
Figure 77 A task notification used to send an event directly from one task to another .....	296
Figure 78. The output produced when Example 16 is executed .....	305
Figure 79. The sequence of execution when Example 24 is executed .....	306
Figure 80. The output produced when Example 25 is executed .....	308
Figure 81 The communication paths from the application tasks to the cloud server, and back again .....	324
Figure 82 FreeRTOS+Trace includes more than 20 interconnected views .....	333
Figure 83 FreeRTOS+Trace main trace view - one of more than 20 interconnected trace views .....	334
Figure 84 FreeRTOS+Trace CPU load view - one of more than 20 interconnected trace views .....	335
Figure 85 FreeRTOS+Trace response time view - one of more than 20 interconnected trace views.....	335
Figure 86 FreeRTOS+Trace user event plot view - one of more than 20 interconnected trace views.....	336
Figure 87 FreeRTOS+Trace kernel object history view - one of more than 20 interconnected trace views.....	336
Figure 88 Example output generated by vTaskList() .....	345
Figure 89 Example output generated by vTaskGetRunTimeStats() .....	346
Figure 90 FreeRTOS ThreadSpy Eclipse plug-in from Code Confidence Ltd. ....	354

## List of Code Listings

Listing 1. The template for a new main() function.....	20
Listing 2. Using GCC syntax to declare the array that will be used by heap_4, and place the array in a memory section named .my_heap .....	37
Listing 3. Using IAR syntax to declare the array that will be used by heap_4, and place the array at the absolute address 0x20000000 .....	37
Listing 4. The vPortDefineHeapRegions() API function prototype .....	38
Listing 5. The HeapRegion_t structure.....	38
Listing 6. An array of HeapRegion_t structures that together describe the 3 regions of RAM in their entirety .....	40
Listing 7. An array of HeapRegion_t structures that describe all of RAM2, all of RAM3, but only part of RAM1 .....	41
Listing 8. The xPortGetFreeHeapSize() API function prototype.....	43
Listing 9. The xPortGetMinimumEverFreeHeapSize() API function prototype .....	43
Listing 10. The malloc failed hook function name and prototype. ....	44
Listing 11. The task function prototype.....	48
Listing 12. The structure of a typical task function.....	48
Listing 13. The xTaskCreate() API function prototype .....	50
Listing 14. Implementation of the first task used in Example 1 .....	53
Listing 15. Implementation of the second task used in Example 1 .....	53
Listing 16. Starting the Example 1 tasks .....	54
Listing 17. Creating a task from within another task after the scheduler has started .....	56
Listing 18. The single task function used to create two tasks in Example 2.....	57
Listing 19. The main() function for Example 2. ....	58
Listing 20. Using the pdMS_TO_TICKS() macro to convert 200 milliseconds into an equivalent time in tick periods.....	62
Listing 21. Creating two tasks at different priorities .....	63
Listing 22. The vTaskDelay() API function prototype.....	68
Listing 23. The source code for the example task after the null loop delay has been replaced by a call to vTaskDelay() .....	69
Listing 24. vTaskDelayUntil() API function prototype.....	72
Listing 25. The implementation of the example task using vTaskDelayUntil() .....	73
Listing 26. The continuous processing task used in Example 6.....	74
Listing 27. The periodic task used in Example 6 .....	74
Listing 28. The idle task hook function name and prototype.....	77
Listing 29. A very simple Idle hook function .....	78
Listing 30. The source code for the example task now prints out the ullIdleCycleCount value.....	78
Listing 31. The vTaskPrioritySet() API function prototype .....	80
Listing 32. The uxTaskPriorityGet() API function prototype .....	80
Listing 33. The implementation of Task 1 in Example 8 .....	82
Listing 34. The implementation of Task 2 in Example 8 .....	83
Listing 35. The implementation of main() for Example 8.....	84

Listing 36. The vTaskDelete() API function prototype.....	86
Listing 37. The implementation of main() for Example 9.....	87
Listing 38. The implementation of Task 1 for Example 9.....	88
Listing 39. The implementation of Task 2 for Example 9.....	88
Listing 40. The xQueueCreate() API function prototype.....	109
Listing 41. The xQueueSendToFront() API function prototype.....	110
Listing 42. The xQueueSendToBack() API function prototype.....	110
Listing 43. The xQueueReceive() API function prototype.....	113
Listing 44. The uxQueueMessagesWaiting() API function prototype.....	114
Listing 45. Implementation of the sending task used in Example 10.....	116
Listing 46. Implementation of the receiver task for Example 10.....	117
Listing 47. The implementation of main() in Example 10.....	118
Listing 48. The definition of the structure that is to be passed on a queue, plus the declaration of two variables for use by the example.....	121
Listing 49. The implementation of the sending task for Example 11.....	122
Listing 50. The definition of the receiving task for Example 11.....	123
Listing 51. The implementation of main() for Example 11.....	124
Listing 52. Creating a queue that holds pointers.....	128
Listing 53. Using a queue to send a pointer to a buffer.....	128
Listing 54. Using a queue to receive a pointer to a buffer.....	128
Listing 55. The structure used to send events to the TCP/IP stack task in FreeRTOS+TCP.....	129
Listing 56. Pseudo code showing how an IPStackEvent_t structure is used to send data received from the network to the TCP/IP task.....	130
Listing 57. Pseudo code showing how an IPStackEvent_t structure is used to send the handle of a socket that is accepting a connection to the TCP/IP task.....	130
Listing 58. Pseudo code showing how an IPStackEvent_t structure is used to send a network down event to the TCP/IP task.....	131
Listing 59. Pseudo code showing how an IPStackEvent_t structure is used to send a network down to the TCP/IP task.....	131
Listing 60. The xQueueCreateSet() API function prototype.....	133
Listing 61. The xQueueAddToSet() API function prototype.....	135
Listing 62. The xQueueSelectFromSet() API function prototype.....	136
Listing 63. Implementation of main() for Example 12.....	139
Listing 64. The sending tasks used in Example 12.....	140
Listing 65. The receive task used in Example 12.....	141
Listing 66. Using a queue set that contains queues and semaphores.....	143
Listing 67. A queue being created for use as a mailbox.....	145
Listing 68. The xQueueOverwrite() API function prototype.....	145
Listing 69. Using the xQueueOverwrite() API function.....	146
Listing 70. The xQueuePeek() API function prototype.....	147
Listing 71. Using the xQueuePeek() API function.....	147
Listing 72. The software timer callback function prototype.....	150
Listing 73. The xTimerCreate() API function prototype.....	159

Listing 74. The xTimerStart() API function prototype .....	161
Listing 75. Creating and starting the timers used in Example 13 .....	164
Listing 76. The callback function used by the one-shot timer in Example 13 .....	165
Listing 77. The callback function used by the auto-reload timer in Example 13 .....	165
Listing 78. The vTimerSetTimerID() API function prototype .....	167
Listing 79. The pvTimerGetTimerID() API function prototype .....	167
Listing 80. Creating the timers used in Example 14 .....	168
Listing 81. The timer callback function used in Example 14 .....	169
Listing 82. The xTimerChangePeriod() API function prototype .....	171
Listing 83. Using xTimerChangePeriod() .....	174
Listing 84. The xTimerReset() API function prototype .....	176
Listing 85. The callback function for the one-shot timer used in Example 15 .....	178
Listing 86. The task used to reset the software timer in Example 15 .....	179
Listing 87. The portEND_SWITCHING_ISR() macros .....	189
Listing 88. The portYIELD_FROM_ISR() macros .....	189
Listing 89. The xSemaphoreCreateBinary() API function prototype .....	195
Listing 90. The xSemaphoreTake() API function prototype .....	195
Listing 91. The xSemaphoreGiveFromISR() API function prototype .....	197
Listing 92. Implementation of the task that periodically generates a software interrupt in Example 16 .....	199
Listing 93. The implementation of the task to which the interrupt processing is deferred (the task that synchronizes with the interrupt) in Example 16 .....	200
Listing 94. The ISR for the software interrupt used in Example 16 .....	201
Listing 95. The implementation of main() for Example 16 .....	202
Listing 96. The recommended structure of a deferred interrupt processing task, using a UART receive handler as an example .....	208
Listing 97. The xSemaphoreCreateCounting() API function prototype .....	211
Listing 98. The call to xSemaphoreCreateCounting() used to create the counting semaphore in Example 17 .....	212
Listing 99. The implementation of the interrupt service routine used by Example 17 .....	213
Listing 100. The xTimerPendFunctionCallFromISR() API function prototype .....	215
Listing 101. The prototype to which a function passed in the xFunctionToPend parameter of xTimerPendFunctionCallFromISR() must conform .....	215
Listing 102. The software interrupt handler used in Example 18 .....	218
Listing 103. The function that performs the processing necessitated by the interrupt in Example 18. ....	218
Listing 104. The implementation of main() for Example 18 .....	219
Listing 105. The xQueueSendToFrontFromISR() API function prototype .....	221
Listing 106. The xQueueSendToBackFromISR() API function prototype .....	221
Listing 107. The implementation of the task that writes to the queue in Example 19 .....	224
Listing 108. The implementation of the interrupt service routine used by Example 19 .....	225
Listing 109. The task that prints out the strings received from the interrupt service routine in Example 19 .....	226
Listing 110. The main() function for Example 19 .....	227
Listing 111. An example read, modify, write sequence .....	235

Listing 112. An example of a reentrant function.....	237
Listing 113. An example of a function that is not reentrant .....	237
Listing 114. Using a critical section to guard access to a register .....	239
Listing 115. A possible implementation of vPrintString().....	240
Listing 116. Using a critical section in an interrupt service routine .....	241
Listing 117. The vTaskSuspendAll() API function prototype .....	242
Listing 118. The xTaskResumeAll() API function prototype .....	242
Listing 119. The implementation of vPrintString() .....	243
Listing 120. The xSemaphoreCreateMutex() API function prototype .....	246
Listing 121. The implementation of prvNewPrintString().....	247
Listing 122. The implementation of prvPrintTask() for Example 20.....	248
Listing 123. The implementation of main() for Example 20 .....	249
Listing 124. Creating and using a recursive mutex .....	255
Listing 125. A task that uses a mutex in a tight loop .....	257
Listing 126. Ensuring tasks that use a mutex in a loop receive a more equal amount of processing time, while also ensuring processing time is not wasted by switching between tasks too rapidly .....	259
Listing 127. The name and prototype for a tick hook function.....	261
Listing 128. The gatekeeper task .....	261
Listing 129. The print task implementation for Example 21 .....	262
Listing 130. The tick hook implementation.....	263
Listing 131. The implementation of main() for Example 21 .....	264
Listing 132. The xEventGroupCreate() API function prototype .....	272
Listing 133. The xEventGroupSetBits() API function prototype.....	273
Listing 134. The xEventGroupSetBitsFromISR() API function prototype.....	274
Listing 135. The xEventGroupWaitBits() API function prototype.....	276
Listing 136. Event bit definitions used in Example 22 .....	280
Listing 137. The task that sets two bits in the event group in Example 22 .....	281
Listing 138. The ISR that sets bit 2 in the event group in Example 22.....	282
Listing 139. The task that blocks to wait for event bits to become set in Example 22 .....	283
Listing 140. Creating the event group and tasks in Example 22 .....	284
Listing 141. Pseudo code for two tasks that synchronize with each other to ensure a shared TCP socket is no longer in use by either task before the socket is closed .....	287
Listing 142. The xEventGroupSync() API function prototype .....	289
Listing 143. The implementation of the task used in Example 23 .....	291
Listing 144. The main() function used in Example 23 .....	292
Listing 145. The xTaskNotifyGive() API function prototype.....	299
Listing 146. The vTaskNotifyGiveFromISR() API function prototype.....	300
Listing 147. The ulTaskNotifyTake() API function prototype .....	301
Listing 148. The implementation of the task to which the interrupt processing is deferred (the task that synchronizes with the interrupt) in Example 24.....	304
Listing 149. The implementation of the interrupt service routine used in Example 24.....	305



Listing 150. The implementation of the task to which the interrupt processing is deferred (the task that synchronizes with the interrupt) in Example 25.....	307
Listing 151. The implementation of the interrupt service routine used in Example 25.....	307
Listing 152. Prototypes for the xTaskNotify() and xTaskNotifyFromISR() API functions .....	309
Listing 153. The xTaskNotifyWait() API function prototype.....	311
Listing 154. Pseudo code demonstrating how a binary semaphore can be used in a driver library transmit function.....	316
Listing 155. Pseudo code demonstrating how a task notification can be used in a driver library transmit function.....	318
Listing 156. Pseudo code demonstrating how a task notification can be used in a driver library receive function.....	320
Listing 157. Pseudo code demonstrating how a task notification can be used to pass a value to a task .....	322
Listing 158. The structure and data type sent on a queue to the server task.....	324
Listing 159. The Implementation of the Cloud Read API Function .....	325
Listing 160. The Server Task Processing a Read Request .....	325
Listing 161. The Implementation of the Cloud Write API Function.....	326
Listing 162. The Server Task Processing a Send Request .....	327
Listing 163 Using the standard C assert() macro to check pxMyPointer is not NULL .....	331
Listing 164 A simple configASSERT() definition useful when executing under the control of a debugger .....	332
Listing 165 A configASSERT() definition that records the source code line that failed an assertion.....	332
Listing 166. The uxTaskGetSystemState() API function prototype .....	340
Listing 167. The TaskStatus_t structure.....	342
Listing 168. The vTaskList() API function prototype .....	344
Listing 169. The vTaskGetRunTimeStats() API function prototype.....	345
Listing 170. 16-bit timer overflow interrupt handler used to count timer overflows .....	347
Listing 171. Macros added to FreeRTOSConfig.h to enable the collection of run-time statistics.....	347
Listing 172. The task that prints out the collected run-time statistics .....	348
Listing 173. The uxTaskGetStackHighWaterMark() API function prototype.....	360
Listing 174. The stack overflow hook function prototype .....	361

## List of Tables

---

Table 1. Comparing the FreeRTOS license with the OpenRTOS license.....	8
Table 2. FreeRTOS source files to include in the project .....	22
Table 3. Port specific data types used by FreeRTOS.....	23
Table 4. Macro prefixes .....	25
Table 5. Common macro definitions.....	25
Table 6. vPortDefineHeapRegions() parameters.....	39
Table 7. xPortGetFreeHeapSize() return value .....	43
Table 8. xPortGetMinimumEverFreeHeapSize() return value.....	44
Table 9. xTaskCreate() parameters and return value .....	50

Table 10.	vTaskDelay() parameters .....	68
Table 11.	vTaskDelayUntil() parameters .....	72
Table 12.	vTaskPrioritySet() parameters .....	80
Table 13.	uxTaskPriorityGet() parameters and return value .....	81
Table 14.	vTaskDelete() parameters .....	86
Table 15.	The FreeRTOSConfig.h settings that configure the kernel to use Prioritized Pre-emptive Scheduling with Time Slicing .....	92
Table 16.	An explanation of the terms used to describe the scheduling policy .....	93
Table 17.	The FreeRTOSConfig.h settings that configure the kernel to use Prioritized Pre-emptive Scheduling without Time Slicing.....	97
Table 18.	The FreeRTOSConfig.h settings that configure the kernel to use co-operative scheduling .....	99
Table 19.	xQueueCreate() parameters and return value .....	109
Table 20.	xQueueSendToFront() and xQueueSendToBack() function parameters and return value.....	110
Table 21.	xQueueReceive() function parameters and return values .....	113
Table 22.	uxQueueMessagesWaiting() function parameters and return value.....	115
Table 23.	Key to Figure 36 .....	125
Table 24.	xQueueCreateSet() parameters and return value .....	134
Table 25.	xQueueAddToSet() parameters and return value .....	135
Table 26.	xQueueSelectFromSet() parameters and return value .....	137
Table 27.	xQueueOverwrite() parameters and return value.....	146
Table 28.	xTimerCreate() parameters and return value .....	159
Table 29.	xTimerStart() parameters and return value .....	161
Table 30.	vTimerSetTimerID() parameters .....	167
Table 31.	pvTimerGetTimerID() parameters and return value .....	168
Table 32.	xTimerChangePeriod() parameters and return value .....	172
Table 33.	xTimerReset() parameters and return value .....	176
Table 34.	xSemaphoreCreateBinary() Return Value .....	195
Table 35.	xSemaphoreTake() parameters and return value .....	196
Table 36.	xSemaphoreGiveFromISR() parameters and return value.....	198
Table 37.	xSemaphoreCreateCounting() parameters and return value .....	211
Table 38.	xTimerPendFunctionCallFromISR() parameters and return value .....	215
Table 39.	xQueueSendToFrontFromISR() and xQueueSendToBackFromISR() parameters and return values .....	221
Table 40.	Constants that control interrupt nesting .....	229
Table 41.	xTaskResumeAll() return value .....	242
Table 42.	xSemaphoreCreateMutex() return value.....	246
Table 43.	xEventGroupCreate() return value.....	272
Table 44.	xEventGroupSetBits() parameters and return value .....	273
Table 45.	xEventGroupSetBitsFromISR() parameters and return value .....	274
Table 46.	The Effect of the uxBitsToWaitFor and xWaitForAllBits Parameters .....	276
Table 47.	xEventGroupWaitBits() parameters and return value.....	278
Table 48.	xEventGroupSync() parameters and return value.....	289

Table 49. xTaskNotifyGive() parameters and return value .....	300
Table 50. vTaskNotifyGiveFromISR() parameters and return value .....	300
Table 51. ulTaskNotifyTake() parameters and return value.....	302
Table 52. xTaskNotify() parameters and return value .....	309
Table 53. Valid xTaskNotify() eNotifyAction Parameter Values, and Their Resultant Effect on the Receiving Task's Notification Value .....	310
Table 54. xTaskNotifyWait() parameters and return value .....	311
Table 55. Macros used in the collection of run-time statistics.....	339
Table 56. uxTaskGetSystemState() parameters and return value.....	341
Table 57. TaskStatus_t structure members.....	342
Table 58. vTaskList() parameters .....	344
Table 59. vTaskGetRunTimeStats() parameters.....	345
Table 60. A selection of the most commonly used trace hook macros .....	349
Table 61. uxTaskGetStackHighWaterMark() parameters and return value.....	360