

Smart Home

1.0.0

Generated by Doxygen 1.9.1

1 uxQueueMessagesWaiting	1
2 vQueueDelete	1
3 vSemaphoreDelete	1
4 xTaskHandle	2
5 taskYIELD	2
6 taskENTER_CRITICAL	2
7 taskEXIT_CRITICAL	2
8 taskDISABLE_INTERRUPTS	2
9 taskENABLE_INTERRUPTS	3
10 xTaskGetTickCount	3
11 uxTaskGetNumberOfTasks	3
12 pcTaskGetTaskName	3
13 vTaskList	4
14 vTaskGetRunTimeStats	4
15 vTaskStartTrace	5
16 usTaskEndTrace	5
17 Module Index	5
17.1 Modules	5
18 Data Structure Index	6
18.1 Data Structures	6
19 File Index	7
19.1 File List	7
20 Module Documentation	10
20.1 xCoRoutineCreate	10
20.2 vCoRoutineSchedule	11
20.3 crSTART	12
20.4 crDELAY	13
20.5 crQUEUE_SEND	14
20.6 crQUEUE_RECEIVE	15
20.7 crQUEUE_SEND_FROM_ISR	16
20.8 crQUEUE_RECEIVE_FROM_ISR	17

20.9 xQueueCreate	19
20.10 xQueueSend	20
20.11 xQueueReceive	26
20.12 xQueueSendFromISR	30
20.13 xQueueReceiveFromISR	34
20.14 vSemaphoreCreateBinary	36
20.15 xSemaphoreTake	36
20.16 xSemaphoreTakeRecursive	37
20.17 xSemaphoreGive	39
20.18 xSemaphoreGiveRecursive	40
20.19 xSemaphoreGiveFromISR	41
20.20 vSemaphoreCreateMutex	43
20.21 xSemaphoreCreateCounting	44
20.22 xTaskCreate	45
20.23 xTaskCreateRestricted	47
20.24 vTaskDelete	49
20.25 vTaskDelay	49
20.26 vTaskDelayUntil	50
20.27 uxTaskPriorityGet	51
20.28 vTaskPrioritySet	52
20.29 vTaskSuspend	52
20.30 vTaskResume	53
20.31 vTaskResumeFromISR	54
20.32 vTaskStartScheduler	55
20.33 vTaskEndScheduler	55
20.34 vTaskSuspendAll	56
20.35 xTaskResumeAll	57
21 Data Structure Documentation	58
21.1 corCoRoutineControlBlock Struct Reference	58
21.1.1 Detailed Description	58
21.1.2 Field Documentation	58
21.2 FLAG_t Struct Reference	59
21.2.1 Detailed Description	59
21.2.2 Field Documentation	60
21.3 PASSWORD_t Struct Reference	60
21.3.1 Detailed Description	60
21.3.2 Field Documentation	61
21.4 QueueDefinition Struct Reference	61
21.4.1 Detailed Description	62
21.4.2 Field Documentation	62
21.5 SENSOR_t Struct Reference	63

21.5.1 Detailed Description	64
21.5.2 Field Documentation	64
21.6 SYSTEM_INPUTS_t Struct Reference	65
21.6.1 Detailed Description	65
21.6.2 Field Documentation	65
21.7 SYSTEM_t Struct Reference	66
21.7.1 Detailed Description	66
21.7.2 Field Documentation	67
21.8 tskTaskControlBlock Struct Reference	67
21.8.1 Detailed Description	68
21.8.2 Field Documentation	68
21.9 xLIST Struct Reference	69
21.9.1 Detailed Description	69
21.9.2 Field Documentation	69
21.10 xLIST_ITEM Struct Reference	70
21.10.1 Detailed Description	70
21.10.2 Field Documentation	71
21.11 xMEMORY_REGION Struct Reference	71
21.11.1 Detailed Description	72
21.11.2 Field Documentation	72
21.12 xMINI_LIST_ITEM Struct Reference	72
21.12.1 Detailed Description	73
21.12.2 Field Documentation	73
21.13 xTASK_PARAMTERS Struct Reference	73
21.13.1 Detailed Description	74
21.13.2 Field Documentation	74
21.14 xTIME_OUT Struct Reference	75
21.14.1 Detailed Description	75
21.14.2 Field Documentation	75
22 File Documentation	76
22.1 code/FreeRTOS/croutine.c File Reference	76
22.1.1 Macro Definition Documentation	76
22.1.2 Function Documentation	77
22.1.3 Variable Documentation	78
22.2 croutine.c	78
22.3 code/FreeRTOS/croutine.h File Reference	83
22.3.1 Macro Definition Documentation	84
22.3.2 Typedef Documentation	86
22.3.3 Function Documentation	87
22.4 croutine.h	88
22.5 code/FreeRTOS/FreeRTOS.h File Reference	90

22.5.1 Macro Definition Documentation	92
22.5.2 Typedef Documentation	101
22.6 FreeRTOS.h	101
22.7 code/FreeRTOS/FreeRTOSConfig.h File Reference	107
22.7.1 Macro Definition Documentation	108
22.8 FreeRTOSConfig.h	111
22.9 code/FreeRTOS/heap_1.c File Reference	112
22.9.1 Macro Definition Documentation	113
22.9.2 Function Documentation	113
22.10 heap_1.c	114
22.11 code/FreeRTOS/list.c File Reference	116
22.11.1 Function Documentation	117
22.12 list.c	118
22.13 code/FreeRTOS/list.h File Reference	120
22.13.1 Macro Definition Documentation	121
22.13.2 Typedef Documentation	123
22.13.3 Function Documentation	123
22.14 list.h	124
22.15 code/FreeRTOS/macros.h File Reference	128
22.15.1 Detailed Description	128
22.15.2 Macro Definition Documentation	128
22.16 macros.h	129
22.17 code/FreeRTOS/mpu_wrappers.h File Reference	130
22.17.1 Macro Definition Documentation	130
22.18 mpu_wrappers.h	131
22.19 code/FreeRTOS/port.c File Reference	132
22.19.1 Macro Definition Documentation	133
22.19.2 Typedef Documentation	134
22.19.3 Function Documentation	134
22.19.4 Variable Documentation	135
22.20 port.c	135
22.21 code/FreeRTOS/portable.h File Reference	140
22.21.1 Macro Definition Documentation	141
22.21.2 Function Documentation	142
22.22 portable.h	144
22.23 code/FreeRTOS/portmacro.h File Reference	148
22.23.1 Macro Definition Documentation	149
22.23.2 Typedef Documentation	152
22.23.3 Function Documentation	152
22.24 portmacro.h	152
22.25 code/FreeRTOS/projdefs.h File Reference	154
22.25.1 Macro Definition Documentation	154

22.25.2 Typedef Documentation	155
22.26 projdefs.h	156
22.27 code/FreeRTOS/queue.c File Reference	156
22.27.1 Macro Definition Documentation	158
22.27.2 Typedef Documentation	160
22.27.3 Function Documentation	160
22.28 queue.c	163
22.29 code/FreeRTOS/queue.h File Reference	181
22.29.1 Macro Definition Documentation	183
22.29.2 Typedef Documentation	186
22.29.3 Function Documentation	186
22.30 queue.h	190
22.31 code/FreeRTOS/semphr.h File Reference	192
22.31.1 Macro Definition Documentation	194
22.31.2 Typedef Documentation	197
22.32 semphr.h	197
22.33 code/FreeRTOS/StackMacros.h File Reference	199
22.33.1 Macro Definition Documentation	199
22.34 StackMacros.h	200
22.35 code/FreeRTOS/task.h File Reference	202
22.35.1 Macro Definition Documentation	204
22.35.2 Typedef Documentation	206
22.35.3 Function Documentation	206
22.36 task.h	213
22.37 code/FreeRTOS/tasks.c File Reference	217
22.37.1 Macro Definition Documentation	219
22.37.2 Typedef Documentation	221
22.37.3 Function Documentation	221
22.37.4 Variable Documentation	225
22.38 tasks.c	225
22.39 code/FreeRTOS/timers.c File Reference	255
22.39.1 Macro Definition Documentation	255
22.40 timers.c	256
22.41 code/FreeRTOS/timers.h File Reference	264
22.41.1 Macro Definition Documentation	265
22.41.2 Typedef Documentation	276
22.41.3 Function Documentation	276
22.42 timers.h	280
22.43 code/HAL/DC_MOTOR/DC_MOTOR.c File Reference	281
22.43.1 Function Documentation	282
22.44 DC_MOTOR.c	284
22.45 code/HAL/DC_MOTOR/DC_MOTOR.h File Reference	284

22.45.1 Macro Definition Documentation	286
22.45.2 Function Documentation	286
22.46 DC_MOTOR.h	288
22.47 code/HAL/DC_MOTOR/DC_MOTOR_CFG.h File Reference	288
22.47.1 Macro Definition Documentation	289
22.47.2 Function Documentation	289
22.48 DC_MOTOR_CFG.h	290
22.49 code/HAL/DC_MOTOR_POT/DC_MOTOR_POT.c File Reference	290
22.49.1 Function Documentation	290
22.50 DC_MOTOR_POT.c	292
22.51 code/HAL/DC_MOTOR_POT/DC_MOTOR_POT.h File Reference	293
22.51.1 Macro Definition Documentation	294
22.51.2 Function Documentation	294
22.52 DC_MOTOR_POT.h	296
22.53 code/HAL/DC_MOTOR_POT/DC_MOTOR_POT_CFG.h File Reference	296
22.53.1 Macro Definition Documentation	297
22.54 DC_MOTOR_POT_CFG.h	297
22.55 code/HAL/HCLCD/HCLCD_Config.h File Reference	298
22.55.1 Macro Definition Documentation	298
22.56 HCLCD_Config.h	300
22.57 code/HAL/HCLCD/HCLCD_Interface.h File Reference	300
22.57.1 Macro Definition Documentation	301
22.57.2 Function Documentation	301
22.58 HCLCD_Interface.h	308
22.59 code/HAL/HCLCD/HCLCD_Private.h File Reference	308
22.59.1 Macro Definition Documentation	309
22.59.2 Function Documentation	311
22.60 HCLCD_Private.h	311
22.61 code/HAL/HCLCD/HCLCD_Program.c File Reference	312
22.61.1 Function Documentation	312
22.62 HCLCD_Program.c	319
22.63 code/HAL/KEY_PAD/HKPD_Config.h File Reference	321
22.63.1 Macro Definition Documentation	322
22.64 HKPD_Config.h	324
22.65 code/HAL/KEY_PAD/HKPD_Interface.h File Reference	324
22.65.1 Macro Definition Documentation	325
22.65.2 Function Documentation	325
22.66 HKPD_Interface.h	326
22.67 code/HAL/KEY_PAD/HKPD_Private.h File Reference	326
22.68 HKPD_Private.h	326
22.69 code/HAL/KEY_PAD/HKPD_Program.c File Reference	326
22.69.1 Function Documentation	327
22.69.2 Variable Documentation	328

22.70 HKPD_Program.c	329
22.71 code/HAL/SERVO_M/SERVO_Config.h File Reference	330
22.72 SERVO_Config.h	330
22.73 code/HAL/SERVO_M/SERVO_Interface.h File Reference	330
22.73.1 Function Documentation	331
22.74 SERVO_Interface.h	333
22.75 code/HAL/SERVO_M/SERVO_Private.h File Reference	333
22.76 SERVO_Private.h	333
22.77 code/HAL/SERVO_M/SERVO_Program.c File Reference	333
22.77.1 Function Documentation	333
22.78 SERVO_Program.c	335
22.79 code/LIB/LBIT_MATH.h File Reference	335
22.79.1 Macro Definition Documentation	336
22.80 LBIT_MATH.h	336
22.81 code/LIB/LSTD_TYPES.h File Reference	336
22.81.1 Macro Definition Documentation	337
22.81.2 Typedef Documentation	337
22.81.3 Enumeration Type Documentation	338
22.82 LSTD_TYPES.h	339
22.83 code/main.c File Reference	339
22.83.1 Detailed Description	340
22.83.2 Function Documentation	341
22.83.3 Variable Documentation	351
22.84 main.c	352
22.85 code/main.h File Reference	356
22.85.1 Function Documentation	357
22.86 main.h	367
22.87 code/main_cfg.c File Reference	368
22.87.1 Variable Documentation	368
22.88 main_cfg.c	369
22.89 code/main_cfg.h File Reference	370
22.89.1 Macro Definition Documentation	370
22.89.2 Variable Documentation	371
22.90 main_cfg.h	371
22.91 code/MCAL/MADC/MADC_Config.h File Reference	372
22.91.1 Macro Definition Documentation	372
22.92 MADC_Config.h	373
22.93 code/MCAL/MADC/MADC_Interface.h File Reference	373
22.93.1 Macro Definition Documentation	374
22.93.2 Function Documentation	375
22.94 MADC_Interface.h	377
22.95 code/MCAL/MADC/MADC_Private.h File Reference	377

22.95.1 Macro Definition Documentation	378
22.95.2 Function Documentation	380
22.96 MADC_Private.h	381
22.97 code/MCAL/MADC/MADC_Program.c File Reference	381
22.97.1 Function Documentation	382
22.97.2 Variable Documentation	383
22.98 MADC_Program.c	384
22.99 code/MCAL/MDIO/MDIO_Config.h File Reference	385
22.100 MDIO_Config.h	385
22.101 code/MCAL/MDIO/MDIO_Interface.h File Reference	385
22.101.1 Macro Definition Documentation	385
22.101.2 Enumeration Type Documentation	386
22.101.3 Function Documentation	387
22.102 MDIO_Interface.h	389
22.103 code/MCAL/MDIO/MDIO_Private.h File Reference	390
22.103.1 Macro Definition Documentation	391
22.104 MDIO_Private.h	392
22.105 code/MCAL/MDIO/MDIO_Program.c File Reference	393
22.105.1 Function Documentation	393
22.106 MDIO_Program.c	396
22.107 code/MCAL/MGIE/MGIE_Interface.h File Reference	399
22.107.1 Function Documentation	400
22.108 MGIE_Interface.h	400
22.109 code/MCAL/MGIE/MGIE_Private.h File Reference	401
22.109.1 Macro Definition Documentation	401
22.110 MGIE_Private.h	401
22.111 code/MCAL/MGIE/MGIE_Program.c File Reference	402
22.111.1 Function Documentation	402
22.112 MGIE_Program.c	403
22.113 code/MCAL/TIMER/TIMER_Config.h File Reference	403
22.113.1 Macro Definition Documentation	403
22.114 TIMER_Config.h	404
22.115 code/MCAL/TIMER/TIMER_Interface.h File Reference	405
22.115.1 Function Documentation	405
22.116 TIMER_Interface.h	406
22.117 code/MCAL/TIMER/TIMER_Private.h File Reference	406
22.117.1 Macro Definition Documentation	407
22.118 TIMER_Private.h	410
22.119 code/MCAL/TIMER/TIMER_Program.c File Reference	410
22.119.1 Function Documentation	411
22.119.2 Variable Documentation	412
22.120 TIMER_Program.c	412

22.121 code/MCAL/TIMER2/TIMER2_Config.h File Reference	414
22.121.1 Macro Definition Documentation	414
22.122 TIMER2_Config.h	415
22.123 code/MCAL/TIMER2/TIMER2_Interface.h File Reference	416
22.123.1 Function Documentation	416
22.124 TIMER2_Interface.h	419
22.125 code/MCAL/TIMER2/TIMER2_Private.h File Reference	419
22.125.1 Macro Definition Documentation	420
22.126 TIMER2_Private.h	423
22.127 code/MCAL/TIMER2/TIMER2_Program.c File Reference	423
22.127.1 Function Documentation	424
22.127.2 Variable Documentation	426
22.128 TIMER2_Program.c	427
22.129 code/Release/FreeRTOS/croutine.d File Reference	428
22.130 croutine.d	428
22.131 code/Release/FreeRTOS/heap_1.d File Reference	429
22.132 heap_1.d	429
22.133 code/Release/FreeRTOS/list.d File Reference	429
22.134 list.d	429
22.135 code/Release/FreeRTOS/port.d File Reference	429
22.136 port.d	429
22.137 code/Release/FreeRTOS/queue.d File Reference	430
22.138 queue.d	430
22.139 code/Release/FreeRTOS/tasks.d File Reference	430
22.140 tasks.d	430
22.141 code/Release/FreeRTOS/timers.d File Reference	430
22.142 timers.d	430
22.143 code/Release/HAL/DC_MOTOR/DC_MOTOR.d File Reference	431
22.144 DC_MOTOR.d	431
22.145 code/Release/HAL/DC_MOTOR_POT/DC_MOTOR_POT.d File Reference	431
22.146 DC_MOTOR_POT.d	431
22.147 code/Release/HAL/HCLCD/HCLCD_Program.d File Reference	431
22.148 HCLCD_Program.d	431
22.149 code/Release/HAL/KEY_PAD/HKPD_Program.d File Reference	432
22.150 HKPD_Program.d	432
22.151 code/Release/HAL/SERVO_M/SERVO_Program.d File Reference	432
22.152 SERVO_Program.d	432
22.153 code/Release/main.d File Reference	432
22.154 main.d	432
22.155 code/Release/main_cfg.d File Reference	433
22.156 main_cfg.d	433
22.157 code/Release/MCAL/MADC/MADC_Program.d File Reference	434

22.158 MADC_Program.d	434
22.159 code/Release/MCAL/MDIO/MDIO_Program.d File Reference	434
22.160 MDIO_Program.d	434
22.161 code/Release/MCAL/MGIE/MGIE_Program.d File Reference	435
22.162 MGIE_Program.d	435
22.163 code/Release/MCAL/TIMER/TIMER_Program.d File Reference	435
22.164 TIMER_Program.d	435
22.165 code/Release/MCAL/TIMER2/TIMER2_Program.d File Reference	435
22.166 TIMER2_Program.d	435
Index	437

1 uxQueueMessagesWaiting

queue. h

```
unsigned portBASE_TYPE uxQueueMessagesWaiting( const xQueueHandle xQueue );
```

Return the number of messages stored in a queue.

Parameters

<i>xQueue</i>	A handle to the queue being queried.
---------------	--------------------------------------

Returns

The number of messages available in the queue.

2 vQueueDelete

queue. h

```
void vQueueDelete( xQueueHandle xQueue );
```

Delete a queue - freeing all the memory allocated for storing of items placed on the queue.

Parameters

<i>xQueue</i>	A handle to the queue to be deleted.
---------------	--------------------------------------

3 vSemaphoreDelete

semphr. h

```
void vSemaphoreDelete( xSemaphoreHandle xSemaphore );
```

Delete a semaphore. This function must be used with care. For example, do not delete a mutex type semaphore if the mutex is held by a task.

Parameters

<i>xSemaphore</i>	A handle to the semaphore to be deleted.
-------------------	--

4 xTaskHandle

task.h

Type by which tasks are referenced. For example, a call to xTaskCreate returns (via a pointer parameter) an *xTaskHandle* variable that can then be used as a parameter to vTaskDelete to delete the task.

5 taskYIELD

task.h

Macro for forcing a context switch.

6 taskENTER_CRITICAL

task.h

Macro to mark the start of a critical code region. Preemptive context switches cannot occur when in a critical region.

NOTE: This may alter the stack (depending on the portable implementation) so must be used with care!

7 taskEXIT_CRITICAL

task.h

Macro to mark the end of a critical code region. Preemptive context switches cannot occur when in a critical region.

NOTE: This may alter the stack (depending on the portable implementation) so must be used with care!

8 taskDISABLE_INTERRUPTS

task.h

Macro to disable all maskable interrupts.

9 taskENABLE_INTERRUPTS

task.h

Macro to enable microcontroller interrupts.

10 xTaskGetTickCount

task.h

```
portTickType xTaskGetTickCount( void );
```

Returns

The count of ticks since vTaskStartScheduler was called.

task.h

```
portTickType xTaskGetTickCountFromISR( void );
```

Returns

The count of ticks since vTaskStartScheduler was called.

This is a version of [xTaskGetTickCount\(\)](#) that is safe to be called from an ISR - provided that portTickType is the natural word size of the microcontroller being used or interrupt nesting is either not supported or not being used.

11 uxTaskGetNumberOfTasks

task.h

```
unsigned short uxTaskGetNumberOfTasks( void );
```

Returns

The number of tasks that the real time kernel is currently managing. This includes all ready, blocked and suspended tasks. A task that has been deleted but not yet freed by the idle task will also be included in the count.

12 pcTaskGetTaskName

task.h

```
signed char *pcTaskGetTaskName( xTaskHandle xTaskToQuery );
```

Returns

The text (human readable) name of the task referenced by the handle xTaskToQuery. A task can query its own name by either passing in its own handle, or by setting xTaskToQuery to NULL. INCLUDE_pcTaskGetTaskName must be set to 1 in [FreeRTOSConfig.h](#) for [pcTaskGetTaskName\(\)](#) to be available.

13 vTaskList

task.h

```
void vTaskList( char *pcWriteBuffer );
```

configUSE_TRACE_FACILITY must be defined as 1 for this function to be available. See the configuration section for more information.

NOTE: This function will disable interrupts for its duration. It is not intended for normal application runtime use but as a debug aid.

Lists all the current tasks, along with their current state and stack usage high water mark.

Tasks are reported as blocked ('B'), ready ('R'), deleted ('D') or suspended ('S').

Parameters

<i>pcWriteBuffer</i>	A buffer into which the above mentioned details will be written, in ascii form. This buffer is assumed to be large enough to contain the generated report. Approximately 40 bytes per task should be sufficient.
----------------------	--

14 vTaskGetRunTimeStats

task.h

```
void vTaskGetRunTimeStats( char *pcWriteBuffer );
```

configGENERATE_RUN_TIME_STATS must be defined as 1 for this function to be available. The application must also then provide definitions for [portCONFIGURE_TIMER_FOR_RUN_TIME_STATS\(\)](#) and [portGET_RUN_TIME_COUNTER_VALUE](#) to configure a peripheral timer/counter and return the timers current count value respectively. The counter should be at least 10 times the frequency of the tick count.

NOTE: This function will disable interrupts for its duration. It is not intended for normal application runtime use but as a debug aid.

Setting configGENERATE_RUN_TIME_STATS to 1 will result in a total accumulated execution time being stored for each task. The resolution of the accumulated time value depends on the frequency of the timer configured by the [portCONFIGURE_TIMER_FOR_RUN_TIME_STATS\(\)](#) macro. Calling [vTaskGetRunTimeStats\(\)](#) writes the total execution time of each task into a buffer, both as an absolute count value and as a percentage of the total system execution time.

Parameters

<i>pcWriteBuffer</i>	A buffer into which the execution times will be written, in ascii form. This buffer is assumed to be large enough to contain the generated report. Approximately 40 bytes per task should be sufficient.
----------------------	--

15 vTaskStartTrace

task.h

```
void vTaskStartTrace( char * pcBuffer, unsigned portBASE_TYPE uxBufferSize );
```

Starts a real time kernel activity trace. The trace logs the identity of which task is running when.

The trace file is stored in binary format. A separate DOS utility called convrce.exe is used to convert this into a tab delimited text file which can be viewed and plotted in a spread sheet.

Parameters

<i>pcBuffer</i>	The buffer into which the trace will be written.
<i>ulBufferSize</i>	The size of pcBuffer in bytes. The trace will continue until either the buffer is full, or ulTaskEndTrace () is called.

16 usTaskEndTrace

task.h

```
unsigned long ulTaskEndTrace( void );
```

Stops a kernel activity trace. See vTaskStartTrace ().

Returns

The number of bytes that have been written into the trace buffer.

17 Module Index

17.1 Modules

Here is a list of all modules:

xCoRoutineCreate	10
vCoRoutineSchedule	11
crSTART	12
crDELAY	13
crQUEUE_SEND	14
crQUEUE_RECEIVE	15
crQUEUE_SEND_FROM_ISR	16

crQUEUE_RECEIVE_FROM_ISR	17
xQueueCreate	19
xQueueSend	20
xQueueReceive	26
xQueueSendFromISR	30
xQueueReceiveFromISR	34
vSemaphoreCreateBinary	36
xSemaphoreTake	36
xSemaphoreTakeRecursive	37
xSemaphoreGive	39
xSemaphoreGiveRecursive	40
xSemaphoreGiveFromISR	41
vSemaphoreCreateMutex	43
xSemaphoreCreateCounting	44
xTaskCreate	45
xTaskCreateRestricted	47
vTaskDelete	49
vTaskDelay	49
vTaskDelayUntil	50
uxTaskPriorityGet	51
vTaskPrioritySet	52
vTaskSuspend	52
vTaskResume	53
vTaskResumeFromISR	54
vTaskStartScheduler	55
vTaskEndScheduler	55
vTaskSuspendAll	56
xTaskResumeAll	57

18 Data Structure Index

18.1 Data Structures

Here are the data structures with brief descriptions:

<code>corCoRoutineControlBlock</code>	58
<code>FLAG_t</code> This is the structure that holds the system's state flags	59
<code>PASSWORD_t</code> This is the structure that holds the system's password data	60
<code>QueueDefinition</code>	61
<code>SENSOR_t</code> This is the structure that holds the system's sensors data	63
<code>SYSTEM_INPUTS_t</code>	65
<code>SYSTEM_t</code> This is the structure that holds the system's data that is used to control the system and display it on the LCD screen. The system's data is:	66
<code>tskTaskControlBlock</code>	67
<code>xLIST</code>	69
<code>xLIST_ITEM</code>	70
<code>xMEMORY_REGION</code>	71
<code>xMINI_LIST_ITEM</code>	72
<code>xTASK_PARAMTERS</code>	73
<code>xTIME_OUT</code>	75

19 File Index

19.1 File List

Here is a list of all files with brief descriptions:

<code>code/main.c</code> Smart Home Security System	339
<code>code/main.h</code>	356
<code>code/main_cfg.c</code>	368
<code>code/main_cfg.h</code>	370
<code>code/FreeRTOS/croutine.c</code>	76
<code>code/FreeRTOS/croutine.h</code>	83
<code>code/FreeRTOS/FreeRTOS.h</code>	90
<code>code/FreeRTOS/FreeRTOSConfig.h</code>	107
<code>code/FreeRTOS/heap_1.c</code>	112
<code>code/FreeRTOS/list.c</code>	116

code/FreeRTOS/list.h	120
code/FreeRTOS/macros.h	
Set and clear macros and get high nibble and low nibble .	
128	
code/FreeRTOS/mpu_wrappers.h	130
code/FreeRTOS/port.c	132
code/FreeRTOS/portable.h	140
code/FreeRTOS/portmacro.h	148
code/FreeRTOS/projdefs.h	154
code/FreeRTOS/queue.c	156
code/FreeRTOS/queue.h	181
code/FreeRTOS/semphr.h	192
code/FreeRTOS/StackMacros.h	199
code/FreeRTOS/task.h	202
code/FreeRTOS/tasks.c	217
code/FreeRTOS/timers.c	255
code/FreeRTOS/timers.h	264
code/HAL/DC_MOTOR/DC_MOTOR.c	281
code/HAL/DC_MOTOR/DC_MOTOR.h	284
code/HAL/DC_MOTOR/DC_MOTOR_CFG.h	288
code/HAL/DC_MOTOR_POT/DC_MOTOR_POT.c	290
code/HAL/DC_MOTOR_POT/DC_MOTOR_POT.h	293
code/HAL/DC_MOTOR_POT/DC_MOTOR_POT_CFG.h	296
code/HAL/HCLCD/HCLCD_Config.h	298
code/HAL/HCLCD/HCLCD_Interface.h	300
code/HAL/HCLCD/HCLCD_Private.h	308
code/HAL/HCLCD/HCLCD_Program.c	312
code/HAL/KEY_PAD/HKPD_Config.h	321
code/HAL/KEY_PAD/HKPD_Interface.h	324
code/HAL/KEY_PAD/HKPD_Private.h	326
code/HAL/KEY_PAD/HKPD_Program.c	326
code/HAL/SERVO_M/SERVO_Config.h	330
code/HAL/SERVO_M/SERVO_Interface.h	330

code/HAL/SERVO_M/SERVO_Private.h	333
code/HAL/SERVO_M/SERVO_Program.c	333
code/LIB/LBIT_MATH.h	335
code/LIB/LSTD_TYPES.h	336
code/MCAL/MADC/MADC_Config.h	372
code/MCAL/MADC/MADC_Interface.h	373
code/MCAL/MADC/MADC_Private.h	377
code/MCAL/MADC/MADC_Program.c	381
code/MCAL/MDIO/MDIO_Config.h	385
code/MCAL/MDIO/MDIO_Interface.h	385
code/MCAL/MDIO/MDIO_Private.h	390
code/MCAL/MDIO/MDIO_Program.c	393
code/MCAL/MGIE/MGIE_Interface.h	399
code/MCAL/MGIE/MGIE_Private.h	401
code/MCAL/MGIE/MGIE_Program.c	402
code/MCAL/TIMER/TIMER_Config.h	403
code/MCAL/TIMER/TIMER_Interface.h	405
code/MCAL/TIMER/TIMER_Private.h	406
code/MCAL/TIMER/TIMER_Program.c	410
code/MCAL/TIMER2/TIMER2_Config.h	414
code/MCAL/TIMER2/TIMER2_Interface.h	416
code/MCAL/TIMER2/TIMER2_Private.h	419
code/MCAL/TIMER2/TIMER2_Program.c	423
code/Release/main.d	432
code/Release/main_cfg.d	433
code/Release/FreeRTOS/croutine.d	428
code/Release/FreeRTOS/heap_1.d	429
code/Release/FreeRTOS/list.d	429
code/Release/FreeRTOS/port.d	429
code/Release/FreeRTOS/queue.d	430
code/Release/FreeRTOS/tasks.d	430
code/Release/FreeRTOS/timers.d	430

code/Release/HAL/DC_MOTOR/ DC_MOTOR.d	431
code/Release/HAL/DC_MOTOR_POT/ DC_MOTOR_POT.d	431
code/Release/HAL/HCLCD/ HCLCD_Program.d	431
code/Release/HAL/KEY_PAD/ HKPD_Program.d	432
code/Release/HAL/SERVO_M/ SERVO_Program.d	432
code/Release/MCAL/MADC/ MADC_Program.d	434
code/Release/MCAL/MDIO/ MDIO_Program.d	434
code/Release/MCAL/MGIE/ MGIE_Program.d	435
code/Release/MCAL/TIMER/ TIMER_Program.d	435
code/Release/MCAL/TIMER2/ TIMER2_Program.d	435

20 Module Documentation

20.1 xCoRoutineCreate

routine.h

```
portBASE_TYPE xCoRoutineCreate(
    crCOROUTINE_CODE pxCoRoutineCode,
    unsigned portBASE_TYPE uxPriority,
    unsigned portBASE_TYPE uxIndex
);
```

Create a new co-routine and add it to the list of co-routines that are ready to run.

Parameters

<i>pxCoRoutineCode</i>	Pointer to the co-routine function. Co-routine functions require special syntax - see the co-routine section of the WEB documentation for more information.
<i>uxPriority</i>	The priority with respect to other co-routines at which the co-routine will run.
<i>uxIndex</i>	Used to distinguish between different co-routines that execute the same function. See the example below and the co-routine section of the WEB documentation for further information.

Returns

pdPASS if the co-routine was successfully created and added to a ready list, otherwise an error code defined with [ProjDefs.h](#).

Example usage:

```
// Co-routine to be created.
void vFlashCoRoutine( xCoRoutineHandle xHandle, unsigned portBASE_TYPE uxIndex )
{
```

```

// Variables in co-routines must be declared static if they must maintain value across a blocking
// This may not be necessary for const variables.
static const char cLedToFlash[ 2 ] = { 5, 6 };
static const portTickType uxFlashRates[ 2 ] = { 200, 400 };

// Must start every co-routine with a call to crSTART();
crSTART( xHandle );

for( ; ; )
{
    // This co-routine just delays for a fixed period, then toggles
    // an LED. Two co-routines are created using this function, so
    // the uxIndex parameter is used to tell the co-routine which
    // LED to flash and how long to delay. This assumes xQueue has
    // already been created.
    vParTestToggleLED( cLedToFlash[ uxIndex ] );
    crDELAY( xHandle, uxFlashRates[ uxIndex ] );
}

// Must end every co-routine with a call to crEND();
crEND();
}

// Function that creates two co-routines.
void vOtherFunction( void )
{
unsigned char ucParameterToPass;
xTaskHandle xHandle;

// Create two co-routines at priority 0. The first is given index 0
// so (from the code above) toggles LED 5 every 200 ticks. The second
// is given index 1 so toggles LED 6 every 400 ticks.
for( uxIndex = 0; uxIndex < 2; uxIndex++ )
{
    xCoRoutineCreate( vFlashCoRoutine, 0, uxIndex );
}
}
}

```

20.2 vCoRoutineSchedule

routine.h

```
void vCoRoutineSchedule( void );
```

Run a co-routine.

[vCoRoutineSchedule\(\)](#) executes the highest priority co-routine that is able to run. The co-routine will execute until it either blocks, yields or is preempted by a task. Co-routines execute cooperatively so one co-routine cannot be preempted by another, but can be preempted by a task.

If an application comprises of both tasks and co-routines then vCoRoutineSchedule should be called from the idle task (in an idle task hook).

Example usage:

```
// This idle task hook will schedule a co-routine each time it is called.  
// The rest of the idle task will execute between co-routine calls.  
void vApplicationIdleHook( void )  
{  
    vCoRoutineSchedule();  
}  
  
// Alternatively, if you do not require any other part of the idle task to  
// execute, the idle task hook can call vCoRoutineScheduler() within an  
// infinite loop.  
void vApplicationIdleHook( void )  
{  
    for( ;; )  
    {  
        vCoRoutineSchedule();  
    }  
}
```

20.3 crSTART

croutine.h

```
crSTART( xCoRoutineHandle xHandle );
```

This macro MUST always be called at the start of a co-routine function.

Example usage:

```
// Co-routine to be created.  
void vACoRoutine( xCoRoutineHandle xHandle, unsigned portBASE_TYPE uxIndex )  
{  
    // Variables in co-routines must be declared static if they must maintain value across a blocking  
    static long ulAVariable;  
  
    // Must start every co-routine with a call to crSTART();  
    crSTART( xHandle );  
  
    for( ;; )  
    {  
        // Co-routine functionality goes here.  
    }  
  
    // Must end every co-routine with a call to crEND();  
    crEND();  
}
```

croutine.h

```
crEND();
```

This macro MUST always be called at the end of a co-routine function.

Example usage:

```

// Co-routine to be created.
void vACoRoutine( xCoRoutineHandle xHandle, unsigned portBASE_TYPE uxIndex )
{
// Variables in co-routines must be declared static if they must maintain value across a blocking
static long ulAVariable;

// Must start every co-routine with a call to crSTART();
crSTART( xHandle );

for( ;; )
{
    // Co-routine functionality goes here.
}

// Must end every co-routine with a call to crEND();
crEND();
}

```

20.4 crDELAY

routine.h

```
crDELAY( xCoRoutineHandle xHandle, portTickType xTicksToDelay );
```

Delay a co-routine for a fixed period of time.

crDELAY can only be called from the co-routine function itself - not from within a function called by the co-routine function. This is because co-routines do not maintain their own stack.

Parameters

<i>xHandle</i>	The handle of the co-routine to delay. This is the xHandle parameter of the co-routine function.
<i>xTickToDelay</i>	The number of ticks that the co-routine should delay for. The actual amount of time this equates to is defined by configTICK_RATE_HZ (set in FreeRTOSConfig.h). The constant portTICK_RATE_MS can be used to convert ticks to milliseconds.

Example usage:

```

// Co-routine to be created.
void vACoRoutine( xCoRoutineHandle xHandle, unsigned portBASE_TYPE uxIndex )
{
// Variables in co-routines must be declared static if they must maintain value across a blocking
// This may not be necessary for const variables.
// We are to delay for 200ms.
static const xTickType xDelayTime = 200 / portTICK_RATE_MS;

// Must start every co-routine with a call to crSTART();
crSTART( xHandle );

for( ;; )
{
    // Delay for 200ms.
    crDELAY( xHandle, xDelayTime );

    // Do something here.
}

```

```
// Must end every co-routine with a call to crEND();
crEND();
}
```

20.5 crQUEUE_SEND

```
crQUEUE_SEND (
    xCoRoutineHandle xHandle,
    xQueueHandle pxQueue,
    void *pvItemToQueue,
    portTickType xTicksToWait,
    portBASE_TYPE *pxResult
)
```

The macro's [crQUEUE_SEND\(\)](#) and [crQUEUE_RECEIVE\(\)](#) are the co-routine equivalent to the [xQueueSend\(\)](#) and [xQueueReceive\(\)](#) functions used by tasks.

crQUEUE_SEND and crQUEUE_RECEIVE can only be used from a co-routine whereas [xQueueSend\(\)](#) and [xQueueReceive\(\)](#) can only be used from tasks.

crQUEUE_SEND can only be called from the co-routine function itself - not from within a function called by the co-routine function. This is because co-routines do not maintain their own stack.

See the co-routine section of the WEB documentation for information on passing data between tasks and co-routines and between ISR's and co-routines.

Parameters

<i>xHandle</i>	The handle of the calling co-routine. This is the xHandle parameter of the co-routine function.
<i>pxQueue</i>	The handle of the queue on which the data will be posted. The handle is obtained as the return value when the queue is created using the xQueueCreate() API function.
<i>pvItemToQueue</i>	A pointer to the data being posted onto the queue. The number of bytes of each queued item is specified when the queue is created. This number of bytes is copied from <i>pvItemToQueue</i> into the queue itself.
<i>xTickToDelay</i>	The number of ticks that the co-routine should block to wait for space to become available on the queue, should space not be available immediately. The actual amount of time this equates to is defined by configTICK_RATE_HZ (set in FreeRTOSConfig.h). The constant portTICK_RATE_MS can be used to convert ticks to milliseconds (see example below).
<i>pxResult</i>	The variable pointed to by <i>pxResult</i> will be set to pdPASS if data was successfully posted onto the queue, otherwise it will be set to an error defined within ProjDefs.h .

Example usage:

```
// Co-routine function that blocks for a fixed period then posts a number onto
// a queue.
static void prvCoRoutineFlashTask( xCoRoutineHandle xHandle, unsigned portBASE_TYPE uxIndex )
{
// Variables in co-routines must be declared static if they must maintain value across a blocking
static portBASE_TYPE xNumberToPost = 0;
static portBASE_TYPE xResult;

// Co-routines must begin with a call to crSTART().
```

```

crSTART( xHandle );

for( ;; )
{
    // This assumes the queue has already been created.
    crQUEUE_SEND( xHandle, xCoRoutineQueue, &xNumberToPost, NO_DELAY, &xResult );

    if( xResult != pdPASS )
    {
        // The message was not posted!
    }

    // Increment the number to be posted onto the queue.
    xNumberToPost++;

    // Delay for 100 ticks.
    crDELAY( xHandle, 100 );
}

// Co-routines must end with a call to crEND().
crEND();
}

```

20.6 crQUEUE_RECEIVE

routine.h

```

crQUEUE_RECEIVE(
    xCoRoutineHandle xHandle,
    xQueueHandle pxQueue,
    void *pvBuffer,
    portTickType xTicksToWait,
    portBASE_TYPE *pxResult
)

```

The macro's `crQUEUE_SEND()` and `crQUEUE_RECEIVE()` are the co-routine equivalent to the `xQueueSend()` and `xQueueReceive()` functions used by tasks.

`crQUEUE_SEND` and `crQUEUE_RECEIVE` can only be used from a co-routine whereas `xQueueSend()` and `xQueueReceive()` can only be used from tasks.

`crQUEUE_RECEIVE` can only be called from the co-routine function itself - not from within a function called by the co-routine function. This is because co-routines do not maintain their own stack.

See the co-routine section of the WEB documentation for information on passing data between tasks and co-routines and between ISR's and co-routines.

Parameters

<code>xHandle</code>	The handle of the calling co-routine. This is the <code>xHandle</code> parameter of the co-routine function.
<code>pxQueue</code>	The handle of the queue from which the data will be received. The handle is obtained as the return value when the queue is created using the <code>xQueueCreate()</code> API function.
<code>pvBuffer</code>	The buffer into which the received item is to be copied. The number of bytes of each queued item is specified when the queue is created. This number of bytes is copied into <code>pvBuffer</code> .
<code>xTickToDelay</code>	The number of ticks that the co-routine should block to wait for data to become available from the queue, should data not be available immediately. The actual amount of time this equates to is defined by <code>configTICK_RATE_HZ</code> (set in <code>FreeRTOSConfig.h</code>). The constant <code>portTICK_RATE_MS</code> can be used to convert ticks to milliseconds (see the <code>crQUEUE_SEND</code> example).
<code>pxResult</code>	The variable pointed to by <code>pxResult</code> will be set to <code>pdPASS</code> if data was successfully retrieved from the queue, otherwise it will be set to an error code as defined within <code>ProjDefs.h</code> .

Example usage:

```
// A co-routine receives the number of an LED to flash from a queue. It
// blocks on the queue until the number is received.
static void prvCoRoutineFlashWorkTask( xCoRoutineHandle xHandle, unsigned portBASE_TYPE uxIndex )
{
// Variables in co-routines must be declared static if they must maintain value across a blocking
static portBASE_TYPE xResult;
static unsigned portBASE_TYPE uxLEDToFlash;

// All co-routines must start with a call to crSTART().
crSTART( xHandle );

for( ;; )
{
    // Wait for data to become available on the queue.
    crQUEUE_RECEIVE( xHandle, xCoRoutineQueue, &uxLEDToFlash, portMAX_DELAY, &xResult );

    if( xResult == pdPASS )
    {
        // We received the LED to flash - flash it!
        vParTestToggleLED( uxLEDToFlash );
    }
}

crEND();
}
```

20.7 crQUEUE_SEND_FROM_ISR

routine.h

```
crQUEUE_SEND_FROM_ISR(
    xQueueHandle pxQueue,
    void *pvItemToQueue,
    portBASE_TYPE xCoRoutinePreviouslyWoken
)
```

The macro's `crQUEUE_SEND_FROM_ISR()` and `crQUEUE_RECEIVE_FROM_ISR()` are the co-routine equivalent to the `xQueueSendFromISR()` and `xQueueReceiveFromISR()` functions used by tasks.

`crQUEUE_SEND_FROM_ISR()` and `crQUEUE_RECEIVE_FROM_ISR()` can only be used to pass data between a co-routine and an ISR, whereas `xQueueSendFromISR()` and `xQueueReceiveFromISR()` can only be used to pass data between a task and an ISR.

`crQUEUE_SEND_FROM_ISR` can only be called from an ISR to send data to a queue that is being used from within a co-routine.

See the co-routine section of the WEB documentation for information on passing data between tasks and co-routines and between ISR's and co-routines.

Parameters

<code>xQueue</code>	The handle to the queue on which the item is to be posted.
<code>pvItemToQueue</code>	A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from <code>pvItemToQueue</code> into the queue storage area.
<code>xCoRoutinePreviouslyWoken</code>	This is included so an ISR can post onto the same queue multiple times from a single interrupt. The first call should always pass in <code>pdFALSE</code> . Subsequent calls should pass in the value returned from the previous call.

Returns

pdTRUE if a co-routine was woken by posting onto the queue. This is used by the ISR to determine if a context switch may be required following the ISR.

Example usage:

```
// A co-routine that blocks on a queue waiting for characters to be received.
static void vReceivingCoRoutine( xCoRoutineHandle xHandle, unsigned portBASE_TYPE uxIndex )
{
    char cRxedChar;
    portBASE_TYPE xResult;

    // All co-routines must start with a call to crSTART().
    crSTART( xHandle );

    for( ;; )
    {
        // Wait for data to become available on the queue.  This assumes the
        // queue xCommsRxQueue has already been created!
        crQUEUE_RECEIVE( xHandle, xCommsRxQueue, &uxLEDtoFlash, portMAX_DELAY, &xResult );

        // Was a character received?
        if( xResult == pdPASS )
        {
            // Process the character here.
        }
    }

    // All co-routines must end with a call to crEND().
    crEND();
}

// An ISR that uses a queue to send characters received on a serial port to
// a co-routine.
void vUART_ISR( void )
{
    char cRxedChar;
    portBASE_TYPE xCRWokenByPost = pdFALSE;

    // We loop around reading characters until there are none left in the UART.
    while( UART_RX_REG_NOT_EMPTY() )
    {
        // Obtain the character from the UART.
        cRxedChar = UART_RX_REG;

        // Post the character onto a queue.  xCRWokenByPost will be pdFALSE
        // the first time around the loop.  If the post causes a co-routine
        // to be woken (unblocked) then xCRWokenByPost will be set to pdTRUE.
        // In this manner we can ensure that if more than one co-routine is
        // blocked on the queue only one is woken by this ISR no matter how
        // many characters are posted to the queue.
        xCRWokenByPost = crQUEUE_SEND_FROM_ISR( xCommsRxQueue, &cRxedChar, xCRWokenByPost );
    }
}
```

20.8 crQUEUE_RECEIVE_FROM_ISR

croutine.h

```
crQUEUE_SEND_FROM_ISR(
    xQueueHandle pxQueue,
    void *pvBuffer,
    portBASE_TYPE * pxCoRoutineWoken
)
```

The macro's `crQUEUE_SEND_FROM_ISR()` and `crQUEUE_RECEIVE_FROM_ISR()` are the co-routine equivalent to the `xQueueSendFromISR()` and `xQueueReceiveFromISR()` functions used by tasks.

`crQUEUE_SEND_FROM_ISR()` and `crQUEUE_RECEIVE_FROM_ISR()` can only be used to pass data between a co-routine and an ISR, whereas `xQueueSendFromISR()` and `xQueueReceiveFromISR()` can only be used to pass data between a task and an ISR.

`crQUEUE_RECEIVE_FROM_ISR` can only be called from an ISR to receive data from a queue that is being used from within a co-routine (a co-routine posted to the queue).

See the co-routine section of the WEB documentation for information on passing data between tasks and co-routines and between ISR's and co-routines.

Parameters

<code>xQueue</code>	The handle to the queue on which the item is to be posted.
<code>pvBuffer</code>	A pointer to a buffer into which the received item will be placed. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from the queue into <code>pvBuffer</code> .
<code>pxCoRoutineWoken</code>	A co-routine may be blocked waiting for space to become available on the queue. If <code>crQUEUE_RECEIVE_FROM_ISR</code> causes such a co-routine to unblock <code>*pxCoRoutineWoken</code> will get set to <code>pdTRUE</code> , otherwise <code>*pxCoRoutineWoken</code> will remain unchanged.

Returns

`pdTRUE` if an item was successfully received from the queue, otherwise `pdFALSE`.

Example usage:

```
// A co-routine that posts a character to a queue then blocks for a fixed
// period. The character is incremented each time.
static void vSendingCoRoutine( xCoRoutineHandle xHandle, unsigned portBASE_TYPE uxIndex )
{
// cChar holds its value while this co-routine is blocked and must therefore
// be declared static.
static char cCharToTx = 'a';
portBASE_TYPE xResult;

// All co-routines must start with a call to crSTART().
crSTART( xHandle );

for( ;; )
{
    // Send the next character to the queue.
    crQUEUE_SEND( xHandle, xCoRoutineQueue, &cCharToTx, NO_DELAY, &xResult );

    if( xResult == pdPASS )
    {
        // The character was successfully posted to the queue.
    }
}
```

```

    else
    {
        // Could not post the character to the queue.
    }

    // Enable the UART Tx interrupt to cause an interrupt in this
    // hypothetical UART. The interrupt will obtain the character
    // from the queue and send it.
    ENABLE_RX_INTERRUPT();

    // Increment to the next character then block for a fixed period.
    // cCharToTx will maintain its value across the delay as it is
    // declared static.
    cCharToTx++;
    if( cCharToTx > 'x' )
    {
        cCharToTx = 'a';
    }
    crDELAY( 100 );
}

// All co-routines must end with a call to crEND().
crEND();
}

// An ISR that uses a queue to receive characters to send on a UART.
void vUART_ISR( void )
{
char cCharToTx;
portBASE_TYPE xCRWokenByPost = pdFALSE;

while( UART_TX_REG_EMPTY() )
{
    // Are there any characters in the queue waiting to be sent?
    // xCRWokenByPost will automatically be set to pdTRUE if a co-routine
    // is woken by the post - ensuring that only a single co-routine is
    // woken no matter how many times we go around this loop.
    if( crQUEUE_RECEIVE_FROM_ISR( pxQueue, &cCharToTx, &xCRWokenByPost ) )
    {
        SEND_CHARACTER( cCharToTx );
    }
}
}
}

```

20.9 xQueueCreate

queue.h

```

xQueueHandle xQueueCreate(
                        unsigned portBASE_TYPE uxQueueLength,
                        unsigned portBASE_TYPE uxItemSize
                    );

```

Creates a new queue instance. This allocates the storage required by the new queue and returns a handle for the queue.

Parameters

<i>uxQueueLength</i>	The maximum number of items that the queue can contain.
----------------------	---

Parameters

<i>uxItemSize</i>	The number of bytes each item in the queue will require. Items are queued by copy, not by reference, so this is the number of bytes that will be copied for each posted item. Each item on the queue must be the same size.
-------------------	---

Returns

If the queue is successfully created then a handle to the newly created queue is returned. If the queue cannot be created then 0 is returned.

Example usage:

```
struct AMessage
{
    char ucMessageID;
    char ucData[ 20 ];
};

void vATask( void *pvParameters )
{
    xQueueHandle xQueue1, xQueue2;

    // Create a queue capable of containing 10 unsigned long values.
    xQueue1 = xQueueCreate( 10, sizeof( unsigned long ) );
    if( xQueue1 == 0 )
    {
        // Queue was not created and must not be used.
    }

    // Create a queue capable of containing 10 pointers to AMessage structures.
    // These should be passed by pointer as they contain a lot of data.
    xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );
    if( xQueue2 == 0 )
    {
        // Queue was not created and must not be used.
    }

    // ... Rest of task code.
}
```

20.10 xQueueSend

queue.h

```
portBASE_TYPE xQueueSendToFront(
    xQueueHandle xQueue,
    const void    * pvItemToQueue,
    portTickType xTicksToWait
);
```

This is a macro that calls [xQueueGenericSend\(\)](#).

Post an item to the front of a queue. The item is queued by copy, not by reference. This function must not be called from an interrupt service routine. See [xQueueSendFromISR\(\)](#) for an alternative which may be used in an ISR.

Parameters

<i>xQueue</i>	The handle to the queue on which the item is to be posted.
<i>pvlItemToQueue</i>	A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from <i>pvlItemToQueue</i> into the queue storage area.
<i>xTicksToWait</i>	The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if this is set to 0 and the queue is full. The time is defined in tick periods so the constant <i>portTICK_RATE_MS</i> should be used to convert to real time if this is required.

Returns

pdTRUE if the item was successfully posted, otherwise *errQUEUE_FULL*.

Example usage:

```

struct AMessage
{
    char ucMessageID;
    char ucData[ 20 ];
} xMessage;

unsigned long ulVar = 10UL;

void vATask( void *pvParameters )
{
    xQueueHandle xQueue1, xQueue2;
    struct AMessage *pxMessage;

    // Create a queue capable of containing 10 unsigned long values.
    xQueue1 = xQueueCreate( 10, sizeof( unsigned long ) );

    // Create a queue capable of containing 10 pointers to AMessage structures.
    // These should be passed by pointer as they contain a lot of data.
    xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );

    // ...

    if( xQueue1 != 0 )
    {
        // Send an unsigned long. Wait for 10 ticks for space to become
        // available if necessary.
        if( xQueueSendToFront( xQueue1, ( void * ) &ulVar, ( portTickType ) 10 ) != pdPASS )
        {
            // Failed to post the message, even after 10 ticks.
        }
    }

    if( xQueue2 != 0 )
    {
        // Send a pointer to a struct AMessage object. Don't block if the
        // queue is already full.
        pxMessage = & xMessage;
        xQueueSendToFront( xQueue2, ( void * ) &pxMessage, ( portTickType ) 0 );
    }

    // ... Rest of task code.
}

```

queue.h

```
portBASE_TYPE xQueueSendToBack(
    xQueueHandle xQueue,
    const void * pvItemToQueue,
    portTickType xTicksToWait
);
```

This is a macro that calls [xQueueGenericSend\(\)](#).

Post an item to the back of a queue. The item is queued by copy, not by reference. This function must not be called from an interrupt service routine. See [xQueueSendFromISR\(\)](#) for an alternative which may be used in an ISR.

Parameters

<i>xQueue</i>	The handle to the queue on which the item is to be posted.
<i>pvItemToQueue</i>	A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from <i>pvItemToQueue</i> into the queue storage area.
<i>xTicksToWait</i>	The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if this is set to 0 and the queue is full. The time is defined in tick periods so the constant portTICK_RATE_MS should be used to convert to real time if this is required.

Returns

`pdTRUE` if the item was successfully posted, otherwise `errQUEUE_FULL`.

Example usage:

```
struct AMesssage
{
    char ucMessageID;
    char ucData[ 20 ];
} xMessage;

unsigned long ulVar = 10UL;

void vATask( void *pvParameters )
{
    xQueueHandle xQueue1, xQueue2;
    struct AMesssage *pxMessage;

    // Create a queue capable of containing 10 unsigned long values.
    xQueue1 = xQueueCreate( 10, sizeof( unsigned long ) );

    // Create a queue capable of containing 10 pointers to AMesssage structures.
    // These should be passed by pointer as they contain a lot of data.
    xQueue2 = xQueueCreate( 10, sizeof( struct AMesssage * ) );

    // ...

    if( xQueue1 != 0 )
    {
        // Send an unsigned long. Wait for 10 ticks for space to become
        // available if necessary.
```

```

if( xQueueSendToBack( xQueue1, ( void * ) &ulVar, ( portTickType ) 10 ) != pdPASS )
{
    // Failed to post the message, even after 10 ticks.
}

if( xQueue2 != 0 )
{
    // Send a pointer to a struct AMessage object.  Don't block if the
    // queue is already full.
    pxMessage = & xMessage;
    xQueueSendToBack( xQueue2, ( void * ) &pxMessage, ( portTickType ) 0 );
}

// ... Rest of task code.
}

```

queue.h

```

portBASE_TYPE xQueueSend(
    xQueueHandle xQueue,
    const void * pvItemToQueue,
    portTickType xTicksToWait
);

```

This is a macro that calls [xQueueGenericSend\(\)](#). It is included for backward compatibility with versions of FreeRTOS.org that did not include the [xQueueSendToFront\(\)](#) and [xQueueSendToBack\(\)](#) macros. It is equivalent to [xQueueSendToBack\(\)](#).

Post an item on a queue. The item is queued by copy, not by reference. This function must not be called from an interrupt service routine. See [xQueueSendFromISR\(\)](#) for an alternative which may be used in an ISR.

Parameters

<i>xQueue</i>	The handle to the queue on which the item is to be posted.
<i>pvItemToQueue</i>	A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from <i>pvItemToQueue</i> into the queue storage area.
<i>xTicksToWait</i>	The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if this is set to 0 and the queue is full. The time is defined in tick periods so the constant <i>portTICK_RATE_MS</i> should be used to convert to real time if this is required.

Returns

pdTRUE if the item was successfully posted, otherwise *errQUEUE_FULL*.

Example usage:

```

struct AMessage
{
    char ucMessageID;
    char ucData[ 20 ];
} xMessage;

```

```

unsigned long ulVar = 10UL;

void vATask( void *pvParameters )
{
xQueueHandle xQueue1, xQueue2;
struct AMessage *pxMessage;

// Create a queue capable of containing 10 unsigned long values.
xQueue1 = xQueueCreate( 10, sizeof( unsigned long ) );

// Create a queue capable of containing 10 pointers to AMessage structures.
// These should be passed by pointer as they contain a lot of data.
xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );

// ...

if( xQueue1 != 0 )
{
    // Send an unsigned long. Wait for 10 ticks for space to become
    // available if necessary.
    if( xQueueSend( xQueue1, ( void * ) &ulVar, ( portTickType ) 10 ) != pdPASS )
    {
        // Failed to post the message, even after 10 ticks.
    }
}

if( xQueue2 != 0 )
{
    // Send a pointer to a struct AMessage object. Don't block if the
    // queue is already full.
    pxMessage = & xMessage;
    xQueueSend( xQueue2, ( void * ) &pxMessage, ( portTickType ) 0 );
}

// ... Rest of task code.
}

```

queue.h

```

portBASE_TYPE xQueueGenericSend(
                                xQueueHandle xQueue,
                                const void * pvItemToQueue,
                                portTickType xTicksToWait
                                portBASE_TYPE xCopyPosition
                                );

```

It is preferred that the macros [xQueueSend\(\)](#), [xQueueSendToFront\(\)](#) and [xQueueSendToBack\(\)](#) are used in place of calling this function directly.

Post an item on a queue. The item is queued by copy, not by reference. This function must not be called from an interrupt service routine. See [xQueueSendFromISR \(\)](#) for an alternative which may be used in an ISR.

Parameters

<i>xQueue</i>	The handle to the queue on which the item is to be posted.
<i>pvItemToQueue</i>	A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from <i>pvItemToQueue</i> into the queue storage area.

Parameters

<i>xTicksToWait</i>	The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if this is set to 0 and the queue is full. The time is defined in tick periods so the constant portTICK_RATE_MS should be used to convert to real time if this is required.
<i>xCopyPosition</i>	Can take the value queueSEND_TO_BACK to place the item at the back of the queue, or queueSEND_TO_FRONT to place the item at the front of the queue (for high priority messages).

Returns

pdTRUE if the item was successfully posted, otherwise errQUEUE_FULL.

Example usage:

```

struct AMesssage
{
    char ucMessageID;
    char ucData[ 20 ];
} xMessage;

unsigned long ulVar = 10UL;

void vATask( void *pvParameters )
{
    xQueueHandle xQueue1, xQueue2;
    struct AMesssage *pxMessage;

    // Create a queue capable of containing 10 unsigned long values.
    xQueue1 = xQueueCreate( 10, sizeof( unsigned long ) );

    // Create a queue capable of containing 10 pointers to AMesssage structures.
    // These should be passed by pointer as they contain a lot of data.
    xQueue2 = xQueueCreate( 10, sizeof( struct AMesssage * ) );

    // ...

    if( xQueue1 != 0 )
    {
        // Send an unsigned long. Wait for 10 ticks for space to become
        // available if necessary.
        if( xQueueGenericSend( xQueue1, ( void * ) &ulVar, ( portTickType ) 10, queueSEND_TO_BACK ) != pdTRUE )
        {
            // Failed to post the message, even after 10 ticks.
        }
    }

    if( xQueue2 != 0 )
    {
        // Send a pointer to a struct AMesssage object. Don't block if the
        // queue is already full.
        pxMessage = & xMessage;
        xQueueGenericSend( xQueue2, ( void * ) &pxMessage, ( portTickType ) 0, queueSEND_TO_BACK );
    }

    // ... Rest of task code.
}

```

20.11 xQueueReceive

queue.h

```
portBASE_TYPE xQueuePeek(
    xQueueHandle xQueue,
    void *pvBuffer,
    portTickType xTicksToWait
);
```

This is a macro that calls the [xQueueGenericReceive\(\)](#) function.

Receive an item from a queue without removing the item from the queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

Successfully received items remain on the queue so will be returned again by the next call, or a call to [xQueueReceive\(\)](#).

This macro must not be used in an interrupt service routine.

Parameters

<i>pxQueue</i>	The handle to the queue from which the item is to be received.
<i>pvBuffer</i>	Pointer to the buffer into which the received item will be copied.
<i>xTicksToWait</i>	The maximum amount of time the task should block waiting for an item to receive should the queue be empty at the time of the call. The time is defined in tick periods so the constant <code>portTICK_RATE_MS</code> should be used to convert to real time if this is required. xQueuePeek() will return immediately if <i>xTicksToWait</i> is 0 and the queue is empty.

Returns

`pdTRUE` if an item was successfully received from the queue, otherwise `pdFALSE`.

Example usage:

```
struct AMesssage
{
    char ucMessageID;
    char ucData[ 20 ];
} xMessage;

xQueueHandle xQueue;

// Task to create a queue and post a value.
void vATask( void *pvParameters )
{
    struct AMesssage *pxMessage;

    // Create a queue capable of containing 10 pointers to AMesssage structures.
    // These should be passed by pointer as they contain a lot of data.
    xQueue = xQueueCreate( 10, sizeof( struct AMesssage * ) );
    if( xQueue == 0 )
    {
        // Failed to create the queue.
```

```

}

// ...

// Send a pointer to a struct AMessage object.  Don't block if the
// queue is already full.
pxMessage = & xMessage;
xQueueSend( xQueue, ( void * ) &pxMessage, ( portTickType ) 0 );

// ... Rest of task code.
}

// Task to peek the data from the queue.
void vADifferentTask( void *pvParameters )
{
struct AMessage *pxRxedMessage;

if( xQueue != 0 )
{
    // Peek a message on the created queue.  Block for 10 ticks if a
    // message is not immediately available.
    if( xQueuePeek( xQueue, &( pxRxedMessage ), ( portTickType ) 10 ) )
    {
        // pcRxedMessage now points to the struct AMessage variable posted
        // by vATask, but the item still remains on the queue.
    }
}

// ... Rest of task code.
}

```

queue.h

```

portBASE_TYPE xQueueReceive(
    xQueueHandle xQueue,
    void *pvBuffer,
    portTickType xTicksToWait
);

```

This is a macro that calls the [xQueueGenericReceive\(\)](#) function.

Receive an item from a queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

Successfully received items are removed from the queue.

This function must not be used in an interrupt service routine. See [xQueueReceiveFromISR](#) for an alternative that can.

Parameters

<i>pxQueue</i>	The handle to the queue from which the item is to be received.
<i>pvBuffer</i>	Pointer to the buffer into which the received item will be copied.
<i>xTicksToWait</i>	The maximum amount of time the task should block waiting for an item to receive should the queue be empty at the time of the call. xQueueReceive() will return immediately if <i>xTicksToWait</i> is zero and the queue is empty. The time is defined in tick periods so the constant portTICK_RATE_MS should be used to convert to real time if this is required.

Returns

pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

Example usage:

```

struct AMessage
{
    char ucMessageID;
    char ucData[ 20 ];
} xMessage;

xQueueHandle xQueue;

// Task to create a queue and post a value.
void vATask( void *pvParameters )
{
    struct AMessage *pxMessage;

    // Create a queue capable of containing 10 pointers to AMessage structures.
    // These should be passed by pointer as they contain a lot of data.
    xQueue = xQueueCreate( 10, sizeof( struct AMessage * ) );
    if( xQueue == 0 )
    {
        // Failed to create the queue.
    }

    // ...

    // Send a pointer to a struct AMessage object.  Don't block if the
    // queue is already full.
    pxMessage = & xMessage;
    xQueueSend( xQueue, ( void * ) &pxMessage, ( portTickType ) 0 );

    // ... Rest of task code.
}

// Task to receive from the queue.
void vADifferentTask( void *pvParameters )
{
    struct AMessage *pxRxedMessage;

    if( xQueue != 0 )
    {
        // Receive a message on the created queue.  Block for 10 ticks if a
        // message is not immediately available.
        if( xQueueReceive( xQueue, &( pxRxedMessage ), ( portTickType ) 10 ) )
        {
            // pcRxedMessage now points to the struct AMessage variable posted
            // by vATask.
        }
    }

    // ... Rest of task code.
}

queue.h

portBASE_TYPE xQueueGenericReceive(
                                xQueueHandle xQueue,
                                void *pvBuffer,
                                portTickType xTicksToWait
                                portBASE_TYPE      xJustPeek
                                );

```

It is preferred that the macro [xQueueReceive\(\)](#) be used rather than calling this function directly.

Receive an item from a queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

This function must not be used in an interrupt service routine. See [xQueueReceiveFromISR](#) for an alternative that can.

Parameters

<i>pxQueue</i>	The handle to the queue from which the item is to be received.
<i>pvBuffer</i>	Pointer to the buffer into which the received item will be copied.
<i>xTicksToWait</i>	The maximum amount of time the task should block waiting for an item to receive should the queue be empty at the time of the call. The time is defined in tick periods so the constant <code>portTICK_RATE_MS</code> should be used to convert to real time if this is required. xQueueGenericReceive() will return immediately if the queue is empty and <i>xTicksToWait</i> is 0.
<i>xJustPeek</i>	When set to true, the item received from the queue is not actually removed from the queue - meaning a subsequent call to xQueueReceive() will return the same item. When set to false, the item being received from the queue is also removed from the queue.

Returns

`pdTRUE` if an item was successfully received from the queue, otherwise `pdFALSE`.

Example usage:

```

struct AMessage
{
    char ucMessageID;
    char ucData[ 20 ];
} xMessage;

xQueueHandle xQueue;

// Task to create a queue and post a value.
void vATask( void *pvParameters )
{
    struct AMessage *pxMessage;

    // Create a queue capable of containing 10 pointers to AMessage structures.
    // These should be passed by pointer as they contain a lot of data.
    xQueue = xQueueCreate( 10, sizeof( struct AMessage * ) );
    if( xQueue == 0 )
    {
        // Failed to create the queue.
    }

    // ...

    // Send a pointer to a struct AMessage object. Don't block if the
    // queue is already full.
    pxMessage = & xMessage;
    xQueueSend( xQueue, ( void * ) &pxMessage, ( portTickType ) 0 );

    // ... Rest of task code.
}

// Task to receive from the queue.

```

```
void vADifferentTask( void *pvParameters )
{
    struct AMessage *pxRxedMessage;

    if( xQueue != 0 )
    {
        // Receive a message on the created queue. Block for 10 ticks if a
        // message is not immediately available.
        if( xQueueGenericReceive( xQueue, &( pxRxedMessage ), ( portTickType ) 10 ) )
        {
            // pcRxedMessage now points to the struct AMessage variable posted
            // by vATask.
        }
    }

    // ... Rest of task code.
}
```

20.12 xQueueSendFromISR

queue.h

```
portBASE_TYPE xQueueSendToFrontFromISR(
    xQueueHandle pxQueue,
    const void *pvItemToQueue,
    portBASE_TYPE *pxHigherPriorityTaskWoken
);
```

This is a macro that calls [xQueueGenericSendFromISR\(\)](#).

Post an item to the front of a queue. It is safe to use this macro from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Parameters

<i>xQueue</i>	The handle to the queue on which the item is to be posted.
<i>pvItemToQueue</i>	A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from <i>pvItemToQueue</i> into the queue storage area.
<i>pxHigherPriorityTaskWoken</i>	xQueueSendToFrontFromISR() will set <i>*pxHigherPriorityTaskWoken</i> to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueSendToFrontFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

Returns

pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

Example usage for buffered IO (where the ISR can obtain more than one value per call):

```
void vBufferISR( void )
```

```

{
char cIn;
portBASE_TYPE xHigherPriorityTaskWoken;

// We have not woken a task at the start of the ISR.
xHigherPriorityTaskWoken = pdFALSE;

// Loop until the buffer is empty.
do
{
    // Obtain a byte from the buffer.
    cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );

    // Post the byte.
    xQueueSendToFrontFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWoken );

} while( portINPUT_BYTE( BUFFER_COUNT ) );

// Now the buffer is empty we can switch context if necessary.
if( xHigherPriorityTaskWoken )
{
    taskYIELD();
}
}

```

queue. h

```

portBASE_TYPE xQueueSendToBackFromISR(
    xQueueHandle pxQueue,
    const void *pvItemToQueue,
    portBASE_TYPE *pxHigherPriorityTaskWoken
);

```

This is a macro that calls [xQueueGenericSendFromISR\(\)](#).

Post an item to the back of a queue. It is safe to use this macro from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Parameters

<i>xQueue</i>	The handle to the queue on which the item is to be posted.
<i>pvItemToQueue</i>	A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from <i>pvItemToQueue</i> into the queue storage area.
<i>pxHigherPriorityTaskWoken</i>	xQueueSendToBackFromISR() will set <i>*pxHigherPriorityTaskWoken</i> to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueSendToBackFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

Returns

pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

Example usage for buffered IO (where the ISR can obtain more than one value per call):

```

void vBufferISR( void )
{
char cIn;
portBASE_TYPE xHigherPriorityTaskWoken;

// We have not woken a task at the start of the ISR.
xHigherPriorityTaskWoken = pdFALSE;

// Loop until the buffer is empty.
do
{
    // Obtain a byte from the buffer.
    cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );

    // Post the byte.
    xQueueSendToBackFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWoken );

} while( portINPUT_BYTE( BUFFER_COUNT ) );

// Now the buffer is empty we can switch context if necessary.
if( xHigherPriorityTaskWoken )
{
    taskYIELD();
}
}

```

queue.h

```

portBASE_TYPE xQueueSendFromISR(
                                xQueueHandle pxQueue,
                                const void *pvItemToQueue,
                                portBASE_TYPE *pxHigherPriorityTaskWoken
                                );

```

This is a macro that calls [xQueueGenericSendFromISR\(\)](#). It is included for backward compatibility with versions of FreeRTOS.org that did not include the [xQueueSendToBackFromISR\(\)](#) and [xQueueSendToFrontFromISR\(\)](#) macros.

Post an item to the back of a queue. It is safe to use this function from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Parameters

<i>xQueue</i>	The handle to the queue on which the item is to be posted.
<i>pvItemToQueue</i>	A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from <i>pvItemToQueue</i> into the queue storage area.
<i>pxHigherPriorityTaskWoken</i>	xQueueSendFromISR() will set <i>*pxHigherPriorityTaskWoken</i> to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueSendFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

Returns

pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

Example usage for buffered IO (where the ISR can obtain more than one value per call):

```
void vBufferISR( void )
{
    char cIn;
    portBASE_TYPE xHigherPriorityTaskWoken;

    // We have not woken a task at the start of the ISR.
    xHigherPriorityTaskWoken = pdFALSE;

    // Loop until the buffer is empty.
    do
    {
        // Obtain a byte from the buffer.
        cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );

        // Post the byte.
        xQueueSendFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWoken );

    } while( portINPUT_BYTE( BUFFER_COUNT ) );

    // Now the buffer is empty we can switch context if necessary.
    if( xHigherPriorityTaskWoken )
    {
        // Actual macro used here is port specific.
        taskYIELD_FROM_ISR();
    }
}
```

queue.h

```
portBASE_TYPE xQueueGenericSendFromISR(
    xQueueHandle pxQueue,
    const void *pvItemToQueue,
    portBASE_TYPE *pxHigherPriorityTaskWoken,
    portBASE_TYPE xCopyPosition
);
```

It is preferred that the macros [xQueueSendFromISR\(\)](#), [xQueueSendToFrontFromISR\(\)](#) and [xQueueSendToBackFromISR\(\)](#) be used in place of calling this function directly.

Post an item on a queue. It is safe to use this function from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Parameters

<i>xQueue</i>	The handle to the queue on which the item is to be posted.
<i>pvItemToQueue</i>	A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from <i>pvItemToQueue</i> into the queue storage area.
<i>pxHigherPriorityTaskWoken</i>	xQueueGenericSendFromISR() will set <i>*pxHigherPriorityTaskWoken</i> to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueGenericSendFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.
<i>xCopyPosition</i>	Can take the value queueSEND_TO_BACK to place the item at the back of the queue, or queueSEND_TO_FRONT to place the item at the front of the queue (for high priority messages).

Returns

pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

Example usage for buffered IO (where the ISR can obtain more than one value per call):

```
void vBufferISR( void )
{
    char cIn;
    portBASE_TYPE xHigherPriorityTaskWokenByPost;

    // We have not woken a task at the start of the ISR.
    xHigherPriorityTaskWokenByPost = pdFALSE;

    // Loop until the buffer is empty.
    do
    {
        // Obtain a byte from the buffer.
        cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );

        // Post each byte.
        xQueueGenericSendFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWokenByPost, queueSEND_TO_BACK );

        } while( portINPUT_BYTE( BUFFER_COUNT ) );

        // Now the buffer is empty we can switch context if necessary. Note that the
        // name of the yield function required is port specific.
        if( xHigherPriorityTaskWokenByPost )
        {
            taskYIELD_YIELD_FROM_ISR();
        }
}
```

20.13 xQueueReceiveFromISR

queue.h

```
portBASE_TYPE xQueueReceiveFromISR(
    xQueueHandle pxQueue,
    void *pvBuffer,
    portBASE_TYPE *pxTaskWoken
);
```

Receive an item from a queue. It is safe to use this function from within an interrupt service routine.

Parameters

<i>pxQueue</i>	The handle to the queue from which the item is to be received.
<i>pvBuffer</i>	Pointer to the buffer into which the received item will be copied.
<i>pxTaskWoken</i>	A task may be blocked waiting for space to become available on the queue. If xQueueReceiveFromISR causes such a task to unblock *pxTaskWoken will get set to pdTRUE, otherwise *pxTaskWoken will remain unchanged.

Returns

pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

Example usage:

```
xQueueHandle xQueue;

// Function to create a queue and post some values.
void vAFunction( void *pvParameters )
{
char cValueToPost;
const portTickType xBlockTime = ( portTickType )0xff;

// Create a queue capable of containing 10 characters.
xQueue = xQueueCreate( 10, sizeof( char ) );
if( xQueue == 0 )
{
    // Failed to create the queue.
}

// ...

// Post some characters that will be used within an ISR.  If the queue
// is full then this task will block for xBlockTime ticks.
cValueToPost = 'a';
xQueueSend( xQueue, ( void * ) &cValueToPost, xBlockTime );
cValueToPost = 'b';
xQueueSend( xQueue, ( void * ) &cValueToPost, xBlockTime );

// ... keep posting characters ... this task may block when the queue
// becomes full.

cValueToPost = 'c';
xQueueSend( xQueue, ( void * ) &cValueToPost, xBlockTime );
}

// ISR that outputs all the characters received on the queue.
void vISR_Routine( void )
{
portBASE_TYPE xTaskWokenByReceive = pdFALSE;
char cRxedChar;

while( xQueueReceiveFromISR( xQueue, ( void * ) &cRxedChar, &xTaskWokenByReceive ) )
{
    // A character was received.  Output the character now.
    vOutputCharacter( cRxedChar );

    // If removing the character from the queue woke the task that was
    // posting onto the queue cTaskWokenByReceive will have been set to
    // pdTRUE.  No matter how many times this loop iterates only one
    // task will be woken.
}

if( cTaskWokenByPost != ( char ) pdFALSE;
{
    taskYIELD();
}
}
```

20.14 vSemaphoreCreateBinary

semphr.h

```
vSemaphoreCreateBinary( xSemaphoreHandle xSemaphore )
```

Macro that implements a semaphore by using the existing queue mechanism. The queue length is 1 as this is a binary semaphore. The data size is 0 as we don't want to actually store any data - we just want to know if the queue is empty or full.

This type of semaphore can be used for pure synchronisation between tasks or between an interrupt and a task. The semaphore need not be given back once obtained, so one task/interrupt can continuously 'give' the semaphore while another continuously 'takes' the semaphore. For this reason this type of semaphore does not use a priority inheritance mechanism. For an alternative that does use priority inheritance see [xSemaphoreCreateMutex\(\)](#).

Parameters

<i>xSemaphore</i>	Handle to the created semaphore. Should be of type xSemaphoreHandle.
-------------------	--

Example usage:

```
xSemaphoreHandle xSemaphore;

void vATask( void * pvParameters )
{
    // Semaphore cannot be used before a call to vSemaphoreCreateBinary () .
    // This is a macro so pass the variable in directly.
    vSemaphoreCreateBinary( xSemaphore );

    if( xSemaphore != NULL )
    {
        // The semaphore was created successfully.
        // The semaphore can now be used.
    }
}
```

20.15 xSemaphoreTake

semphr.h

```
xSemaphoreTake(
    xSemaphoreHandle xSemaphore,
    portTickType xBlockTime
)
```

Macro to obtain a semaphore. The semaphore must have previously been created with a call to [vSemaphoreCreateBinary\(\)](#), [xSemaphoreCreateMutex\(\)](#) or [xSemaphoreCreateCounting\(\)](#).

Parameters

<i>xSemaphore</i>	A handle to the semaphore being taken - obtained when the semaphore was created.
<i>xBlockTime</i>	The time in ticks to wait for the semaphore to become available. The macro portTICK_RATE_MS can be used to convert this to a real time. A block time of zero can be used to poll the semaphore. A block time of portMAX_DELAY can be used to block indefinitely (provided INCLUDE_vTaskSuspend is set to 1 in FreeRTOSConfig.h).

Returns

pdTRUE if the semaphore was obtained. pdFALSE if xBlockTime expired without the semaphore becoming available.

Example usage:

```
xSemaphoreHandle xSemaphore = NULL;

// A task that creates a semaphore.
void vATask( void * pvParameters )
{
    // Create the semaphore to guard a shared resource.
    vSemaphoreCreateBinary( xSemaphore );
}

// A task that uses the semaphore.
void vAnotherTask( void * pvParameters )
{
    // ... Do other things.

    if( xSemaphore != NULL )
    {
        // See if we can obtain the semaphore. If the semaphore is not available
        // wait 10 ticks to see if it becomes free.
        if( xSemaphoreTake( xSemaphore, ( portTickType ) 10 ) == pdTRUE )
        {
            // We were able to obtain the semaphore and can now access the
            // shared resource.

            // ...

            // We have finished accessing the shared resource. Release the
            // semaphore.
            xSemaphoreGive( xSemaphore );
        }
        else
        {
            // We could not obtain the semaphore and can therefore not access
            // the shared resource safely.
        }
    }
}
```

20.16 xSemaphoreTakeRecursive

semphr.h xSemaphoreTakeRecursive(xSemaphoreHandle xMutex, portTickType xBlockTime)

Macro to recursively obtain, or 'take', a mutex type semaphore.

The mutex must have previously been created using a call to [xSemaphoreCreateRecursiveMutex\(\)](#);

configUSE_RECURSIVE_MUTEXES must be set to 1 in [FreeRTOSConfig.h](#) for this macro to be available.

This macro must not be used on mutexes created using [xSemaphoreCreateMutex\(\)](#).

A mutex used recursively can be 'taken' repeatedly by the owner. The mutex doesn't become available again until the owner has called [xSemaphoreGiveRecursive\(\)](#) for each successful 'take' request. For example, if a task successfully 'takes' the same mutex 5 times then the mutex will not be available to any other task until it has also 'given' the mutex back exactly five times.

Parameters

<code>xMutex</code>	A handle to the mutex being obtained. This is the handle returned by <code>xSemaphoreCreateRecursiveMutex()</code> ;
<code>xBlockTime</code>	The time in ticks to wait for the semaphore to become available. The macro <code>portTICK_RATE_MS</code> can be used to convert this to a real time. A block time of zero can be used to poll the semaphore. If the task already owns the semaphore then <code>xSemaphoreTakeRecursive()</code> will return immediately no matter what the value of <code>xBlockTime</code> .

Returns

`pdTRUE` if the semaphore was obtained. `pdFALSE` if `xBlockTime` expired without the semaphore becoming available.

Example usage:

```

xSemaphoreHandle xMutex = NULL;

// A task that creates a mutex.
void vATask( void * pvParameters )
{
    // Create the mutex to guard a shared resource.
    xMutex = xSemaphoreCreateRecursiveMutex();
}

// A task that uses the mutex.
void vAnotherTask( void * pvParameters )
{
    // ... Do other things.

    if( xMutex != NULL )
    {
        // See if we can obtain the mutex.  If the mutex is not available
        // wait 10 ticks to see if it becomes free.
        if( xSemaphoreTakeRecursive( xSemaphore, ( portTickType ) 10 ) == pdTRUE )
        {
            // We were able to obtain the mutex and can now access the
            // shared resource.

            // ...
            // For some reason due to the nature of the code further calls to
            // xSemaphoreTakeRecursive() are made on the same mutex.  In real
            // code these would not be just sequential calls as this would make
            // no sense.  Instead the calls are likely to be buried inside
            // a more complex call structure.
            xSemaphoreTakeRecursive( xMutex, ( portTickType ) 10 );
            xSemaphoreTakeRecursive( xMutex, ( portTickType ) 10 );

            // The mutex has now been 'taken' three times, so will not be
            // available to another task until it has also been given back
            // three times. Again it is unlikely that real code would have
            // these calls sequentially, but instead buried in a more complex
            // call structure. This is just for illustrative purposes.
            xSemaphoreGiveRecursive( xMutex );
            xSemaphoreGiveRecursive( xMutex );
            xSemaphoreGiveRecursive( xMutex );

            // Now the mutex can be taken by other tasks.
        }
    }
}

```

```

        else
        {
            // We could not obtain the mutex and can therefore not access
            // the shared resource safely.
        }
    }
}

```

20.17 xSemaphoreGive

semphr.h

`xSemaphoreGive(xSemaphoreHandle xSemaphore)`

Macro to release a semaphore. The semaphore must have previously been created with a call to `vSemaphoreCreateBinary()`, `xSemaphoreCreateMutex()` or `xSemaphoreCreateCounting()`. and obtained using `sSemaphoreTake()`.

This macro must not be used from an ISR. See `xSemaphoreGiveFromISR()` for an alternative which can be used from an ISR.

This macro must also not be used on semaphores created using `xSemaphoreCreateRecursiveMutex()`.

Parameters

<code>xSemaphore</code>	A handle to the semaphore being released. This is the handle returned when the semaphore was created.
-------------------------	---

Returns

`pdTRUE` if the semaphore was released. `pdFALSE` if an error occurred. Semaphores are implemented using queues. An error can occur if there is no space on the queue to post a message - indicating that the semaphore was not first obtained correctly.

Example usage:

```

xSemaphoreHandle xSemaphore = NULL;

void vATask( void * pvParameters )
{
    // Create the semaphore to guard a shared resource.
    vSemaphoreCreateBinary( xSemaphore );

    if( xSemaphore != NULL )
    {
        if( xSemaphoreGive( xSemaphore ) != pdTRUE )
        {
            // We would expect this call to fail because we cannot give
            // a semaphore without first "taking" it!
        }

        // Obtain the semaphore - don't block if the semaphore is not
        // immediately available.
        if( xSemaphoreTake( xSemaphore, ( portTickType ) 0 ) )

```

```
{  
    // We now have the semaphore and can access the shared resource.  
  
    // ...  
  
    // We have finished accessing the shared resource so can free the  
    // semaphore.  
    if( xSemaphoreGive( xSemaphore ) != pdTRUE )  
    {  
        // We would not expect this call to fail because we must have  
        // obtained the semaphore to get here.  
    }  
}  
}  
}
```

20.18 xSemaphoreGiveRecursive

semphr.h

`xSemaphoreGiveRecursive(xSemaphoreHandle xMutex)`

Macro to recursively release, or 'give', a mutex type semaphore. The mutex must have previously been created using a call to [xSemaphoreCreateRecursiveMutex\(\)](#);

`configUSE_RECURSIVE_MUTEXES` must be set to 1 in [FreeRTOSConfig.h](#) for this macro to be available.

This macro must not be used on mutexes created using [xSemaphoreCreateMutex\(\)](#).

A mutex used recursively can be 'taken' repeatedly by the owner. The mutex doesn't become available again until the owner has called [xSemaphoreGiveRecursive\(\)](#) for each successful 'take' request. For example, if a task successfully 'takes' the same mutex 5 times then the mutex will not be available to any other task until it has also 'given' the mutex back exactly five times.

Parameters

<code>xMutex</code>	A handle to the mutex being released, or 'given'. This is the handle returned by xSemaphoreCreateMutex() ;
---------------------	--

Returns

`pdTRUE` if the semaphore was given.

Example usage:

```
xSemaphoreHandle xMutex = NULL;  
  
// A task that creates a mutex.  
void vATask( void * pvParameters )  
{  
    // Create the mutex to guard a shared resource.  
    xMutex = xSemaphoreCreateRecursiveMutex();  
}
```

```

// A task that uses the mutex.
void vAnotherTask( void * pvParameters )
{
    // ... Do other things.

    if( xMutex != NULL )
    {
        // See if we can obtain the mutex.  If the mutex is not available
        // wait 10 ticks to see if it becomes free.
        if( xSemaphoreTakeRecursive( xMutex, ( portTickType ) 10 ) == pdTRUE )
        {
            // We were able to obtain the mutex and can now access the
            // shared resource.

            // ...
            // For some reason due to the nature of the code further calls to
            // xSemaphoreTakeRecursive() are made on the same mutex.  In real
            // code these would not be just sequential calls as this would make
            // no sense.  Instead the calls are likely to be buried inside
            // a more complex call structure.
            xSemaphoreTakeRecursive( xMutex, ( portTickType ) 10 );
            xSemaphoreTakeRecursive( xMutex, ( portTickType ) 10 );

            // The mutex has now been 'taken' three times, so will not be
            // available to another task until it has also been given back
            // three times. Again it is unlikely that real code would have
            // these calls sequentially, it would be more likely that the calls
            // to xSemaphoreGiveRecursive() would be called as a call stack
            // unwound. This is just for demonstrative purposes.
            xSemaphoreGiveRecursive( xMutex );
            xSemaphoreGiveRecursive( xMutex );
            xSemaphoreGiveRecursive( xMutex );

            // Now the mutex can be taken by other tasks.
        }
        else
        {
            // We could not obtain the mutex and can therefore not access
            // the shared resource safely.
        }
    }
}

```

20.19 xSemaphoreGiveFromISR

semphr.h

```

xSemaphoreGiveFromISR(
    xSemaphoreHandle xSemaphore,
    signed portBASE_TYPE *pxHigherPriorityTaskWoken
)

```

Macro to release a semaphore. The semaphore must have previously been created with a call to [vSemaphoreCreateBinary\(\)](#) or [xSemaphoreCreateCounting\(\)](#).

Mutex type semaphores (those created using a call to [xSemaphoreCreateMutex\(\)](#)) must not be used with this macro.

This macro can be used from an ISR.

Parameters

<i>xSemaphore</i>	A handle to the semaphore being released. This is the handle returned when the semaphore was created.
<i>pxHigherPriorityTaskWoken</i>	<i>xSemaphoreGiveFromISR()</i> will set * <i>pxHigherPriorityTaskWoken</i> to pdTRUE if giving the semaphore caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If <i>xSemaphoreGiveFromISR()</i> sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

Returns

pdTRUE if the semaphore was successfully given, otherwise errQUEUE_FULL.

Example usage:

```
#define LONG_TIME 0xffff
#define TICKS_TO_WAIT 10
xSemaphoreHandle xSemaphore = NULL;

// Repetitive task.
void vATask( void * pvParameters )
{
    for( ; ; )
    {
        // We want this task to run every 10 ticks of a timer.  The semaphore
        // was created before this task was started.

        // Block waiting for the semaphore to become available.
        if( xSemaphoreTake( xSemaphore, LONG_TIME ) == pdTRUE )
        {
            // It is time to execute.

            // ...

            // We have finished our task.  Return to the top of the loop where
            // we will block on the semaphore until it is time to execute
            // again.  Note when using the semaphore for synchronisation with an
            // ISR in this manner there is no need to 'give' the semaphore back.
        }
    }
}

// Timer ISR
void vTimerISR( void * pvParameters )
{
    static unsigned char ucLocalTickCount = 0;
    static signed portBASE_TYPE xHigherPriorityTaskWoken;

    // A timer tick has occurred.

    // ... Do other time functions.

    // Is it time for vATask () to run?
    xHigherPriorityTaskWoken = pdFALSE;
    ucLocalTickCount++;
    if( ucLocalTickCount >= TICKS_TO_WAIT )
    {
        // Unblock the task by releasing the semaphore.
    }
}
```

```

    xSemaphoreGiveFromISR( xSemaphore, &xHigherPriorityTaskWoken );

    // Reset the count so we release the semaphore again in 10 ticks time.
    ucLocalTickCount = 0;
}

if( xHigherPriorityTaskWoken != pdFALSE )
{
    // We can force a context switch here. Context switching from an
    // ISR uses port specific syntax. Check the demo task for your port
    // to find the syntax required.
}
}

```

20.20 vSemaphoreCreateMutex

semphr.h

```
xSemaphoreHandle xSemaphoreCreateMutex( void )
```

Macro that implements a mutex semaphore by using the existing queue mechanism.

Mutexes created using this macro can be accessed using the [xSemaphoreTake\(\)](#) and [xSemaphoreGive\(\)](#) macros. The [xSemaphoreTakeRecursive\(\)](#) and [xSemaphoreGiveRecursive\(\)](#) macros should not be used.

This type of semaphore uses a priority inheritance mechanism so a task 'taking' a semaphore MUST ALWAYS 'give' the semaphore back once the semaphore it is no longer required.

Mutex type semaphores cannot be used from within interrupt service routines.

See [vSemaphoreCreateBinary\(\)](#) for an alternative implementation that can be used for pure synchronisation (where one task or interrupt always 'gives' the semaphore and another always 'takes' the semaphore) and from within interrupt service routines.

Returns

xSemaphore Handle to the created mutex semaphore. Should be of type xSemaphoreHandle.

Example usage:

```

xSemaphoreHandle xSemaphore;

void vATask( void * pvParameters )
{
    // Semaphore cannot be used before a call to xSemaphoreCreateMutex() .
    // This is a macro so pass the variable in directly.
    xSemaphore = xSemaphoreCreateMutex();

    if( xSemaphore != NULL )
    {
        // The semaphore was created successfully.
        // The semaphore can now be used.
    }
}

```

semphr.h

```
xSemaphoreHandle xSemaphoreCreateRecursiveMutex( void )
```

Macro that implements a recursive mutex by using the existing queue mechanism.

Mutexes created using this macro can be accessed using the [xSemaphoreTakeRecursive\(\)](#) and [xSemaphoreGiveRecursive\(\)](#) macros. The [xSemaphoreTake\(\)](#) and [xSemaphoreGive\(\)](#) macros should not be used.

A mutex used recursively can be 'taken' repeatedly by the owner. The mutex doesn't become available again until the owner has called [xSemaphoreGiveRecursive\(\)](#) for each successful 'take' request. For example, if a task successfully 'takes' the same mutex 5 times then the mutex will not be available to any other task until it has also 'given' the mutex back exactly five times.

This type of semaphore uses a priority inheritance mechanism so a task 'taking' a semaphore MUST ALWAYS 'give' the semaphore back once the semaphore it is no longer required.

Mutex type semaphores cannot be used from within interrupt service routines.

See [vSemaphoreCreateBinary\(\)](#) for an alternative implementation that can be used for pure synchronisation (where one task or interrupt always 'gives' the semaphore and another always 'takes' the semaphore) and from within interrupt service routines.

Returns

xSemaphore Handle to the created mutex semaphore. Should be of type xSemaphoreHandle.

Example usage:

```
xSemaphoreHandle xSemaphore;

void vATask( void * pvParameters )
{
    // Semaphore cannot be used before a call to xSemaphoreCreateMutex() .
    // This is a macro so pass the variable in directly.
    xSemaphore = xSemaphoreCreateRecursiveMutex();

    if( xSemaphore != NULL )
    {
        // The semaphore was created successfully.
        // The semaphore can now be used.
    }
}
```

20.21 xSemaphoreCreateCounting

semphr.h

```
xSemaphoreHandle xSemaphoreCreateCounting( unsigned portBASE_TYPE uxMaxCount, unsigned portBASE_TY
```

Macro that creates a counting semaphore by using the existing queue mechanism.

Counting semaphores are typically used for two things:

- 1) Counting events.

In this usage scenario an event handler will 'give' a semaphore each time an event occurs (incrementing the semaphore count value), and a handler task will 'take' a semaphore each time it processes an event (decrementing the semaphore count value). The count value is therefore the difference between the number of events that have occurred and the number that have been processed. In this case it is desirable for the initial count value to be zero.

- 2) Resource management.

In this usage scenario the count value indicates the number of resources available. To obtain control of a resource a task must first obtain a semaphore - decrementing the semaphore count value. When the count value reaches zero there are no free resources. When a task finishes with the resource it 'gives' the semaphore back - incrementing the semaphore count value. In this case it is desirable for the initial count value to be equal to the maximum count value, indicating that all resources are free.

Parameters

<code>uxMaxCount</code>	The maximum count value that can be reached. When the semaphore reaches this value it can no longer be 'given'.
<code>uxInitialCount</code>	The count value assigned to the semaphore when it is created.

Returns

Handle to the created semaphore. Null if the semaphore could not be created.

Example usage:

```
xSemaphoreHandle xSemaphore;

void vATask( void * pvParameters )
{
xSemaphoreHandle xSemaphore = NULL;

    // Semaphore cannot be used before a call to xSemaphoreCreateCounting\(\).
    // The max value to which the semaphore can count should be 10, and the
    // initial value assigned to the count should be 0.
    xSemaphore = xSemaphoreCreateCounting\( 10, 0 \);

    if( xSemaphore != NULL )
    {
        // The semaphore was created successfully.
        // The semaphore can now be used.
    }
}
```

20.22 xTaskCreate

task.h

```
portBASE_TYPE xTaskCreate(
    pdTASK_CODE pvTaskCode,
    const char * const pcName,
    unsigned short usStackDepth,
    void *pvParameters,
    unsigned portBASE_TYPE uxPriority,
    xTaskHandle *pvCreatedTask
);

```

Create a new task and add it to the list of tasks that are ready to run.

[xTaskCreate\(\)](#) can only be used to create a task that has unrestricted access to the entire microcontroller memory map. Systems that include MPU support can alternatively create an MPU constrained task using [xTaskCreateRestricted\(\)](#).

Parameters

<i>pvTaskCode</i>	Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop).
<i>pcName</i>	A descriptive name for the task. This is mainly used to facilitate debugging. Max length defined by tskMAX_TASK_NAME_LEN - default is 16.
<i>usStackDepth</i>	The size of the task stack specified as the number of variables the stack can hold - not the number of bytes. For example, if the stack is 16 bits wide and usStackDepth is defined as 100, 200 bytes will be allocated for stack storage.
<i>pvParameters</i>	Pointer that will be used as the parameter for the task being created.
<i>uxPriority</i>	The priority at which the task should run. Systems that include MPU support can optionally create tasks in a privileged (system) mode by setting bit portPRIVILEGE_BIT of the priority parameter. For example, to create a privileged task at priority 2 the uxPriority parameter should be set to (2 portPRIVILEGE_BIT).
<i>pvCreatedTask</i>	Used to pass back a handle by which the created task can be referenced.

Returns

pdPASS if the task was successfully created and added to a ready list, otherwise an error code defined in the file errors.h

Example usage:

```
// Task to be created.
void vTaskCode( void * pvParameters )
{
    for( ;; )
    {
        // Task code goes here.
    }
}

// Function that creates a task.
void vOtherFunction( void )
{
    static unsigned char ucParameterToPass;
    xTaskHandle xHandle;

    // Create the task, storing the handle. Note that the passed parameter ucParameterToPass
    // must exist for the lifetime of the task, so in this case is declared static. If it was just
    // an automatic stack variable it might no longer exist, or at least have been corrupted, by t
```

```

    // the new task attempts to access it.
    xTaskCreate( vTaskCode, "NAME", STACK_SIZE, &ucParameterToPass, tskIDLE_PRIORITY, &xHandle );

    // Use the handle to delete the task.
    vTaskDelete( xHandle );
}

```

20.23 xTaskCreateRestricted

task.h

```
portBASE_TYPE xTaskCreateRestricted( xTaskParameters *pxTaskDefinition, xTaskHandle *pxCreatedTask );
```

[xTaskCreateRestricted\(\)](#) should only be used in systems that include an MPU implementation.

Create a new task and add it to the list of tasks that are ready to run. The function parameters define the memory regions and associated access permissions allocated to the task.

Parameters

<i>pxTaskDefinition</i>	Pointer to a structure that contains a member for each of the normal xTaskCreate() parameters (see the xTaskCreate() API documentation) plus an optional stack buffer and the memory region definitions.
<i>pxCreatedTask</i>	Used to pass back a handle by which the created task can be referenced.

Returns

pdPASS if the task was successfully created and added to a ready list, otherwise an error code defined in the file errors.h

Example usage:

```

// Create an xTaskParameters structure that defines the task to be created.
static const xTaskParameters xCheckTaskParameters =
{
    vATask,           // pvTaskCode - the function that implements the task.
    "ATask",          // pcName - just a text name for the task to assist debugging.
    100,              // usStackDepth - the stack size DEFINED IN WORDS.
    NULL,             // pvParameters - passed into the task function as the function parameters.
    ( 1UL | portPRIVILEGE_BIT ), // uxPriority - task priority, set the portPRIVILEGE_BIT if the task needs privileged access.
    cStackBuffer, // puxStackBuffer - the buffer to be used as the task stack.

    // xRegions - Allocate up to three separate memory regions for access by
    // the task, with appropriate access permissions. Different processors have
    // different memory alignment requirements - refer to the FreeRTOS documentation
    // for full information.
    {
        // Base address           Length   Parameters
        { cReadWriteArray,       32,      portMPU_REGION_READ_WRITE },
        { cReadOnlyArray,         32,      portMPU_REGION_READ_ONLY },
        { cPrivilegedOnlyAccessArray, 128,      portMPU_REGION_PRIVILEGED_READ_WRITE }
    }
};

```

```

int main( void )
{
xTaskHandle xHandle;

// Create a task from the const structure defined above. The task handle
// is requested (the second parameter is not NULL) but in this case just for
// demonstration purposes as its not actually used.
xTaskCreateRestricted( &xRegTest1Parameters, &xHandle );

// Start the scheduler.
vTaskStartScheduler();

// Will only get here if there was insufficient memory to create the idle
// task.
for( ;; );
}

```

task.h

```
void vTaskAllocateMPURegions( xTaskHandle xTask, const xMemoryRegion * const pxRegions );
```

Memory regions are assigned to a restricted task when the task is created by a call to [xTaskCreateRestricted\(\)](#). These regions can be redefined using [vTaskAllocateMPURegions\(\)](#).

Parameters

<i>xTask</i>	The handle of the task being updated.
<i>xRegions</i>	A pointer to an xMemoryRegion structure that contains the new memory region definitions.

Example usage:

```

// Define an array of xMemoryRegion structures that configures an MPU region
// allowing read/write access for 1024 bytes starting at the beginning of the
// ucOneKByte array. The other two of the maximum 3 definable regions are
// unused so set to zero.
static const xMemoryRegion xAltRegions[ portNUM_CONFIGURABLE_REGIONS ] =
{
    // Base address      Length      Parameters
    { ucOneKByte,      1024,       portMPU_REGION_READ_WRITE },
    { 0,               0,          0 },
    { 0,               0,          0 }
};

void vATask( void *pvParameters )
{
    // This task was created such that it has access to certain regions of
    // memory as defined by the MPU configuration. At some point it is
    // desired that these MPU regions are replaced with that defined in the
    // xAltRegions const struct above. Use a call to vTaskAllocateMPURegions\(\)
    // for this purpose. NULL is used as the task handle to indicate that this
    // function should modify the MPU regions of the calling task.
    vTaskAllocateMPURegions( NULL, xAltRegions );

    // Now the task can continue its function, but from this point on can only
    // access its stack and the ucOneKByte array (unless any other statically
    // defined or shared regions have been declared elsewhere).
}

```

20.24 vTaskDelete

task.h

```
void vTaskDelete( xTaskHandle pxTask );
```

INCLUDE_vTaskDelete must be defined as 1 for this function to be available. See the configuration section for more information.

Remove a task from the RTOS real time kernels management. The task being deleted will be removed from all ready, blocked, suspended and event lists.

NOTE: The idle task is responsible for freeing the kernel allocated memory from tasks that have been deleted. It is therefore important that the idle task is not starved of microcontroller processing time if your application makes any calls to vTaskDelete(). Memory allocated by the task code is not automatically freed, and should be freed before the task is deleted.

See the demo application file death.c for sample code that utilises vTaskDelete().

Parameters

<code>pxTask</code>	The handle of the task to be deleted. Passing NULL will cause the calling task to be deleted.
---------------------	---

Example usage:

```
void vOtherFunction( void )
{
    xTaskHandle xHandle;

    // Create the task, storing the handle.
    xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle );

    // Use the handle to delete the task.
    vTaskDelete( xHandle );
}
```

20.25 vTaskDelay

task.h

```
void vTaskDelay( portTickType xTicksToDelay );
```

Delay a task for a given number of ticks. The actual time that the task remains blocked depends on the tick rate. The constant portTICK_RATE_MS can be used to calculate real time from the tick rate - with the resolution of one tick period.

INCLUDE_vTaskDelay must be defined as 1 for this function to be available. See the configuration section for more information.

vTaskDelay() specifies a time at which the task wishes to unblock relative to the time at which vTaskDelay() is called. For example, specifying a block period of 100 ticks will cause the task to unblock 100 ticks after vTaskDelay() is called. vTaskDelay() does not therefore provide a good method of controlling the frequency of a cyclical task as the path taken through the code, as well as other task and interrupt activity, will effect the frequency at which vTaskDelay() gets called and therefore the time at which the task next executes. See vTaskDelayUntil() for an alternative API function designed to facilitate fixed frequency execution. It does this by specifying an absolute time (rather than a relative time) at which the calling task should unblock.

Parameters

<i>xTicksToDelay</i>	The amount of time, in tick periods, that the calling task should block.
----------------------	--

Example usage:

```
void vTaskFunction( void * pvParameters ) { void vTaskFunction( void * pvParameters ) { Block for 500ms. const
portTickType xDelay = 500 / portTICK_RATE_MS;
```

```
for( ;; )
{
```

Simply toggle the LED every 500ms, blocking between each toggle. vToggleLED(); vTaskDelay(xDelay); } }

20.26 vTaskDelayUntil

task.h

```
void vTaskDelayUntil( portTickType *pxPreviousWakeTime, portTickType xTimeIncrement );
```

INCLUDE_vTaskDelayUntil must be defined as 1 for this function to be available. See the configuration section for more information.

Delay a task until a specified time. This function can be used by cyclical tasks to ensure a constant execution frequency.

This function differs from vTaskDelay () in one important aspect: vTaskDelay () will cause a task to block for the specified number of ticks from the time vTaskDelay () is called. It is therefore difficult to use vTaskDelay () by itself to generate a fixed execution frequency as the time between a task starting to execute and that task calling vTaskDelay () may not be fixed [the task may take a different path though the code between calls, or may get interrupted or preempted a different number of times each time it executes].

Whereas vTaskDelay () specifies a wake time relative to the time at which the function is called, vTaskDelayUntil () specifies the absolute (exact) time at which it wishes to unblock.

The constant portTICK_RATE_MS can be used to calculate real time from the tick rate - with the resolution of one tick period.

Parameters

<i>pxPreviousWakeTime</i>	Pointer to a variable that holds the time at which the task was last unblocked. The variable must be initialised with the current time prior to its first use (see the example below). Following this the variable is automatically updated within vTaskDelayUntil () .
<i>xTimeIncrement</i>	The cycle time period. The task will be unblocked at time *pxPreviousWakeTime + xTimeIncrement. Calling vTaskDelayUntil with the same xTimeIncrement parameter value will cause the task to execute with a fixed interface period.

Example usage:

```
// Perform an action every 10 ticks.
void vTaskFunction( void * pvParameters )
```

```
{  
portTickType xLastWakeTime;  
const portTickType xFrequency = 10;  
  
    // Initialise the xLastWakeTime variable with the current time.  
    xLastWakeTime = xTaskGetTickCount ();  
    for( ;; )  
    {  
        // Wait for the next cycle.  
        vTaskDelayUntil( &xLastWakeTime, xFrequency );  
  
        // Perform action here.  
    }  
}
```

20.27 uxTaskPriorityGet

task.h

```
unsigned portBASE_TYPE uxTaskPriorityGet( xTaskHandle pxTask );
```

INCLUDE_xTaskPriorityGet must be defined as 1 for this function to be available. See the configuration section for more information.

Obtain the priority of any task.

Parameters

<i>pxTask</i>	Handle of the task to be queried. Passing a NULL handle results in the priority of the calling task being returned.
---------------	---

Returns

The priority of pxTask.

Example usage:

```
void vAFunction( void )  
{  
xTaskHandle xHandle;  
  
    // Create a task, storing the handle.  
    xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle );  
  
    // ...  
  
    // Use the handle to obtain the priority of the created task.  
    // It was created with tskIDLE_PRIORITY, but may have changed  
    // it itself.  
    if( uxTaskPriorityGet( xHandle ) != tskIDLE_PRIORITY )  
    {  
        // The task has changed it's priority.  
    }
```

```
// ...  
  
// Is our priority higher than the created task?  
if( uxTaskPriorityGet( xHandle ) < uxTaskPriorityGet( NULL ) )  
{  
    // Our priority (obtained using NULL handle) is higher.  
}  
}
```

20.28 vTaskPrioritySet

task.h

```
void vTaskPrioritySet( xTaskHandle pxTask, unsigned portBASE_TYPE uxNewPriority );
```

INCLUDE_vTaskPrioritySet must be defined as 1 for this function to be available. See the configuration section for more information.

Set the priority of any task.

A context switch will occur before the function returns if the priority being set is higher than the currently executing task.

Parameters

<i>pxTask</i>	Handle to the task for which the priority is being set. Passing a NULL handle results in the priority of the calling task being set.
<i>uxNewPriority</i>	The priority to which the task will be set.

Example usage:

```
void vAFunction( void )  
{  
    xTaskHandle xHandle;  
  
    // Create a task, storing the handle.  
    xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle );  
  
    // ...  
  
    // Use the handle to raise the priority of the created task.  
    vTaskPrioritySet( xHandle, tskIDLE_PRIORITY + 1 );  
  
    // ...  
  
    // Use a NULL handle to raise our priority to the same value.  
    vTaskPrioritySet( NULL, tskIDLE_PRIORITY + 1 );  
}
```

20.29 vTaskSuspend

task.h

```
void vTaskSuspend( xTaskHandle pxTaskToSuspend );
```

INCLUDE_vTaskSuspend must be defined as 1 for this function to be available. See the configuration section for more information.

Suspend any task. When suspended a task will never get any microcontroller processing time, no matter what its priority.

Calls to vTaskSuspend are not accumulative - i.e. calling vTaskSuspend () twice on the same task still only requires one call to vTaskResume () to ready the suspended task.

Parameters

<i>pxTaskToSuspend</i>	Handle to the task being suspended. Passing a NULL handle will cause the calling task to be suspended.
------------------------	--

Example usage:

```
void vAFunction( void )
{
    xTaskHandle xHandle;

    // Create a task, storing the handle.
    xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle );

    // ...

    // Use the handle to suspend the created task.
    vTaskSuspend( xHandle );

    // ...

    // The created task will not run during this period, unless
    // another task calls vTaskResume( xHandle ).

    //...

    // Suspend ourselves.
    vTaskSuspend( NULL );

    // We cannot get here unless another task calls vTaskResume
    // with our handle as the parameter.
}
```

20.30 vTaskResume

task.h

```
void vTaskResume( xTaskHandle pxTaskToResume );
```

INCLUDE_vTaskSuspend must be defined as 1 for this function to be available. See the configuration section for more information.

Resumes a suspended task.

A task that has been suspended by one of more calls to vTaskSuspend () will be made available for running again by a single call to vTaskResume ().

Parameters

<i>pxTaskToResume</i>	Handle to the task being readied.
-----------------------	-----------------------------------

Example usage:

```
void vAFunction( void )
{
xTaskHandle xHandle;

// Create a task, storing the handle.
xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle );

// ...

// Use the handle to suspend the created task.
vTaskSuspend( xHandle );

// ...

// The created task will not run during this period, unless
// another task calls vTaskResume( xHandle ).

//...

// Resume the suspended task ourselves.
vTaskResume( xHandle );

// The created task will once again get microcontroller processing
// time in accordance with its priority within the system.
}
```

20.31 vTaskResumeFromISR

task.h

```
void xTaskResumeFromISR( xTaskHandle pxTaskToResume );
```

INCLUDE_xTaskResumeFromISR must be defined as 1 for this function to be available. See the configuration section for more information.

An implementation of [vTaskResume\(\)](#) that can be called from within an ISR.

A task that has been suspended by one or more calls to vTaskSuspend () will be made available for running again by a single call to xTaskResumeFromISR ().

Parameters

<i>pxTaskToResume</i>	Handle to the task being readied.
-----------------------	-----------------------------------

20.32 vTaskStartScheduler

task.h

```
void vTaskStartScheduler( void );
```

Starts the real time kernel tick processing. After calling the kernel has control over which tasks are executed and when. This function does not return until an executing task calls vTaskEndScheduler () .

At least one task should be created via a call to xTaskCreate () before calling vTaskStartScheduler () . The idle task is created automatically when the first application task is created.

See the demo application file [main.c](#) for an example of creating tasks and starting the kernel.

Example usage:

```
void vAFunction( void )
{
    // Create at least one task before starting the kernel.
    xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );

    // Start the real time kernel with preemption.
    vTaskStartScheduler();

    // Will not get here unless a task calls vTaskEndScheduler ()
}
```

20.33 vTaskEndScheduler

task.h

```
void vTaskEndScheduler( void );
```

Stops the real time kernel tick. All created tasks will be automatically deleted and multitasking (either preemptive or cooperative) will stop. Execution then resumes from the point where vTaskStartScheduler () was called, as if vTaskStartScheduler () had just returned.

See the demo application file main. c in the demo/PC directory for an example that uses vTaskEndScheduler () .

vTaskEndScheduler () requires an exit function to be defined within the portable layer (see vPortEndScheduler () in port. c for the PC port). This performs hardware specific operations such as stopping the kernel tick.

vTaskEndScheduler () will cause all of the resources allocated by the kernel to be freed - but will not free resources allocated by application tasks.

Example usage:

```
void vTaskCode( void * pvParameters )
{
    for( ;; )
    {
        // Task code goes here.

        // At some point we want to end the real time kernel processing
        // so call ...
        vTaskEndScheduler ();
    }
}

void vAFunction( void )
{
    // Create at least one task before starting the kernel.
    xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );

    // Start the real time kernel with preemption.
    vTaskStartScheduler ();

    // Will only get here when the vTaskCode () task has called
    // vTaskEndScheduler (). When we get here we are back to single task
    // execution.
}
```

20.34 vTaskSuspendAll

task.h

```
void vTaskSuspendAll( void );
```

Suspends all real time kernel activity while keeping interrupts (including the kernel tick) enabled.

After calling vTaskSuspendAll () the calling task will continue to execute without risk of being swapped out until a call to xTaskResumeAll () has been made.

API functions that have the potential to cause a context switch (for example, [vTaskDelayUntil\(\)](#), [xQueueSend\(\)](#), etc.) must not be called while the scheduler is suspended.

Example usage:

```
void vTask1( void * pvParameters )
{
    for( ;; )
    {
        // Task code goes here.

        // ...

        // At some point the task wants to perform a long operation during
        // which it does not want to get swapped out. It cannot use
        // taskENTER_CRITICAL ()/taskEXIT_CRITICAL () as the length of the
        // operation may cause interrupts to be missed - including the
        // ticks.

        // Prevent the real time kernel swapping out the task.
        vTaskSuspendAll ();
}
```

```
// Perform the operation here. There is no need to use critical
// sections as we have all the microcontroller processing time.
// During this time interrupts will still operate and the kernel
// tick count will be maintained.

// ...

// The operation is complete. Restart the kernel.
xTaskResumeAll ();
}

}
```

20.35 xTaskResumeAll

task.h

```
char xTaskResumeAll( void );
```

Resumes real time kernel activity following a call to vTaskSuspendAll(). After a call to vTaskSuspendAll() the kernel will take control of which task is executing at any time.

Returns

If resuming the scheduler caused a context switch then pdTRUE is returned, otherwise pdFALSE is returned.

Example usage:

```
void vTask1( void * pvParameters )
{
    for( ;; )
    {
        // Task code goes here.

        // ...

        // At some point the task wants to perform a long operation during
        // which it does not want to get swapped out. It cannot use
        // taskENTER_CRITICAL() / taskEXIT_CRITICAL() as the length of the
        // operation may cause interrupts to be missed - including the
        // ticks.

        // Prevent the real time kernel swapping out the task.
        vTaskSuspendAll();

        // Perform the operation here. There is no need to use critical
        // sections as we have all the microcontroller processing time.
        // During this time interrupts will still operate and the real
        // time kernel tick count will be maintained.

        // ...

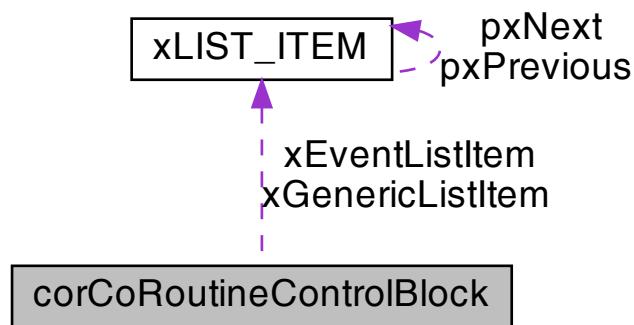
        // The operation is complete. Restart the kernel. We want to force
        // a context switch - but there is no point if resuming the scheduler
        // caused a context switch already.
        if( !xTaskResumeAll() )
        {
            taskYIELD();
        }
    }
}
```

21 Data Structure Documentation

21.1 corCoRoutineControlBlock Struct Reference

```
#include <croutine.h>
```

Collaboration diagram for corCoRoutineControlBlock:



Data Fields

- `crCOROUTINE_CODE pxCoRoutineFunction`
- `xListIItem xGenericListIItem`
- `xListIItem xEventListIItem`
- `unsigned portBASE_TYPE uxPriority`
- `unsigned portBASE_TYPE uxIndex`
- `unsigned short uxState`

21.1.1 Detailed Description

Definition at line 75 of file [croutine.h](#).

21.1.2 Field Documentation

21.1.2.1 pxCoRoutineFunction `crCOROUTINE_CODE pxCoRoutineFunction`

Definition at line 77 of file [croutine.h](#).

21.1.2.2 uxIndex `unsigned portBASE_TYPE uxIndex`

Definition at line 81 of file [croutine.h](#).

21.1.2.3 uxPriority `unsigned portBASE_TYPE uxPriority`

Definition at line 80 of file [croutine.h](#).

21.1.2.4 uxState `unsigned short uxState`

Definition at line 82 of file [croutine.h](#).

21.1.2.5 xEventListItem `xList Item xEventListItem`

Definition at line 79 of file [croutine.h](#).

21.1.2.6 xGenericListItem `xList Item xGenericListItem`

Definition at line 78 of file [croutine.h](#).

The documentation for this struct was generated from the following file:

- code/FreeRTOS/croutine.h

21.2 FLAG_t Struct Reference

This is the structure that holds the system's state flags.

```
#include <main_cfg.h>
```

Data Fields

- `u8 correctPassword`
- `u8 buzzer`
- `u8 openDoor`
- `u8 fire`

21.2.1 Detailed Description

This is the structure that holds the system's state flags.

Definition at line 26 of file [main_cfg.h](#).

21.2.2 Field Documentation

21.2.2.1 **buzzer** `u8 buzzer`

Definition at line [28](#) of file [main_cfg.h](#).

21.2.2.2 **correctPassword** `u8 correctPassword`

Definition at line [27](#) of file [main_cfg.h](#).

21.2.2.3 **fire** `u8 fire`

Definition at line [30](#) of file [main_cfg.h](#).

21.2.2.4 **openDoor** `u8 openDoor`

Definition at line [29](#) of file [main_cfg.h](#).

The documentation for this struct was generated from the following file:

- code/[main_cfg.h](#)

21.3 PASSWORD_t Struct Reference

This is the structure that holds the system's password data.

```
#include <main_cfg.h>
```

Data Fields

- `u8 value [PASSWORD_LENGTH]`
- `u8 attempts`
- const `u8 maxAttempts`

21.3.1 Detailed Description

This is the structure that holds the system's password data.

Definition at line [36](#) of file [main_cfg.h](#).

21.3.2 Field Documentation

21.3.2.1 attempts `u8 attempts`

Definition at line 38 of file [main_cfg.h](#).

21.3.2.2 maxAttempts `const u8 maxAttempts`

Definition at line 39 of file [main_cfg.h](#).

21.3.2.3 value `u8 value[PASSWORD_LENGTH]`

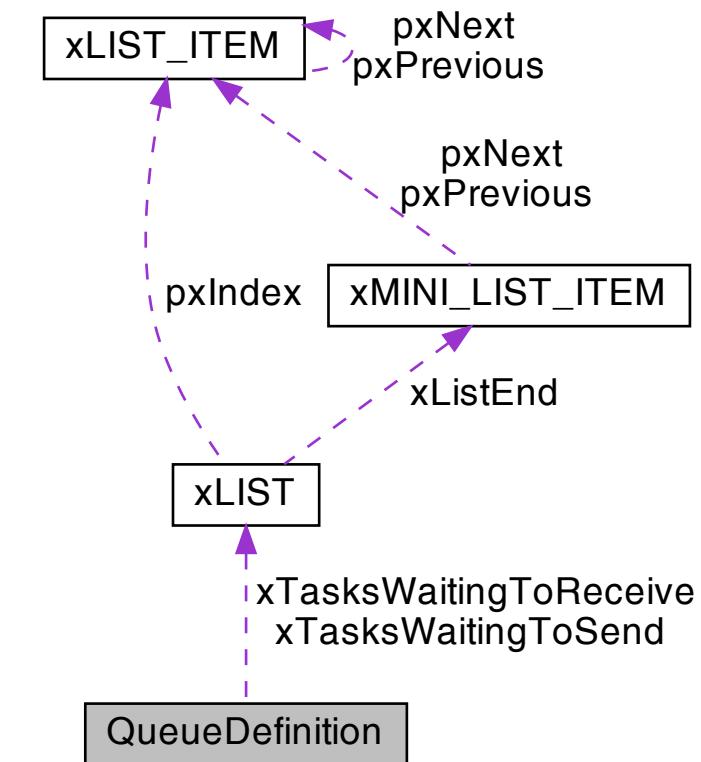
Definition at line 37 of file [main_cfg.h](#).

The documentation for this struct was generated from the following file:

- code/[main_cfg.h](#)

21.4 QueueDefinition Struct Reference

Collaboration diagram for QueueDefinition:



Data Fields

- signed char * **pcHead**
- signed char * **pcTail**
- signed char * **pcWriteTo**
- signed char * **pcReadFrom**
- **xList xTasksWaitingToSend**
- **xList xTasksWaitingToReceive**
- volatile unsigned **portBASE_TYPE uxMessagesWaiting**
- unsigned **portBASE_TYPE uxLength**
- unsigned **portBASE_TYPE uxItemSize**
- signed **portBASE_TYPE xRxLock**
- signed **portBASE_TYPE xTxLock**

21.4.1 Detailed Description

Definition at line 101 of file [queue.c](#).

21.4.2 Field Documentation

21.4.2.1 **pcHead** signed char* pcHead

Definition at line 103 of file [queue.c](#).

21.4.2.2 **pcReadFrom** signed char* pcReadFrom

Definition at line 107 of file [queue.c](#).

21.4.2.3 **pcTail** signed char* pcTail

Definition at line 104 of file [queue.c](#).

21.4.2.4 **pcWriteTo** signed char* pcWriteTo

Definition at line 106 of file [queue.c](#).

21.4.2.5 uxItemSize `unsigned portBASE_TYPE uxItemSize`

Definition at line 114 of file [queue.c](#).

21.4.2.6 uxLength `unsigned portBASE_TYPE uxLength`

Definition at line 113 of file [queue.c](#).

21.4.2.7 uxMessagesWaiting `volatile unsigned portBASE_TYPE uxMessagesWaiting`

Definition at line 112 of file [queue.c](#).

21.4.2.8 xRxLock `signed portBASE_TYPE xRxLock`

Definition at line 116 of file [queue.c](#).

21.4.2.9 xTasksWaitingToReceive `xList xTasksWaitingToReceive`

Definition at line 110 of file [queue.c](#).

21.4.2.10 xTasksWaitingToSend `xList xTasksWaitingToSend`

Definition at line 109 of file [queue.c](#).

21.4.2.11 xTxLock `signed portBASE_TYPE xTxLock`

Definition at line 117 of file [queue.c](#).

The documentation for this struct was generated from the following file:

- [code/FreeRTOS/queue.c](#)

21.5 SENSOR_t Struct Reference

This is the structure that holds the system's sensors data.

```
#include <main_cfg.h>
```

Data Fields

- `u16 u16_CurrentValueInBinary`
- `u8 u8_CriticalValue`
- `u8 u8_MaxValue`
- `u8 u8_CurrentValue`
- `u8 u8_AdcChannel`

21.5.1 Detailed Description

This is the structure that holds the system's sensors data.

Definition at line [45](#) of file [main_cfg.h](#).

21.5.2 Field Documentation

21.5.2.1 `u16_CurrentValueInBinary` `u16 u16_CurrentValueInBinary`

Definition at line [46](#) of file [main_cfg.h](#).

21.5.2.2 `u8_AdcChannel` `u8 u8_AdcChannel`

Definition at line [50](#) of file [main_cfg.h](#).

21.5.2.3 `u8_CriticalValue` `u8 u8_CriticalValue`

Definition at line [47](#) of file [main_cfg.h](#).

21.5.2.4 `u8_CurrentValue` `u8 u8_CurrentValue`

Definition at line [49](#) of file [main_cfg.h](#).

21.5.2.5 `u8_MaxValue` `u8 u8_MaxValue`

Definition at line [48](#) of file [main_cfg.h](#).

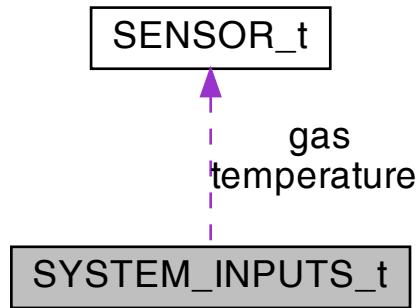
The documentation for this struct was generated from the following file:

- code/[main_cfg.h](#)

21.6 SYSTEM_INPUTS_t Struct Reference

```
#include <main_cfg.h>
```

Collaboration diagram for SYSTEM_INPUTS_t:



Data Fields

- `SENSOR_t temperature`
- `SENSOR_t gas`

21.6.1 Detailed Description

Definition at line 53 of file [main_cfg.h](#).

21.6.2 Field Documentation

21.6.2.1 `gas` `SENSOR_t` `gas`

Definition at line 55 of file [main_cfg.h](#).

21.6.2.2 `temperature` `SENSOR_t` `temperature`

Definition at line 54 of file [main_cfg.h](#).

The documentation for this struct was generated from the following file:

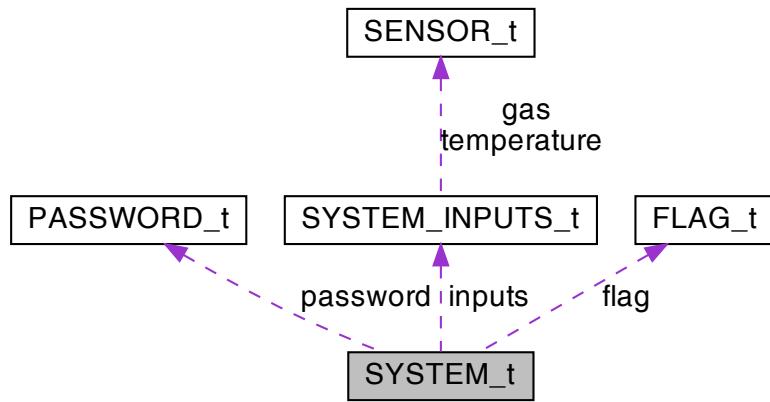
- [code/main_cfg.h](#)

21.7 SYSTEM_t Struct Reference

This is the structure that holds the system's data that is used to control the system and display it on the LCD screen. The system's data is:

```
#include <main_cfg.h>
```

Collaboration diagram for SYSTEM_t:



Data Fields

- SYSTEM_INPUTS_t inputs
- PASSWORD_t password
- FLAG_t flag

21.7.1 Detailed Description

This is the structure that holds the system's data that is used to control the system and display it on the LCD screen. The system's data is:

- The system's state flags:
 - correctPassword: The system's password is correct
 - buzzer: The system's buzzer is enabled
 - openDoor: The system's door should be opened
- The system's password data:
 - The system's password value
 - The system's password attempts counter
- The system's sensors data:
 - The system's temperature sensor data:
 - * The system's current temperature value in Celsius

- * The system's current temperature value in binary
- * The system's maximum temperature value in Celsius
- The system's gas sensor data
 - * The system's current gas level value in binary
 - * The system's maximum gas level in percentage
 - * The system's current gas level in percentage

Definition at line 80 of file [main_cfg.h](#).

21.7.2 Field Documentation

21.7.2.1 flag [FLAG_t](#) flag

Definition at line 83 of file [main_cfg.h](#).

21.7.2.2 inputs [SYSTEM_INPUTS_t](#) inputs

Definition at line 81 of file [main_cfg.h](#).

21.7.2.3 password [PASSWORD_t](#) password

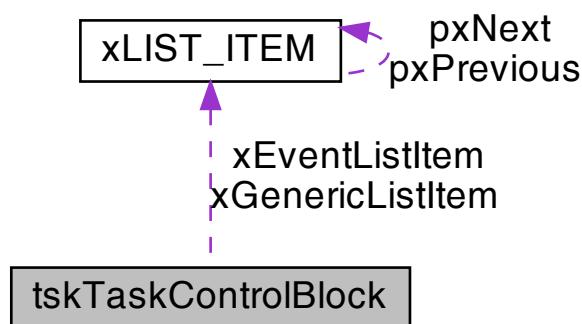
Definition at line 82 of file [main_cfg.h](#).

The documentation for this struct was generated from the following file:

- code/[main_cfg.h](#)

21.8 tskTaskControlBlock Struct Reference

Collaboration diagram for tskTaskControlBlock:



Data Fields

- volatile `portSTACK_TYPE` * `pxTopOfStack`
- `xListItem` `xGenericListItem`
- `xListItem` `xEventListItem`
- unsigned `portBASE_TYPE` `uxPriority`
- `portSTACK_TYPE` * `pxStack`
- signed char `pcTaskName` [`configMAX_TASK_NAME_LEN`]

21.8.1 Detailed Description

Definition at line 80 of file [tasks.c](#).

21.8.2 Field Documentation

21.8.2.1 `pcTaskName` signed char `pcTaskName`[`configMAX_TASK_NAME_LEN`]

Definition at line 92 of file [tasks.c](#).

21.8.2.2 `pxStack` `portSTACK_TYPE*` `pxStack`

Definition at line 91 of file [tasks.c](#).

21.8.2.3 `pxTopOfStack` volatile `portSTACK_TYPE*` `pxTopOfStack`

Definition at line 82 of file [tasks.c](#).

21.8.2.4 `uxPriority` unsigned `portBASE_TYPE` `uxPriority`

Definition at line 90 of file [tasks.c](#).

21.8.2.5 `xEventListItem` `xListItem` `xEventListItem`

Definition at line 89 of file [tasks.c](#).

21.8.2.6 xGenericListItem `xListIem` `xGenericListItem`

Definition at line 88 of file [tasks.c](#).

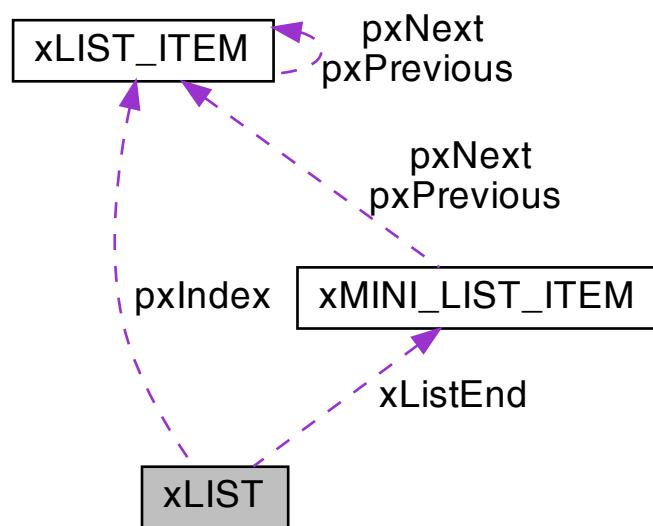
The documentation for this struct was generated from the following file:

- [code/FreeRTOS/tasks.c](#)

21.9 xLIST Struct Reference

```
#include <list.h>
```

Collaboration diagram for xLIST:



Data Fields

- volatile unsigned `portBASE_TYPE uxNumberOfItems`
- volatile `xListIem * pxIndex`
- volatile `xMiniListIem xListEnd`

21.9.1 Detailed Description

Definition at line 113 of file [list.h](#).

21.9.2 Field Documentation

21.9.2.1 pxIndex volatile `xListItem*` pxIndex

Definition at line 116 of file [list.h](#).

21.9.2.2 uxNumberOfItems volatile unsigned `portBASE_TYPE` uxNumberOfItems

Definition at line 115 of file [list.h](#).

21.9.2.3 xListEnd volatile `xMiniListItem` xListEnd

Definition at line 117 of file [list.h](#).

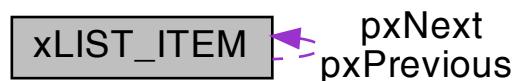
The documentation for this struct was generated from the following file:

- code/FreeRTOS/list.h

21.10 xLIST_ITEM Struct Reference

```
#include <list.h>
```

Collaboration diagram for xLIST_ITEM:



Data Fields

- `portTickType xItemValue`
- volatile struct `xLIST_ITEM` * `pxNext`
- volatile struct `xLIST_ITEM` * `pxPrevious`
- `void * pvOwner`
- `void * pvContainer`

21.10.1 Detailed Description

Definition at line 92 of file [list.h](#).

21.10.2 Field Documentation

21.10.2.1 **pvContainer** void* pvContainer

Definition at line 98 of file [list.h](#).

21.10.2.2 **pvOwner** void* pvOwner

Definition at line 97 of file [list.h](#).

21.10.2.3 **pxNext** volatile struct [xLIST_ITEM](#)* pxNext

Definition at line 95 of file [list.h](#).

21.10.2.4 **pxPrevious** volatile struct [xLIST_ITEM](#)* pxPrevious

Definition at line 96 of file [list.h](#).

21.10.2.5 **xItemValue** [portTickType](#) xItemValue

Definition at line 94 of file [list.h](#).

The documentation for this struct was generated from the following file:

- code/FreeRTOS/list.h

21.11 xMEMORY_REGION Struct Reference

```
#include <task.h>
```

Data Fields

- void * [pvBaseAddress](#)
- unsigned long [ulLengthInBytes](#)
- unsigned long [ulParameters](#)

21.11.1 Detailed Description

Definition at line 99 of file [task.h](#).

21.11.2 Field Documentation

21.11.2.1 **pvBaseAddress** void* pvBaseAddress

Definition at line 101 of file [task.h](#).

21.11.2.2 **ulLengthInBytes** unsigned long ulLengthInBytes

Definition at line 102 of file [task.h](#).

21.11.2.3 **ulParameters** unsigned long ulParameters

Definition at line 103 of file [task.h](#).

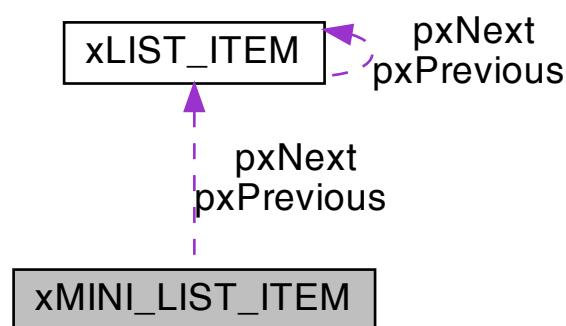
The documentation for this struct was generated from the following file:

- [code/FreeRTOS/task.h](#)

21.12 xMINI_LIST_ITEM Struct Reference

```
#include <list.h>
```

Collaboration diagram for xMINI_LIST_ITEM:



Data Fields

- `portTickType xItemValue`
- volatile struct `xLIST_ITEM` * `pxNext`
- volatile struct `xLIST_ITEM` * `pxPrevious`

21.12.1 Detailed Description

Definition at line 102 of file [list.h](#).

21.12.2 Field Documentation

21.12.2.1 pxNext volatile struct `xLIST_ITEM`* pxNext

Definition at line 105 of file [list.h](#).

21.12.2.2 pxPrevious volatile struct `xLIST_ITEM`* pxPrevious

Definition at line 106 of file [list.h](#).

21.12.2.3 xItemValue portTickType xItemValue

Definition at line 104 of file [list.h](#).

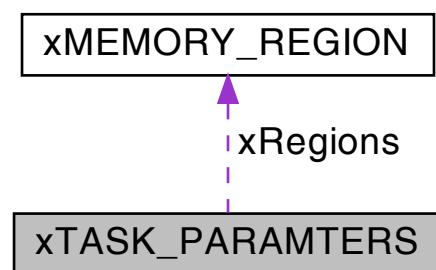
The documentation for this struct was generated from the following file:

- [code/FreeRTOS/list.h](#)

21.13 xTASK_PARAMTERS Struct Reference

```
#include <task.h>
```

Collaboration diagram for xTASK_PARAMTERS:



Data Fields

- `pdTASK_CODE pvTaskCode`
- `const signed char *const pcName`
- `unsigned short usStackDepth`
- `void * pvParameters`
- `unsigned portBASE_TYPE uxPriority`
- `portSTACK_TYPE * puxStackBuffer`
- `xMemoryRegion xRegions [portNUM_CONFIGURABLE_REGIONS]`

21.13.1 Detailed Description

Definition at line 109 of file [task.h](#).

21.13.2 Field Documentation

21.13.2.1 `pcName const signed char* const pcName`

Definition at line 112 of file [task.h](#).

21.13.2.2 `pxxStackBuffer portSTACK_TYPE* puxStackBuffer`

Definition at line 116 of file [task.h](#).

21.13.2.3 `pvParameters void* pvParameters`

Definition at line 114 of file [task.h](#).

21.13.2.4 `pvTaskCode pdTASK_CODE pvTaskCode`

Definition at line 111 of file [task.h](#).

21.13.2.5 `usStackDepth unsigned short usStackDepth`

Definition at line 113 of file [task.h](#).

21.13.2.6 uxPriority `unsigned portBASE_TYPE uxPriority`

Definition at line 115 of file [task.h](#).

21.13.2.7 xRegions `xMemoryRegion xRegions[portNUM_CONFIGURABLE_REGIONS]`

Definition at line 117 of file [task.h](#).

The documentation for this struct was generated from the following file:

- [code/FreeRTOS/task.h](#)

21.14 xTIME_OUT Struct Reference

```
#include <task.h>
```

Data Fields

- `portBASE_TYPE xOverflowCount`
- `portTickType xTimeOnEntering`

21.14.1 Detailed Description

Definition at line 90 of file [task.h](#).

21.14.2 Field Documentation

21.14.2.1 xOverflowCount `portBASE_TYPE xOverflowCount`

Definition at line 92 of file [task.h](#).

21.14.2.2 xTimeOnEntering `portTickType xTimeOnEntering`

Definition at line 93 of file [task.h](#).

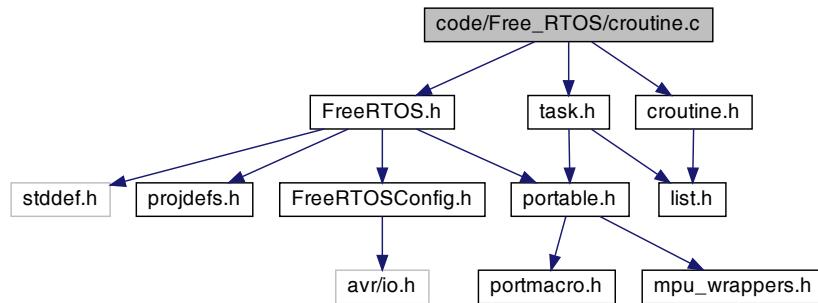
The documentation for this struct was generated from the following file:

- [code/FreeRTOS/task.h](#)

22 File Documentation

22.1 code/FreeRTOS/croutine.c File Reference

```
#include "FreeRTOS.h"
#include "task.h"
#include "croutine.h"
Include dependency graph for croutine.c:
```



Macros

- #define corINITIAL_STATE (0)
- #define prvAddCoRoutineToReadyQueue(pxCRCB)

Functions

- signed portBASE_TYPE xCoRoutineCreate (crCOROUTINE_CODE pxCoRoutineCode, unsigned portBASE_TYPE uxPriority, unsigned portBASE_TYPE uxIndex)
- void vCoRoutineAddToDelayedList (portTickType xTicksToDelay, xList *pxEventList)
- void vCoRoutineSchedule (void)
- signed portBASE_TYPE xCoRoutineRemoveFromEventList (const xList *pxEventList)

Variables

- corCRCB * pxCurrentCoRoutine = NULL

22.1.1 Macro Definition Documentation

22.1.1.1 corINITIAL_STATE #define corINITIAL_STATE (0)

Definition at line 81 of file [croutine.c](#).

22.1.1.2 prvAddCoRoutineToReadyQueue #define prvAddCoRoutineToReadyQueue(pxCRCB)

Value:

```
{
    \
    if( pxCRCB->uxPriority > uxTopCoRoutineReadyPriority )
        \
    {
        \
        uxTopCoRoutineReadyPriority = pxCRCB->uxPriority;
        \
    }
    \
    vListInsertEnd( ( xList * ) &( pxReadyCoRoutineLists[ pxCRCB->uxPriority ] ), &
    pxCRCB->xGenericListItem );
}
```

Definition at line 90 of file [croutine.c](#).

22.1.2 Function Documentation

22.1.2.1 vCoRoutineAddToDelayedList() void vCoRoutineAddToDelayedList (
 portTickType xTicksToDelay,
 xList * pxEventList)

Definition at line 182 of file [croutine.c](#).

22.1.2.2 vCoRoutineSchedule() void vCoRoutineSchedule (
 void)

Definition at line 301 of file [croutine.c](#).

22.1.2.3 xCoRoutineCreate() signed portBASE_TYPE xCoRoutineCreate (
 crCOROUTINE_CODE pxCoRoutineCode,
 unsigned portBASE_TYPE uxPriority,
 unsigned portBASE_TYPE uxIndex)

Definition at line 125 of file [croutine.c](#).

Here is the call graph for this function:



22.1.2.4 xCoRoutineRemoveFromEventList() signed `portBASE_TYPE xCoRoutineRemoveFromEventList (const xList * pxEventList)`

Definition at line 351 of file [croutine.c](#).

Here is the call graph for this function:



22.1.3 Variable Documentation

22.1.3.1 pxCurrentCoRoutine `corCRCB* pxCurrentCoRoutine = NULL`

Definition at line 76 of file [croutine.c](#).

22.2 croutine.c

```

00001 /*
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005 ****
00006 *
00007 *   FreeRTOS tutorial books are available in pdf and paperback.
00008 *   Complete, revised, and edited pdf reference manuals are also
00009 *   available.
00010 *
00011 *   Purchasing FreeRTOS documentation will not only help you, by
00012 *   ensuring you get running as quickly as possible and with an
00013 *   in-depth knowledge of how to use FreeRTOS, it will also help
00014 *   the FreeRTOS project to continue with its mission of providing
00015 *   professional grade, cross platform, de facto standard solutions
00016 *   for microcontrollers - completely free of charge!
00017 *
00018 *   >> See http://www.FreeRTOS.org/Documentation for details. <<
00019 *
00020 *   Thank you for using FreeRTOS, and thank you for your support!
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 >>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039 by writing to Richard Barry, contact details for whom are available on the
00040 FreeRTOS WEB site.
  
```

```

00041     1 tab == 4 spaces!
00043
00044     http://www.FreeRTOS.org - Documentation, latest information, license and
00045     contact details.
00046
00047     http://www.SafeRTOS.com - A version that is certified for use in safety
00048     critical systems.
00049
00050     http://www.OpenRTOS.com - Commercial support, development, porting,
00051     licensing and training services.
00052 */
00053
00054 #include "FreeRTOS.h"
00055 #include "task.h"
00056 #include "croutine.h"
00057
00058 /*
00059  * Some kernel aware debuggers require data to be viewed to be global, rather
00060  * than file scope.
00061 */
00062 #ifdef portREMOVE_STATIC_QUALIFIER
00063     #define static
00064 #endif
00065
00066
00067 /* Lists for ready and blocked co-routines. -----*/
00068 static xList pxReadyCoRoutineLists[ configMAX_CO_ROUTINE_PRIORITIES ];  /*< Prioritised ready
    co-routines. */
00069 static xList xDelayedCoRoutineList1;                                     /*< Delayed co-routines. */
00070 static xList xDelayedCoRoutineList2;                                     /*< Delayed co-routines (two
    lists are used - one for delays that have overflowed the current tick count. */
00071 static xList * pxDelayedCoRoutineList;                                    /*< Points to the delayed
    co-routine list currently being used. */
00072 static xList * pxOverflowDelayedCoRoutineList;                          /*< Points to the delayed
    co-routine list currently being used to hold co-routines that have overflowed the current tick count.
    */
00073 static xList xPendingReadyCoRoutineList;                                /*< Holds
    co-routines that have been readied by an external event. They cannot be added directly to the ready
    lists as the ready lists cannot be accessed by interrupts. */
00074
00075 /* Other file private variables. -----*/
00076 corCRCB * pxCurrentCoRoutine = NULL;
00077 static unsigned portBASE_TYPE uxTopCoRoutineReadyPriority = 0;
00078 static portTickType xCoRoutineTickCount = 0, xLastTickCount = 0, xPassedTicks = 0;
00079
00080 /* The initial state of the co-routine when it is created. */
00081 #define corINITIAL_STATE      ( 0 )
00082
00083 /*
00084  * Place the co-routine represented by pxCRCB into the appropriate ready queue
00085  * for the priority. It is inserted at the end of the list.
00086 *
00087  * This macro accesses the co-routine ready lists and therefore must not be
00088  * used from within an ISR.
00089 */
00090 #define prvAddCoRoutineToReadyQueue( pxCRCB )
    \
00091 {
    \
00092     if( pxCRCB->uxPriority > uxTopCoRoutineReadyPriority )
    \
00093     {
    \
00094         uxTopCoRoutineReadyPriority = pxCRCB->uxPriority;
    \
00095     }
    \
00096     vListInsertEnd( ( xList * ) &( pxReadyCoRoutineLists[ pxCRCB->uxPriority ] ), &( \
00097         pxCRCB->xGenericListItem ) );
00098
00099 /*
00100  * Utility to ready all the lists used by the scheduler. This is called
00101  * automatically upon the creation of the first co-routine.
00102 */
00103 static void prvInitialiseCoRoutineLists( void );
00104
00105 /*
00106  * Co-routines that are readied by an interrupt cannot be placed directly into
00107  * the ready lists (there is no mutual exclusion). Instead they are placed in
00108  * in the pending ready list in order that they can later be moved to the ready
00109  * list by the co-routine scheduler.
00110 */
00111 static void prvCheckPendingReadyList( void );
00112
00113 */

```

```

00114 * Macro that looks at the list of co-routines that are currently delayed to
00115 * see if any require waking.
00116 *
00117 * Co-routines are stored in the queue in the order of their wake time -
00118 * meaning once one co-routine has been found whose timer has not expired
00119 * we need not look any further down the list.
00120 */
00121 static void prvCheckDelayedList( void );
00122
00123 /*****+
00124
00125 signed portBASE_TYPE xCoRoutineCreate( crCOROUTINE_CODE pxCoRoutineCode, unsigned portBASE_TYPE
00126     uxPriority, unsigned portBASE_TYPE uxIndex )
00127 {
00128     signed portBASE_TYPE xReturn;
00129     corCRCB *pxCoRoutine;
00130
00131     /* Allocate the memory that will store the co-routine control block. */
00132     pxCoRoutine = ( corCRCB * ) pvPortMalloc( sizeof( corCRCB ) );
00133     if( pxCoRoutine )
00134     {
00135         /* If pxCurrentCoRoutine is NULL then this is the first co-routine to
00136         be created and the co-routine data structures need initialising. */
00137         if( pxCurrentCoRoutine == NULL )
00138         {
00139             pxCurrentCoRoutine = pxCoRoutine;
00140             prvInitialiseCoRoutineLists();
00141         }
00142
00143         /* Check the priority is within limits. */
00144         if( uxPriority >= configMAX_CO_ROUTINE_PRIORITIES )
00145         {
00146             uxPriority = configMAX_CO_ROUTINE_PRIORITIES - 1;
00147         }
00148
00149         /* Fill out the co-routine control block from the function parameters. */
00150         pxCoRoutine->uxState = corINITIAL_STATE;
00151         pxCoRoutine->uxPriority = uxPriority;
00152         pxCoRoutine->uxIndex = uxIndex;
00153         pxCoRoutine->pxCoRoutineFunction = pxCoRoutineCode;
00154
00155         /* Initialise all the other co-routine control block parameters. */
00156         vListInitialiseItem( &( pxCoRoutine->xGenericListItem ) );
00157         vListInitialiseItem( &( pxCoRoutine->xEventListItem ) );
00158
00159         /* Set the co-routine control block as a link back from the xListItemIcon.
00160         This is so we can get back to the containing CRCB from a generic item
00161         in a list. */
00162         listSET_LIST_ITEM_OWNER( &( pxCoRoutine->xGenericListItem ), pxCoRoutine );
00163         listSET_LIST_ITEM_OWNER( &( pxCoRoutine->xEventListItem ), pxCoRoutine );
00164
00165         /* Event lists are always in priority order. */
00166         listSET_LIST_ITEM_VALUE( &( pxCoRoutine->xEventListItem ), configMAX_PRIORITIES -
00167             ( portTickType ) uxPriority );
00168
00169         /* Now the co-routine has been initialised it can be added to the ready
00170         list at the correct priority. */
00171         prvAddCoRoutineToReadyQueue( pxCoRoutine );
00172
00173         xReturn = pdPASS;
00174     }
00175     else
00176     {
00177         xReturn = errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY;
00178     }
00179 }
00180 /*****+
00181
00182 void vCoRoutineAddToDelayedList( portTickType xTicksToDelay, xList *pxEventList )
00183 {
00184     portTickType xTimeToWake;
00185
00186     /* Calculate the time to wake - this may overflow but this is
00187     not a problem. */
00188     xTimeToWake = xCoRoutineTickCount + xTicksToDelay;
00189
00190     /* We must remove ourselves from the ready list before adding
00191     ourselves to the blocked list as the same list item is used for
00192     both lists. */
00193     vListRemove( ( xListItemIcon * ) &( pxCurrentCoRoutine->xGenericListItem ) );
00194
00195     /* The list item will be inserted in wake time order. */
00196     listSET_LIST_ITEM_VALUE( &( pxCurrentCoRoutine->xGenericListItem ), xTimeToWake );
00197
00198     if( xTimeToWake < xCoRoutineTickCount )

```

```

00199     {
00200         /* Wake time has overflowed. Place this item in the
00201             overflow list. */
00202         vListInsert( ( xList * ) pxOverflowDelayedCoRoutineList, ( xListItem * ) &(
00203             pxCurrentCoRoutine->xGenericListItem ) );
00204     }
00205     {
00206         /* The wake time has not overflowed, so we can use the
00207             current block list. */
00208         vListInsert( ( xList * ) pxDelayedCoRoutineList, ( xListItem * ) &(
00209             pxCurrentCoRoutine->xGenericListItem ) );
00210     }
00211     if( pxEventList )
00212     {
00213         /* Also add the co-routine to an event list. If this is done then the
00214             function must be called with interrupts disabled. */
00215         vListInsert( pxEventList, &( pxCurrentCoRoutine->xEventListItem ) );
00216     }
00217 }
00218 /*-----*/
00219
00220 static void prvCheckPendingReadyList( void )
00221 {
00222     /* Are there any co-routines waiting to get moved to the ready list? These
00223         are co-routines that have been readied by an ISR. The ISR cannot access
00224         the ready lists itself. */
00225     while( listLIST_IS_EMPTY( &xPendingReadyCoRoutineList ) == pdFALSE )
00226     {
00227         corCRCB *pxUnblockedCRCB;
00228
00229         /* The pending ready list can be accessed by an ISR. */
00230         portDISABLE_INTERRUPTS();
00231         {
00232             pxUnblockedCRCB = ( corCRCB * ) listGET_OWNER_OF_HEAD_ENTRY( &xPendingReadyCoRoutineList )
00233         };
00234         vListRemove( &( pxUnblockedCRCB->xEventListItem ) );
00235         portENABLE_INTERRUPTS();
00236
00237         vListRemove( &( pxUnblockedCRCB->xGenericListItem ) );
00238         prvAddCoRoutineToReadyQueue( pxUnblockedCRCB );
00239     }
00240 }
00241 /*-----*/
00242
00243 static void prvCheckDelayedList( void )
00244 {
00245     corCRCB *pxCRCB;
00246
00247     xPassedTicks = xTaskGetTickCount() - xLastTickCount;
00248     while( xPassedTicks )
00249     {
00250         xCoRoutineTickCount++;
00251         xPassedTicks--;
00252
00253         /* If the tick count has overflowed we need to swap the ready lists. */
00254         if( xCoRoutineTickCount == 0 )
00255         {
00256             xList * pxTemp;
00257
00258             /* Tick count has overflowed so we need to swap the delay lists. If there are
00259                 any items in pxDelayedCoRoutineList here then there is an error! */
00260             pxTemp = pxDelayedCoRoutineList;
00261             pxDelayedCoRoutineList = pxOverflowDelayedCoRoutineList;
00262             pxOverflowDelayedCoRoutineList = pxTemp;
00263         }
00264
00265         /* See if this tick has made a timeout expire. */
00266         while( listLIST_IS_EMPTY( pxDelayedCoRoutineList ) == pdFALSE )
00267         {
00268             pxCRCB = ( corCRCB * ) listGET_OWNER_OF_HEAD_ENTRY( pxDelayedCoRoutineList );
00269
00270             if( xCoRoutineTickCount < listGET_LIST_ITEM_VALUE( &( pxCRCB->xGenericListItem ) ) )
00271             {
00272                 /* Timeout not yet expired. */
00273
00274                 break;
00275             }
00276             portDISABLE_INTERRUPTS();
00277             {
00278                 /* The event could have occurred just before this critical
00279                     section. If this is the case then the generic list item will
00280                     have been moved to the pending ready list and the following

```

```

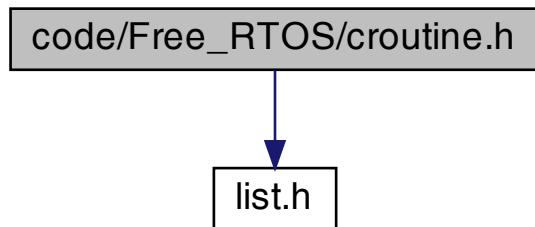
00281             line is still valid. Also the pxContainer parameter will have
00282             been set to NULL so the following lines are also valid. */
00283             vListRemove( &( pxCRCB->xGenericListItem ) );
00284
00285             /* Is the co-routine waiting on an event also? */
00286
00287             if( pxCRCB->xEventListItem.pxContainer )
00288             {
00289                 vListRemove( &( pxCRCB->xEventListItem ) );
00290             }
00291             portENABLE_INTERRUPTS();
00292
00293             prvAddCoRoutineToReadyQueue( pxCRCB );
00294         }
00295     }
00296
00297     xLastTickCount = xCoRoutineTickCount;
00298 }
00299 /*-----*/
00300
00301 void vCoRoutineSchedule( void )
00302 {
00303     /* See if any co-routines readied by events need moving to the ready lists. */
00304     prvCheckPendingReadyList();
00305
00306     /* See if any delayed co-routines have timed out. */
00307     prvCheckDelayedList();
00308
00309     /* Find the highest priority queue that contains ready co-routines. */
00310     while( listLIST_IS_EMPTY( &( pxReadyCoRoutineLists[ uxTopCoRoutineReadyPriority ] ) ) )
00311     {
00312         if( uxTopCoRoutineReadyPriority == 0 )
00313         {
00314             /* No more co-routines to check. */
00315             return;
00316         }
00317         --uxTopCoRoutineReadyPriority;
00318     }
00319
00320     /* listGET_OWNER_OF_NEXT_ENTRY walks through the list, so the co-routines
00321      of the same priority get an equal share of the processor time. */
00322     listGET_OWNER_OF_NEXT_ENTRY( pxCurrentCoRoutine, &( pxReadyCoRoutineLists[
00323         uxTopCoRoutineReadyPriority ] ) );
00324
00325     /* Call the co-routine. */
00326     ( pxCurrentCoRoutine->pxCoRoutineFunction )( pxCurrentCoRoutine, pxCurrentCoRoutine->uxIndex );
00327
00328 }
00329 /*-----*/
00330
00331 static void prvInitialiseCoRoutineLists( void )
00332 {
00333     unsigned portBASE_TYPE uxPriority;
00334
00335     for( uxPriority = 0; uxPriority < configMAX_CO_ROUTINE_PRIORITIES; uxPriority++ )
00336     {
00337         vListInitialise( ( xList * ) &( pxReadyCoRoutineLists[ uxPriority ] ) );
00338     }
00339
00340     vListInitialise( ( xList * ) &xDelayedCoRoutineList1 );
00341     vListInitialise( ( xList * ) &xDelayedCoRoutineList2 );
00342     vListInitialise( ( xList * ) &xPendingReadyCoRoutineList );
00343
00344     /* Start with pxDelayedCoRoutineList using list1 and the
00345      pxOverflowDelayedCoRoutineList using list2. */
00346     pxDelayedCoRoutineList = &xDelayedCoRoutineList1;
00347     pxOverflowDelayedCoRoutineList = &xDelayedCoRoutineList2;
00348 }
00349 /*-----*/
00350
00351 signed portBASE_TYPE xCoRoutineRemoveFromEventList( const xList *pxEventList )
00352 {
00353     corCRCB *pxUnblockedCRCB;
00354     signed portBASE_TYPE xReturn;
00355
00356     /* This function is called from within an interrupt. It can only access
00357      event lists and the pending ready list. This function assumes that a
00358      check has already been made to ensure pxEventList is not empty. */
00359     pxUnblockedCRCB = ( corCRCB * ) listGET_OWNER_OF_HEAD_ENTRY( pxEventList );
00360     vListRemove( &( pxUnblockedCRCB->xEventListItem ) );
00361     vListInsertEnd( ( xList * ) &xPendingReadyCoRoutineList ), &( pxUnblockedCRCB->xEventListItem )

```

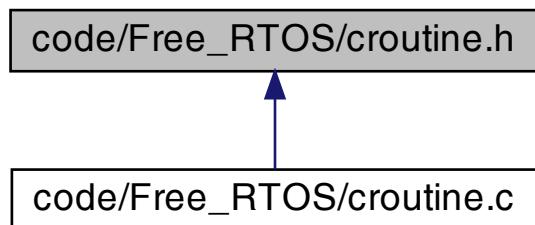
```
00362     );
00363     if( pxUnblockedCRCB->uxPriority >= pxCurrentCoRoutine->uxPriority )
00364     {
00365         xReturn = pdTRUE;
00366     }
00367     else
00368     {
00369         xReturn = pdFALSE;
00370     }
00371     return xReturn;
00372 }
00373 }
```

22.3 code/FreeRTOS/croutine.h File Reference

```
#include "list.h"
Include dependency graph for croutine.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [corCoRoutineControlBlock](#)

Macros

- #define crSTART(pxCRCB) switch(((corCRCB *)(pxCRCB))->uxState) { case 0:
- #define crEND() }
- #define crSET_STATE0(xHandle) ((corCRCB *)(xHandle))->uxState = (__LINE__ * 2); return; case (__LINE__ * 2):
- #define crSET_STATE1(xHandle) ((corCRCB *)(xHandle))->uxState = ((__LINE__ * 2)+1); return; case ((__LINE__ * 2)+1):
- #define crDELAY(xHandle, xTicksToDelay)
- #define crQUEUE_SEND(xHandle, pxQueue, pvlItemToQueue, xTicksToWait, pxResult)
- #define crQUEUE_RECEIVE(xHandle, pxQueue, pvBuffer, xTicksToWait, pxResult)
- #define crQUEUE_SEND_FROM_ISR(pxQueue, pvlItemToQueue, xCoRoutinePreviouslyWoken) xQueueCRSendFromISR((pxQueue), (pvlItemToQueue), (xCoRoutinePreviouslyWoken))
- #define crQUEUE_RECEIVE_FROM_ISR(pxQueue, pvBuffer, pxCoRoutineWoken) xQueueCRReceiveFromISR((pxQueue), (pvBuffer), (pxCoRoutineWoken))

Typedefs

- typedef void * xCoRoutineHandle
- typedef void(* crCOROUTINE_CODE)(xCoRoutineHandle, unsigned portBASE_TYPE)
- typedef struct corCoRoutineControlBlock corCRCB

Functions

- signed portBASE_TYPE xCoRoutineCreate (crCOROUTINE_CODE pxCoRoutineCode, unsigned portBASE_TYPE uxPriority, unsigned portBASE_TYPE uxIndex)
- void vCoRoutineSchedule (void)
- void vCoRoutineAddToDelayedList (portTickType xTicksToDelay, xList *pxEventList)
- signed portBASE_TYPE xCoRoutineRemoveFromEventList (const xList *pxEventList)

22.3.1 Macro Definition Documentation

22.3.1.1 crDELAY #define crDELAY(
 xHandle,
 xTicksToDelay)

Value:

```
if( ( xTicksToDelay ) > 0 )
{
    vCoRoutineAddToDelayedList( ( xTicksToDelay ), NULL );
}
crSET_STATE0( ( xHandle ) );
```

Definition at line 316 of file [croutine.h](#).

22.3.1.2 crEND #define crEND() }

Definition at line 261 of file [croutine.h](#).

```
22.3.1.3 crQUEUE_RECEIVE #define crQUEUE_RECEIVE(
    xHandle,
    pxQueue,
    pvBuffer,
    xTicksToWait,
    pxResult )
```

Value:

```
{
    *( pxResult ) = xQueueCRReceive( ( pxQueue ) , ( pvBuffer ) , ( xTicksToWait ) );
    if( *( pxResult ) == errQUEUE_BLOCKED )
    {
        crSET_STATE0( ( xHandle ) );
        *( pxResult ) = xQueueCRReceive( ( pxQueue ) , ( pvBuffer ) , 0 );
    }
    if( *( pxResult ) == errQUEUE_YIELD )
    {
        crSET_STATE1( ( xHandle ) );
        *( pxResult ) = pdPASS;
    }
}
```



Definition at line 498 of file [croutine.h](#).

```
22.3.1.4 crQUEUE_RECEIVE_FROM_ISR #define crQUEUE_RECEIVE_FROM_ISR(
```

```
    pxQueue,
    pvBuffer,
    pxCoRoutineWoken ) xQueueCRReceiveFromISR( ( pxQueue ) , ( pvBuffer ) , ( pxCo←
    RoutineWoken ) )
```

Definition at line 720 of file [croutine.h](#).

```
22.3.1.5 crQUEUE_SEND #define crQUEUE_SEND(
```

```
    xHandle,
    pxQueue,
    pvItemToQueue,
    xTicksToWait,
    pxResult )
```

Value:

```
{
    *( pxResult ) = xQueueCRSend( ( pxQueue ) , ( pvItemToQueue ) , ( xTicksToWait ) );
    if( *( pxResult ) == errQUEUE_BLOCKED )
    {
        crSET_STATE0( ( xHandle ) );
        *pxResult = xQueueCRSend( ( pxQueue ) , ( pvItemToQueue ) , 0 );
    }
    if( *pxResult == errQUEUE_YIELD )
    {
        crSET_STATE1( ( xHandle ) );
        *pxResult = pdPASS;
    }
}
```



Definition at line 406 of file [croutine.h](#).

```
22.3.1.6 crQUEUE_SEND_FROM_ISR #define crQUEUE_SEND_FROM_ISR(  
    pxQueue,  
    pvItemToQueue,  
    xCoRoutinePreviouslyWoken ) xQueueCRSendFromISR( ( pxQueue ), ( pvItemToQueue ),  
( xCoRoutinePreviouslyWoken ) )
```

Definition at line 607 of file [croutine.h](#).

```
22.3.1.7 crSET_STATE0 #define crSET_STATE0(  
    xHandle ) ( ( corCRCB * )( xHandle ) )->uxState = ( __LINE__ * 2 ); return; case  
( __LINE__ * 2 ) :
```

Definition at line 267 of file [croutine.h](#).

```
22.3.1.8 crSET_STATE1 #define crSET_STATE1(  
    xHandle ) ( ( corCRCB * )( xHandle ) )->uxState = ( ( __LINE__ * 2 ) + 1 ); return;  
case ( ( __LINE__ * 2 ) + 1 ) :
```

Definition at line 268 of file [croutine.h](#).

```
22.3.1.9 crSTART #define crSTART(  
    pxCRCB ) switch( ( ( corCRCB * )( pxCRCB ) )->uxState ) { case 0:
```

Definition at line 230 of file [croutine.h](#).

22.3.2 Typedef Documentation

```
22.3.2.1 corCRCB typedef struct corCoRoutineControlBlock corCRCB
```

```
22.3.2.2 crCOROUTINE_CODE typedef void(* crCOROUTINE_CODE) ( xCoRoutineHandle, unsigned  
portBASE_TYPE )
```

Definition at line 73 of file [croutine.h](#).

```
22.3.2.3 xCoRoutineHandle typedef void* xCoRoutineHandle
```

Definition at line 70 of file [croutine.h](#).

22.3.3 Function Documentation

22.3.3.1 vCoRoutineAddToDelayedList() void vCoRoutineAddToDelayedList (portTickType xTicksToDelay, xList * pxEventList)

Definition at line 182 of file [croutine.c](#).

22.3.3.2 vCoRoutineSchedule() void vCoRoutineSchedule (void)

Definition at line 301 of file [croutine.c](#).

22.3.3.3 xCoRoutineCreate() signed portBASE_TYPE xCoRoutineCreate (crCOROUTINE_CODE pxCoRoutineCode, unsigned portBASE_TYPE uxPriority, unsigned portBASE_TYPE uxIndex)

Definition at line 125 of file [croutine.c](#).

Here is the call graph for this function:



22.3.3.4 xCoRoutineRemoveFromEventList() signed portBASE_TYPE xCoRoutineRemoveFromEventList (const xList * pxEventList)

Definition at line 351 of file [croutine.c](#).

Here is the call graph for this function:



22.4 croutine.h

```

00001 /*          *
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005 ****
00006 *
00007 *     FreeRTOS tutorial books are available in pdf and paperback. *
00008 *     Complete, revised, and edited pdf reference manuals are also *
00009 *     available. *
00010 *
00011 *     Purchasing FreeRTOS documentation will not only help you, by *
00012 *     ensuring you get running as quickly as possible and with an *
00013 *     in-depth knowledge of how to use FreeRTOS, it will also help *
00014 *     the FreeRTOS project to continue with its mission of providing *
00015 *     professional grade, cross platform, de facto standard solutions *
00016 *     for microcontrollers - completely free of charge! *
00017 *
00018 *     >>> See http://www.FreeRTOS.org/Documentation for details. << *
00019 *
00020 *     Thank you for using FreeRTOS, and thank you for your support! *
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 =>>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039 by writing to Richard Barry, contact details for whom are available on the
00040 FreeRTOS WEB site.
00041
00042 1 tab == 4 spaces!
00043
00044 http://www.FreeRTOS.org - Documentation, latest information, license and
00045 contact details.
00046
00047 http://www.SafeRTOS.com - A version that is certified for use in safety
00048 critical systems.
00049
00050 http://www.OpenRTOS.com - Commercial support, development, porting,
00051 licensing and training services.
00052 */
00053
00054 #ifndef CO_ROUTINE_H
00055 #define CO_ROUTINE_H
00056
00057 #ifndef INC_FREERTOS_H
00058     #error "include FreeRTOS.h must appear in source files before include croutine.h"
00059 #endif
00060
00061 #include "list.h"
00062
00063 #ifdef __cplusplus
00064 extern "C" {
00065 #endif
00066
00067 /* Used to hide the implementation of the co-routine control block. The
00068 control block structure however has to be included in the header due to
00069 the macro implementation of the co-routine functionality. */
00070 typedef void * xCoRoutineHandle;
00071
00072 /* Defines the prototype to which co-routine functions must conform. */
00073 typedef void (*crCOROUTINE_CODE)( xCoRoutineHandle, unsigned portBASE_TYPE );
00074
00075 typedef struct corCoRoutineControlBlock
00076 {
00077     crCOROUTINE_CODE      pxCoRoutineFunction;
00078     xListItem              xGenericListItem;    /*< List item used to place the CRCB in ready and
00079     blocked queues. */           xEventListItem;    /*< List item used to place the CRCB in event lists.
00080     */                     uxPriority;        /*< The priority of the co-routine in relation to
00081     other co-routines. */           uxIndex;         /*< Used to distinguish between co-routines when
00082     multiple co-routines use the same co-routine function. */

```

```

00082     unsigned short      uxState;           /*< Used internally by the co-routine implementation. */
00083 } corCRCB; /* Co-routine control block. Note must be identical in size down to uxPriority with
00084 tskTCB. */
00157 signed portBASE_TYPE xCoRoutineCreate( crCOROUTINE_CODE pxCoRoutineCode, unsigned portBASE_TYPE
00158 uxPriority, unsigned portBASE_TYPE uxIndex );
00159
00199 void vCoRoutineSchedule( void );
00200
00230 #define crSTART( pxCRCB ) switch( ( ( corCRCB * )( pxCRCB ) )->uxState ) { case 0:
00231
00261 #define crEND() }
00262
00263 /*
00264  * These macros are intended for internal use by the co-routine implementation
00265  * only. The macros should not be used directly by application writers.
00266 */
00267 #define crSET_STATE0( xHandle ) ( ( corCRCB * )( xHandle ) )->uxState = ( __LINE__ * 2); return; case
00268 ( __LINE__ * 2):
00269 #define crSET_STATE1( xHandle ) ( ( corCRCB * )( xHandle ) )->uxState = ( ( __LINE__ * 2)+1); return;
00270 case ( ( __LINE__ * 2)+1):
00271
00316 #define crDELAY( xHandle, xTicksToDelay )
00317     if( ( xTicksToDelay ) > 0 )
00318     {
00319         vCoRoutineAddToDelayedList( ( xTicksToDelay ), NULL );
00320     }
00321     crSET_STATE0( ( xHandle ) );
00322
00406 #define crQUEUE_SEND( xHandle, pxQueue, pvItemToQueue, xTicksToWait, pxResult )
00407 {
00408     *( pxResult ) = xQueueCRSend( ( pxQueue ), ( pvItemToQueue ), ( xTicksToWait ) );
00409     if( *( pxResult ) == errQUEUE_BLOCKED )
00410     {
00411         crSET_STATE0( ( xHandle ) );
00412         *pxResult = xQueueCRSend( ( pxQueue ), ( pvItemToQueue ), 0 );
00413     }
00414     if( *pxResult == errQUEUE_YIELD )
00415     {
00416         crSET_STATE1( ( xHandle ) );
00417         *pxResult = pdPASS;
00418     }
00419 }
00420
00498 #define crQUEUE_RECEIVE( xHandle, pxQueue, pvBuffer, xTicksToWait, pxResult )
00499 {
00500     *( pxResult ) = xQueueCRReceive( ( pxQueue ), ( pvBuffer ), ( xTicksToWait ) );
00501     if( *( pxResult ) == errQUEUE_BLOCKED )
00502     {
00503         crSET_STATE0( ( xHandle ) );
00504         *( pxResult ) = xQueueCRReceive( ( pxQueue ), ( pvBuffer ), 0 );
00505     }
00506     if( *( pxResult ) == errQUEUE_YIELD )
00507     {
00508         crSET_STATE1( ( xHandle ) );
00509         *( pxResult ) = pdPASS;
00510     }
00511 }
00512
00607 #define crQUEUE_SEND_FROM_ISR( pxQueue, pvItemToQueue, xCoRoutinePreviouslyWoken )
00608     xQueueCRSendFromISR( ( pxQueue ), ( pvItemToQueue ), ( xCoRoutinePreviouslyWoken ) )
00609
00720 #define crQUEUE_RECEIVE_FROM_ISR( pxQueue, pvBuffer, pxCoRoutineWoken ) xQueueCRReceiveFromISR( (
00721     pxQueue ), ( pvBuffer ), ( pxCoRoutineWoken ) )
00722
00723 /*
00724  * This function is intended for internal use by the co-routine macros only.
00725  * The macro nature of the co-routine implementation requires that the
00726  * prototype appears here. The function should not be used by application
00727  * writers.
00728  *
00729  * Removes the current co-routine from its ready list and places it in the
00730  * appropriate delayed list.
00731 */
00732 void vCoRoutineAddToDelayedList( portTickType xTicksToDelay, xList *pxEventList );
00733
00734 /*
00735  * This function is intended for internal use by the queue implementation only.
00736  * The function should not be used by application writers.
00737  *
00738  * Removes the highest priority co-routine from the event list and places it in
00739  * the pending ready list.
00740 */
00740 signed portBASE_TYPE xCoRoutineRemoveFromEventList( const xList *pxEventList );
00741

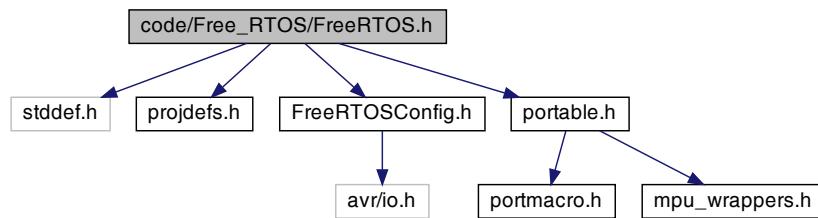
```

```
00742 #ifdef __cplusplus
00743 }
00744 #endif
00745
00746 #endif /* CO_ROUTINE_H */
```

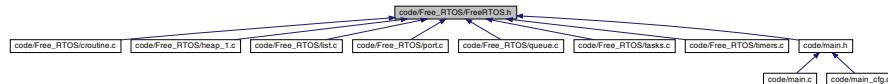
22.5 code/FreeRTOS/FreeRTOS.h File Reference

```
#include <stddef.h>
#include "projdefs.h"
#include "FreeRTOSConfig.h"
#include "portable.h"

Include dependency graph for FreeRTOS.h:
```



This graph shows which files directly or indirectly include this file:



Macros

- #define INCLUDE_xTaskGetIdleTaskHandle 0
- #define INCLUDE_xTimerGetTimerDaemonTaskHandle 0
- #define INCLUDE_pcTaskGetTaskName 0
- #define configUSE_APPLICATION_TASK_TAG 0
- #define INCLUDE uxTaskGetStackHighWaterMark 0
- #define configUSE_RECURSIVE_MUTEXES 0
- #define configUSE_MUTEXES 0
- #define configUSE_TIMERS 0
- #define configUSE_ALTERNATIVE_API 0
- #define portCRITICAL_NESTING_IN_TCB 0
- #define INCLUDE_xTaskResumeFromISR 1
- #define configASSERT(x)
- #define INCLUDE_xTaskGetSchedulerState 0
- #define INCLUDE_xTaskGetCurrentTaskHandle 0
- #define portSET_INTERRUPT_MASK_FROM_ISR() 0
- #define portCLEAR_INTERRUPT_MASK_FROM_ISR(uxSavedStatusValue) (void) uxSavedStatusValue
- #define vQueueAddToRegistry(xQueue, pcName)
- #define vQueueUnregisterQueue(xQueue)

- `#define portPOINTER_SIZE_TYPE` unsigned long
- `#define traceSTART()`
- `#define traceEND()`
- `#define traceTASK_SWITCHED_IN()`
- `#define traceTASK_SWITCHED_OUT()`
- `#define traceBLOCKING_ON_QUEUE_RECEIVE(pxQueue)`
- `#define traceBLOCKING_ON_QUEUE_SEND(pxQueue)`
- `#define configCHECK_FOR_STACK_OVERFLOW 0`
- `#define traceQUEUE_CREATE(pxNewQueue)`
- `#define traceQUEUE_CREATE_FAILED()`
- `#define traceCREATE_MUTEX(pxNewQueue)`
- `#define traceCREATE_MUTEX_FAILED()`
- `#define traceGIVE_MUTEX_RECURSIVE(pxMutex)`
- `#define traceGIVE_MUTEX_RECURSIVE_FAILED(pxMutex)`
- `#define traceTAKE_MUTEX_RECURSIVE(pxMutex)`
- `#define traceTAKE_MUTEX_RECURSIVE_FAILED(pxMutex)`
- `#define traceCREATE_COUNTING_SEMAPHORE()`
- `#define traceCREATE_COUNTING_SEMAPHORE_FAILED()`
- `#define traceQUEUE_SEND(pxQueue)`
- `#define traceQUEUE_SEND_FAILED(pxQueue)`
- `#define traceQUEUE_RECEIVE(pxQueue)`
- `#define traceQUEUE_PEEK(pxQueue)`
- `#define traceQUEUE_RECEIVE_FAILED(pxQueue)`
- `#define traceQUEUE_SEND_FROM_ISR(pxQueue)`
- `#define traceQUEUE_SEND_FROM_ISR_FAILED(pxQueue)`
- `#define traceQUEUE_RECEIVE_FROM_ISR(pxQueue)`
- `#define traceQUEUE_RECEIVE_FROM_ISR_FAILED(pxQueue)`
- `#define traceQUEUE_DELETE(pxQueue)`
- `#define traceTASK_CREATE(pxNewTCB)`
- `#define traceTASK_CREATE_FAILED()`
- `#define traceTASK_DELETE(pxTaskToDelete)`
- `#define traceTASK_DELAY_UNTIL()`
- `#define traceTASK_DELAY()`
- `#define traceTASK_PRIORITY_SET(pxTask, uxNewPriority)`
- `#define traceTASK_SUSPEND(pxTaskToSuspend)`
- `#define traceTASK_RESUME(pxTaskToResume)`
- `#define traceTASK_RESUME_FROM_ISR(pxTaskToResume)`
- `#define traceTASK_INCREMENT_TICK(xTickCount)`
- `#define traceTIMER_CREATE(pxNewTimer)`
- `#define traceTIMER_CREATE_FAILED()`
- `#define traceTIMER_COMMAND_SEND(xTimer, xMessageID, xMessageValueValue, xReturn)`
- `#define traceTIMER_EXPIRED(pxTimer)`
- `#define traceTIMER_COMMAND_RECEIVED(pxTimer, xMessageID, xMessageValue)`
- `#define configGENERATE_RUN_TIME_STATS 0`
- `#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()`
- `#define configUSE_MALLOC_FAILED_HOOK 0`
- `#define portPRIVILEGE_BIT ((unsigned portBASE_TYPE) 0x00)`
- `#define portYIELD_WITHIN_API portYIELD`
- `#define pvPortMallocAligned(x, puxStackBuffer) (((puxStackBuffer) == NULL) ? (pvPortMalloc((x))) : (puxStackBuffer))`
- `#define vPortFreeAligned(pvBlockToFree) vPortFree(pvBlockToFree)`

Typedefs

- `typedef portBASE_TYPE(* pdTASK_HOOK_CODE) (void *)`

22.5.1 Macro Definition Documentation

22.5.1.1 configASSERT `#define configASSERT(
x)`

Definition at line [192](#) of file [FreeRTOS.h](#).

22.5.1.2 configCHECK_FOR_STACK_OVERFLOW `#define configCHECK_FOR_STACK_OVERFLOW 0`

Definition at line [285](#) of file [FreeRTOS.h](#).

22.5.1.3 configGENERATE_RUN_TIME_STATS `#define configGENERATE_RUN_TIME_STATS 0`

Definition at line [431](#) of file [FreeRTOS.h](#).

22.5.1.4 configUSE_ALTERNATIVE_API `#define configUSE_ALTERNATIVE_API 0`

Definition at line [168](#) of file [FreeRTOS.h](#).

22.5.1.5 configUSE_APPLICATION_TASK_TAG `#define configUSE_APPLICATION_TASK_TAG 0`

Definition at line [144](#) of file [FreeRTOS.h](#).

22.5.1.6 configUSE_MALLOC_FAILED_HOOK `#define configUSE_MALLOC_FAILED_HOOK 0`

Definition at line [453](#) of file [FreeRTOS.h](#).

22.5.1.7 configUSE_MUTEXES `#define configUSE_MUTEXES 0`

Definition at line [156](#) of file [FreeRTOS.h](#).

22.5.1.8 configUSE_RECURSIVE_MUTEXES #define configUSE_RECURSIVE_MUTEXES 0

Definition at line 152 of file [FreeRTOS.h](#).

22.5.1.9 configUSE_TIMERS #define configUSE_TIMERS 0

Definition at line 160 of file [FreeRTOS.h](#).

22.5.1.10 INCLUDE_pcTaskGetTaskName #define INCLUDE_pcTaskGetTaskName 0

Definition at line 140 of file [FreeRTOS.h](#).

22.5.1.11 INCLUDE_uxTaskGetStackHighWaterMark #define INCLUDE_uxTaskGetStackHighWaterMark 0

Definition at line 148 of file [FreeRTOS.h](#).

22.5.1.12 INCLUDE_xTaskGetCurrentTaskHandle #define INCLUDE_xTaskGetCurrentTaskHandle 0

Definition at line 217 of file [FreeRTOS.h](#).

22.5.1.13 INCLUDE_xTaskGetIdleTaskHandle #define INCLUDE_xTaskGetIdleTaskHandle 0

Definition at line 132 of file [FreeRTOS.h](#).

22.5.1.14 INCLUDE_xTaskGetSchedulerState #define INCLUDE_xTaskGetSchedulerState 0

Definition at line 213 of file [FreeRTOS.h](#).

22.5.1.15 INCLUDE_xTaskResumeFromISR #define INCLUDE_xTaskResumeFromISR 1

Definition at line 188 of file [FreeRTOS.h](#).

22.5.1.16 INCLUDE_xTimerGetTimerDaemonTaskHandle #define INCLUDE_xTimerGetTimerDaemonTaskHandle 0

Definition at line 136 of file [FreeRTOS.h](#).

22.5.1.17 portCLEAR_INTERRUPT_MASK_FROM_ISR #define portCLEAR_INTERRUPT_MASK_FROM_ISR(uxSavedStatusValue) (void) uxSavedStatusValue

Definition at line 226 of file [FreeRTOS.h](#).

22.5.1.18 portCONFIGURE_TIMER_FOR_RUN_TIME_STATS #define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()

Definition at line 449 of file [FreeRTOS.h](#).

22.5.1.19 portCRITICAL_NESTING_IN_TCB #define portCRITICAL_NESTING_IN_TCB 0

Definition at line 172 of file [FreeRTOS.h](#).

22.5.1.20 portPOINTER_SIZE_TYPE #define portPOINTER_SIZE_TYPE unsigned long

Definition at line 240 of file [FreeRTOS.h](#).

22.5.1.21 portPRIVILEGE_BIT #define portPRIVILEGE_BIT ((unsigned portBASE_TYPE) 0x00)

Definition at line 457 of file [FreeRTOS.h](#).

22.5.1.22 portSET_INTERRUPT_MASK_FROM_ISR #define portSET_INTERRUPT_MASK_FROM_ISR() 0

Definition at line 222 of file [FreeRTOS.h](#).

22.5.1.23 portYIELD_WITHIN_API #define portYIELD_WITHIN_API portYIELD

Definition at line 461 of file [FreeRTOS.h](#).

```
22.5.1.24 pvPortMallocAligned #define pvPortMallocAligned(  
    x,  
    puxStackBuffer ) ( ( ( puxStackBuffer ) == NULL ) ? ( pvPortMalloc\( \( x \) \) ) :  
( puxStackBuffer ) )
```

Definition at line [465](#) of file [FreeRTOS.h](#).

```
22.5.1.25 traceBLOCKING_ON_QUEUE_RECEIVE #define traceBLOCKING_ON_QUEUE_RECEIVE(  
    pxQueue )
```

Definition at line [273](#) of file [FreeRTOS.h](#).

```
22.5.1.26 traceBLOCKING_ON_QUEUE_SEND #define traceBLOCKING_ON_QUEUE_SEND(  
    pxQueue )
```

Definition at line [281](#) of file [FreeRTOS.h](#).

```
22.5.1.27 traceCREATE_COUNTING_SEMAPHORE #define traceCREATE_COUNTING_SEMAPHORE( )
```

Definition at line [323](#) of file [FreeRTOS.h](#).

```
22.5.1.28 traceCREATE_COUNTING_SEMAPHORE_FAILED #define traceCREATE_COUNTING_SEMAPHORE_FAILED( )
```

Definition at line [327](#) of file [FreeRTOS.h](#).

```
22.5.1.29 traceCREATE_MUTEX #define traceCREATE_MUTEX(  
    pxNewQueue )
```

Definition at line [299](#) of file [FreeRTOS.h](#).

```
22.5.1.30 traceCREATE_MUTEX_FAILED #define traceCREATE_MUTEX_FAILED( )
```

Definition at line [303](#) of file [FreeRTOS.h](#).

22.5.1.31 traceEND #define traceEND()

Definition at line 253 of file [FreeRTOS.h](#).

22.5.1.32 traceGIVE_MUTEX_RECURSIVE #define traceGIVE_MUTEX_RECURSIVE(
pxMutex)

Definition at line 307 of file [FreeRTOS.h](#).

22.5.1.33 traceGIVE_MUTEX_RECURSIVE_FAILED #define traceGIVE_MUTEX_RECURSIVE_FAILED(
pxMutex)

Definition at line 311 of file [FreeRTOS.h](#).

22.5.1.34 traceQUEUE_CREATE #define traceQUEUE_CREATE(
pxNewQueue)

Definition at line 291 of file [FreeRTOS.h](#).

22.5.1.35 traceQUEUE_CREATE_FAILED #define traceQUEUE_CREATE_FAILED()

Definition at line 295 of file [FreeRTOS.h](#).

22.5.1.36 traceQUEUE_DELETE #define traceQUEUE_DELETE(
pxQueue)

Definition at line 367 of file [FreeRTOS.h](#).

22.5.1.37 traceQUEUE_PEEK #define traceQUEUE_PEEK(
pxQueue)

Definition at line 343 of file [FreeRTOS.h](#).

```
22.5.1.38 traceQUEUE_RECEIVE #define traceQUEUE_RECEIVE(  
    pxQueue )
```

Definition at line 339 of file [FreeRTOS.h](#).

```
22.5.1.39 traceQUEUE_RECEIVE_FAILED #define traceQUEUE_RECEIVE_FAILED(  
    pxQueue )
```

Definition at line 347 of file [FreeRTOS.h](#).

```
22.5.1.40 traceQUEUE_RECEIVE_FROM_ISR #define traceQUEUE_RECEIVE_FROM_ISR(  
    pxQueue )
```

Definition at line 359 of file [FreeRTOS.h](#).

```
22.5.1.41 traceQUEUE_RECEIVE_FROM_ISR_FAILED #define traceQUEUE_RECEIVE_FROM_ISR_FAILED(  
    pxQueue )
```

Definition at line 363 of file [FreeRTOS.h](#).

```
22.5.1.42 traceQUEUE_SEND #define traceQUEUE_SEND(  
    pxQueue )
```

Definition at line 331 of file [FreeRTOS.h](#).

```
22.5.1.43 traceQUEUE_SEND_FAILED #define traceQUEUE_SEND_FAILED(  
    pxQueue )
```

Definition at line 335 of file [FreeRTOS.h](#).

```
22.5.1.44 traceQUEUE_SEND_FROM_ISR #define traceQUEUE_SEND_FROM_ISR(  
    pxQueue )
```

Definition at line 351 of file [FreeRTOS.h](#).

```
22.5.1.45 traceQUEUE_SEND_FROM_ISR_FAILED #define traceQUEUE_SEND_FROM_ISR_FAILED( pxQueue )
```

Definition at line [355](#) of file [FreeRTOS.h](#).

```
22.5.1.46 traceSTART #define traceSTART( )
```

Definition at line [247](#) of file [FreeRTOS.h](#).

```
22.5.1.47 traceTAKE_MUTEX_RECURSIVE #define traceTAKE_MUTEX_RECURSIVE( pxMutex )
```

Definition at line [315](#) of file [FreeRTOS.h](#).

```
22.5.1.48 traceTAKE_MUTEX_RECURSIVE_FAILED #define traceTAKE_MUTEX_RECURSIVE_FAILED( pxMutex )
```

Definition at line [319](#) of file [FreeRTOS.h](#).

```
22.5.1.49 traceTASK_CREATE #define traceTASK_CREATE( pxNewTCB )
```

Definition at line [371](#) of file [FreeRTOS.h](#).

```
22.5.1.50 traceTASK_CREATE_FAILED #define traceTASK_CREATE_FAILED( )
```

Definition at line [375](#) of file [FreeRTOS.h](#).

```
22.5.1.51 traceTASK_DELAY #define traceTASK_DELAY( )
```

Definition at line [387](#) of file [FreeRTOS.h](#).

```
22.5.1.52 traceTASK_DELAY_UNTIL #define traceTASK_DELAY_UNTIL( )
```

Definition at line [383](#) of file [FreeRTOS.h](#).

```
22.5.1.53 traceTASK_DELETE #define traceTASK_DELETE(  
    pxTaskToDelete )
```

Definition at line 379 of file [FreeRTOS.h](#).

```
22.5.1.54 traceTASK_INCREMENT_TICK #define traceTASK_INCREMENT_TICK(  
    xTickCount )
```

Definition at line 407 of file [FreeRTOS.h](#).

```
22.5.1.55 traceTASK_PRIORITY_SET #define traceTASK_PRIORITY_SET(  
    pxTask,  
    uxNewPriority )
```

Definition at line 391 of file [FreeRTOS.h](#).

```
22.5.1.56 traceTASK_RESUME #define traceTASK_RESUME(  
    pxTaskToResume )
```

Definition at line 399 of file [FreeRTOS.h](#).

```
22.5.1.57 traceTASK_RESUME_FROM_ISR #define traceTASK_RESUME_FROM_ISR(  
    pxTaskToResume )
```

Definition at line 403 of file [FreeRTOS.h](#).

```
22.5.1.58 traceTASK_SUSPEND #define traceTASK_SUSPEND(  
    pxTaskToSuspend )
```

Definition at line 395 of file [FreeRTOS.h](#).

```
22.5.1.59 traceTASK_SWITCHED_IN #define traceTASK_SWITCHED_IN( )
```

Definition at line 259 of file [FreeRTOS.h](#).

22.5.1.60 traceTASK_SWITCHED_OUT #define traceTASK_SWITCHED_OUT()

Definition at line 265 of file [FreeRTOS.h](#).

22.5.1.61 traceTIMER_COMMAND_RECEIVED #define traceTIMER_COMMAND_RECEIVED(pxTimer,
xMessageID,
xMessageValue)

Definition at line 427 of file [FreeRTOS.h](#).

22.5.1.62 traceTIMER_COMMAND_SEND #define traceTIMER_COMMAND_SEND(xTimer,
xMessageID,
xMessageValueValue,
xReturn)

Definition at line 419 of file [FreeRTOS.h](#).

22.5.1.63 traceTIMER_CREATE #define traceTIMER_CREATE(pxNewTimer)

Definition at line 411 of file [FreeRTOS.h](#).

22.5.1.64 traceTIMER_CREATE_FAILED #define traceTIMER_CREATE_FAILED()

Definition at line 415 of file [FreeRTOS.h](#).

22.5.1.65 traceTIMER_EXPIRED #define traceTIMER_EXPIRED(pxTimer)

Definition at line 423 of file [FreeRTOS.h](#).

22.5.1.66 vPortFreeAligned #define vPortFreeAligned(pvBlockToFree) vPortFree(pvBlockToFree)

Definition at line 469 of file [FreeRTOS.h](#).

```
22.5.1.67 vQueueAddToRegistry #define vQueueAddToRegistry( xQueue,  
pcName )
```

Definition at line 235 of file [FreeRTOS.h](#).

```
22.5.1.68 vQueueUnregisterQueue #define vQueueUnregisterQueue( xQueue )
```

Definition at line 236 of file [FreeRTOS.h](#).

22.5.2 Typedef Documentation

```
22.5.2.1 pdTASK_HOOK_CODE typedef portBASE_TYPE(* pdTASK_HOOK_CODE) (void *)
```

Definition at line 75 of file [FreeRTOS.h](#).

22.6 FreeRTOS.h

```
00001 /*  
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.  
00003  
00004  
00005 ****  
00006 *  
00007 *   FreeRTOS tutorial books are available in pdf and paperback.  
00008 *   Complete, revised, and edited pdf reference manuals are also  
00009 *   available.  
00010 *  
00011 *   Purchasing FreeRTOS documentation will not only help you, by  
00012 *   ensuring you get running as quickly as possible and with an  
00013 *   in-depth knowledge of how to use FreeRTOS, it will also help  
00014 *   the FreeRTOS project to continue with its mission of providing  
00015 *   professional grade, cross platform, de facto standard solutions  
00016 *   for microcontrollers - completely free of charge!  
00017 *  
00018 *   >> See http://www.FreeRTOS.org/Documentation for details. <<  
00019 *  
00020 *   Thank you for using FreeRTOS, and thank you for your support!  
00021 *  
00022 ****  
00023  
00024  
00025 This file is part of the FreeRTOS distribution.  
00026  
00027 FreeRTOS is free software; you can redistribute it and/or modify it under  
00028 the terms of the GNU General Public License (version 2) as published by the  
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.  
00030 >>NOTE<< The modification to the GPL is included to allow you to  
00031 distribute a combined work that includes FreeRTOS without being obliged to  
00032 provide the source code for proprietary components outside of the FreeRTOS  
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but  
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY  
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for  
00036 more details. You should have received a copy of the GNU General Public  
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it  
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained  
00039 by writing to Richard Barry, contact details for whom are available on the  
00040 FreeRTOS WEB site.  
00041  
00042 1 tab == 4 spaces!  
00043  
00044 http://www.FreeRTOS.org - Documentation, latest information, license and  
00045 contact details.  
00046
```

```

00047     http://www.SafeRTOS.com - A version that is certified for use in safety
00048     critical systems.
00049
00050     http://www.OpenRTOS.com - Commercial support, development, porting,
00051     licensing and training services.
00052 */
00053
00054 #ifndef INC_FREERTOS_H
00055 #define INC_FREERTOS_H
00056
00057
00058 /*
00059  * Include the generic headers required for the FreeRTOS port being used.
00060  */
00061 #include <stddef.h>
00062
00063 /* Basic FreeRTOS definitions. */
00064 #include "projdefs.h"
00065
00066 /* Application specific configuration options. */
00067 #include "FreeRTOSConfig.h"
00068
00069 /* Definitions specific to the port being used. */
00070 #include "portable.h"
00071
00072
00073 /* Defines the prototype to which the application task hook function must
00074 conform. */
00075 typedef portBASE_TYPE (*pdTASK_HOOK_CODE)( void * );
00076
00077
00078
00079
00080
00081 /*
00082  * Check all the required application specific macros have been defined.
00083  * These macros are application specific and (as downloaded) are defined
00084  * within FreeRTOSConfig.h.
00085 */
00086
00087 #ifndef configUSE_PREEMPTION
00088     #error Missing definition: configUSE_PREEMPTION should be defined in FreeRTOSConfig.h as either 1
00089     or 0. See the Configuration section of the FreeRTOS API documentation for details.
00090 #endif
00091 #ifndef configUSE_IDLE_HOOK
00092     #error Missing definition: configUSE_IDLE_HOOK should be defined in FreeRTOSConfig.h as either 1
00093     or 0. See the Configuration section of the FreeRTOS API documentation for details.
00094 #endif
00095 #ifndef configUSE_TICK_HOOK
00096     #error Missing definition: configUSE_TICK_HOOK should be defined in FreeRTOSConfig.h as either 1
00097     or 0. See the Configuration section of the FreeRTOS API documentation for details.
00098
00099 #ifndef configUSE_CO_Routines
00100     #error Missing definition: configUSE_CO_Routines should be defined in FreeRTOSConfig.h as either
00101     1 or 0. See the Configuration section of the FreeRTOS API documentation for details.
00102 #endif
00103 #ifndef INCLUDE_vTaskPrioritySet
00104     #error Missing definition: INCLUDE_vTaskPrioritySet should be defined in FreeRTOSConfig.h as
00105     either 1 or 0. See the Configuration section of the FreeRTOS API documentation for details.
00106
00107 #ifndef INCLUDE_uxTaskPriorityGet
00108     #error Missing definition: INCLUDE_uxTaskPriorityGet should be defined in FreeRTOSConfig.h as
00109     either 1 or 0. See the Configuration section of the FreeRTOS API documentation for details.
00110 #endif
00111 #ifndef INCLUDE_vTaskDelete
00112     #error Missing definition: INCLUDE_vTaskDelete      should be defined in FreeRTOSConfig.h as
00113     either 1 or 0. See the Configuration section of the FreeRTOS API documentation for details.
00114 #endif
00115 #ifndef INCLUDE_vTaskSuspend
00116     #error Missing definition: INCLUDE_vTaskSuspend      should be defined in FreeRTOSConfig.h as
00117     either 1 or 0. See the Configuration section of the FreeRTOS API documentation for details.
00118
00119 #ifndef INCLUDE_vTaskDelayUntil
00120     #error Missing definition: INCLUDE_vTaskDelayUntil should be defined in FreeRTOSConfig.h as
00121     either 1 or 0. See the Configuration section of the FreeRTOS API documentation for details.
00122 #endif
00123 #ifndef INCLUDE_vTaskDelay
00124     #error Missing definition: INCLUDE_vTaskDelay should be defined in FreeRTOSConfig.h as either 1

```

```
    or 0. See the Configuration section of the FreeRTOS API documentation for details.  
00125 #endif  
00126  
00127 #ifndef configUSE_16_BIT_TICKS  
00128     #error Missing definition: configUSE_16_BIT_TICKS should be defined in FreeRTOSConfig.h as either  
     1 or 0. See the Configuration section of the FreeRTOS API documentation for details.  
00129 #endif  
00130  
00131 #ifndef INCLUDE_xTaskGetIdleTaskHandle  
00132     #define INCLUDE_xTaskGetIdleTaskHandle 0  
00133 #endif  
00134  
00135 #ifndef INCLUDE_xTimerGetTimerDaemonTaskHandle  
00136     #define INCLUDE_xTimerGetTimerDaemonTaskHandle 0  
00137 #endif  
00138  
00139 #ifndef INCLUDE_pcTaskGetTaskName  
00140     #define INCLUDE_pcTaskGetTaskName 0  
00141 #endif  
00142  
00143 #ifndef configUSE_APPLICATION_TASK_TAG  
00144     #define configUSE_APPLICATION_TASK_TAG 0  
00145 #endif  
00146  
00147 #ifndef INCLUDE_uxTaskGetStackHighWaterMark  
00148     #define INCLUDE_uxTaskGetStackHighWaterMark 0  
00149 #endif  
00150  
00151 #ifndef configUSE_RECURSIVE_MUTEXES  
00152     #define configUSE_RECURSIVE_MUTEXES 0  
00153 #endif  
00154  
00155 #ifndef configUSE_MUTEXES  
00156     #define configUSE_MUTEXES 0  
00157 #endif  
00158  
00159 #ifndef configUSE_TIMERS  
00160     #define configUSE_TIMERS 0  
00161 #endif  
00162  
00163 #ifndef configUSE_COUNTING_SEMAPHORES  
00164     #define configUSE_COUNTING_SEMAPHORES 0  
00165 #endif  
00166  
00167 #ifndef configUSE_ALTERNATIVE_API  
00168     #define configUSE_ALTERNATIVE_API 0  
00169 #endif  
00170  
00171 #ifndef portCRITICAL_NESTING_IN_TCB  
00172     #define portCRITICAL_NESTING_IN_TCB 0  
00173 #endif  
00174  
00175 #ifndef configMAX_TASK_NAME_LEN  
00176     #define configMAX_TASK_NAME_LEN 16  
00177 #endif  
00178  
00179 #ifndef configIDLE_SHOULD_YIELD  
00180     #define configIDLE_SHOULD_YIELD      1  
00181 #endif  
00182  
00183 #if configMAX_TASK_NAME_LEN < 1  
00184     #error configMAX_TASK_NAME_LEN must be set to a minimum of 1 in FreeRTOSConfig.h  
00185 #endif  
00186  
00187 #ifndef INCLUDE_xTaskResumeFromISR  
00188     #define INCLUDE_xTaskResumeFromISR 1  
00189 #endif  
00190  
00191 #ifndef configASSERT  
00192     #define configASSERT( x )  
00193 #endif  
00194  
00195 /* The timers module relies on xTaskGetSchedulerState(). */  
00196 #if configUSE_TIMERS == 1  
00197  
00198     #ifndef configTIMER_TASK_PRIORITY  
00199         #error If configUSE_TIMERS is set to 1 then configTIMER_TASK_PRIORITY must also be defined.  
00200     #endif /* configTIMER_TASK_PRIORITY */  
00201  
00202     #ifndef configTIMER_QUEUE_LENGTH  
00203         #error If configUSE_TIMERS is set to 1 then configTIMER_QUEUE_LENGTH must also be defined.  
00204     #endif /* configTIMER_QUEUE_LENGTH */  
00205  
00206     #ifndef configTIMER_TASK_STACK_DEPTH  
00207         #error If configUSE_TIMERS is set to 1 then configTIMER_TASK_STACK_DEPTH must also be defined.  
00208     #endif /* configTIMER_TASK_STACK_DEPTH */  
00209
```

```

00210 #endif /* configUSE_TIMERS */
00211
00212 #ifndef INCLUDE_xTaskGetSchedulerState
00213     #define INCLUDE_xTaskGetSchedulerState 0
00214 #endif
00215
00216 #ifndef INCLUDE_xTaskGetCurrentTaskHandle
00217     #define INCLUDE_xTaskGetCurrentTaskHandle 0
00218 #endif
00219
00220
00221 #ifndef portSET_INTERRUPT_MASK_FROM_ISR
00222     #define portSET_INTERRUPT_MASK_FROM_ISR() 0
00223 #endif
00224
00225 #ifndef portCLEAR_INTERRUPT_MASK_FROM_ISR
00226     #define portCLEAR_INTERRUPT_MASK_FROM_ISR( uxSavedStatusValue ) ( void ) uxSavedStatusValue
00227 #endif
00228
00229
00230 #ifndef configQUEUE_REGISTRY_SIZE
00231     #define configQUEUE_REGISTRY_SIZE 0U
00232 #endif
00233
00234 #if ( configQUEUE_REGISTRY_SIZE < 1 )
00235     #define vQueueAddToRegistry( xQueue, pcName )
00236     #define vQueueUnregisterQueue( xQueue )
00237 #endif
00238
00239 #ifndef portPOINTER_SIZE_TYPE
00240     #define portPOINTER_SIZE_TYPE unsigned long
00241 #endif
00242
00243 /* Remove any unused trace macros. */
00244 #ifndef traceSTART
00245     /* Used to perform any necessary initialisation - for example, open a file
00246     into which trace is to be written. */
00247     #define traceSTART()
00248 #endif
00249
00250 #ifndef traceEND
00251     /* Use to close a trace, for example close a file into which trace has been
00252     written. */
00253     #define traceEND()
00254 #endif
00255
00256 #ifndef traceTASK_SWITCHED_IN
00257     /* Called after a task has been selected to run. pxCurrentTCB holds a pointer
00258     to the task control block of the selected task. */
00259     #define traceTASK_SWITCHED_IN()
00260 #endif
00261
00262 #ifndef traceTASK_SWITCHED_OUT
00263     /* Called before a task has been selected to run. pxCurrentTCB holds a pointer
00264     to the task control block of the task being switched out. */
00265     #define traceTASK_SWITCHED_OUT()
00266 #endif
00267
00268 #ifndef traceBLOCKING_ON_QUEUE_RECEIVE
00269     /* Task is about to block because it cannot read from a
00270     queue/mutex/semaphore. pxQueue is a pointer to the queue/mutex/semaphore
00271     upon which the read was attempted. pxCurrentTCB points to the TCB of the
00272     task that attempted the read. */
00273     #define traceBLOCKING_ON_QUEUE_RECEIVE( pxQueue )
00274 #endif
00275
00276 #ifndef traceBLOCKING_ON_QUEUE_SEND
00277     /* Task is about to block because it cannot write to a
00278     queue/mutex/semaphore. pxQueue is a pointer to the queue/mutex/semaphore
00279     upon which the write was attempted. pxCurrentTCB points to the TCB of the
00280     task that attempted the write. */
00281     #define traceBLOCKING_ON_QUEUE_SEND( pxQueue )
00282 #endif
00283
00284 #ifndef configCHECK_FOR_STACK_OVERFLOW
00285     #define configCHECK_FOR_STACK_OVERFLOW 0
00286 #endif
00287
00288 /* The following event macros are embedded in the kernel API calls. */
00289
00290 #ifndef traceQUEUE_CREATE
00291     #define traceQUEUE_CREATE( pxNewQueue )
00292 #endif
00293
00294 #ifndef traceQUEUE_CREATE_FAILED
00295     #define traceQUEUE_CREATE_FAILED()
00296 #endif

```

```
00297
00298 #ifndef traceCREATE_MUTEX
00299     #define traceCREATE_MUTEX( pxNewQueue )
00300 #endif
00301
00302 #ifndef traceCREATE_MUTEX_FAILED
00303     #define traceCREATE_MUTEX_FAILED()
00304 #endif
00305
00306 #ifndef traceGIVE_MUTEX_RECURSIVE
00307     #define traceGIVE_MUTEX_RECURSIVE( pxMutex )
00308 #endif
00309
00310 #ifndef traceGIVE_MUTEX_RECURSIVE_FAILED
00311     #define traceGIVE_MUTEX_RECURSIVE_FAILED( pxMutex )
00312 #endif
00313
00314 #ifndef traceTAKE_MUTEX_RECURSIVE
00315     #define traceTAKE_MUTEX_RECURSIVE( pxMutex )
00316 #endif
00317
00318 #ifndef traceTAKE_MUTEX_RECURSIVE_FAILED
00319     #define traceTAKE_MUTEX_RECURSIVE_FAILED( pxMutex )
00320 #endif
00321
00322 #ifndef traceCREATE_COUNTING_SEMAPHORE
00323     #define traceCREATE_COUNTING_SEMAPHORE()
00324 #endif
00325
00326 #ifndef traceCREATE_COUNTING_SEMAPHORE_FAILED
00327     #define traceCREATE_COUNTING_SEMAPHORE_FAILED()
00328 #endif
00329
00330 #ifndef traceQUEUE_SEND
00331     #define traceQUEUE_SEND( pxQueue )
00332 #endif
00333
00334 #ifndef traceQUEUE_SEND_FAILED
00335     #define traceQUEUE_SEND_FAILED( pxQueue )
00336 #endif
00337
00338 #ifndef traceQUEUE_RECEIVE
00339     #define traceQUEUE_RECEIVE( pxQueue )
00340 #endif
00341
00342 #ifndef traceQUEUE_PEEK
00343     #define traceQUEUE_PEEK( pxQueue )
00344 #endif
00345
00346 #ifndef traceQUEUE_RECEIVE_FAILED
00347     #define traceQUEUE_RECEIVE_FAILED( pxQueue )
00348 #endif
00349
00350 #ifndef traceQUEUE_SEND_FROM_ISR
00351     #define traceQUEUE_SEND_FROM_ISR( pxQueue )
00352 #endif
00353
00354 #ifndef traceQUEUE_SEND_FROM_ISR_FAILED
00355     #define traceQUEUE_SEND_FROM_ISR_FAILED( pxQueue )
00356 #endif
00357
00358 #ifndef traceQUEUE_RECEIVE_FROM_ISR
00359     #define traceQUEUE_RECEIVE_FROM_ISR( pxQueue )
00360 #endif
00361
00362 #ifndef traceQUEUE_RECEIVE_FROM_ISR_FAILED
00363     #define traceQUEUE_RECEIVE_FROM_ISR_FAILED( pxQueue )
00364 #endif
00365
00366 #ifndef traceQUEUE_DELETE
00367     #define traceQUEUE_DELETE( pxQueue )
00368 #endif
00369
00370 #ifndef traceTASK_CREATE
00371     #define traceTASK_CREATE( pxNewTCB )
00372 #endif
00373
00374 #ifndef traceTASK_CREATE_FAILED
00375     #define traceTASK_CREATE_FAILED()
00376 #endif
00377
00378 #ifndef traceTASK_DELETE
00379     #define traceTASK_DELETE( pxTaskToDelete )
00380 #endif
00381
00382 #ifndef traceTASK_DELAY_UNTIL
00383     #define traceTASK_DELAY_UNTIL()
```

```

00384 #endif
00385
00386 #ifndef traceTASK_DELAY
00387     #define traceTASK_DELAY()
00388 #endif
00389
00390 #ifndef traceTASK_PRIORITY_SET
00391     #define traceTASK_PRIORITY_SET( pxTask, uxNewPriority )
00392 #endif
00393
00394 #ifndef traceTASK_SUSPEND
00395     #define traceTASK_SUSPEND( pxTaskToSuspend )
00396 #endif
00397
00398 #ifndef traceTASK_RESUME
00399     #define traceTASK_RESUME( pxTaskToResume )
00400 #endif
00401
00402 #ifndef traceTASK_RESUME_FROM_ISR
00403     #define traceTASK_RESUME_FROM_ISR( pxTaskToResume )
00404 #endif
00405
00406 #ifndef traceTASK_INCREMENT_TICK
00407     #define traceTASK_INCREMENT_TICK( xTickCount )
00408 #endif
00409
00410 #ifndef traceTIMER_CREATE
00411     #define traceTIMER_CREATE( pxNewTimer )
00412 #endif
00413
00414 #ifndef traceTIMER_CREATE_FAILED
00415     #define traceTIMER_CREATE_FAILED()
00416 #endif
00417
00418 #ifndef traceTIMER_COMMAND_SEND
00419     #define traceTIMER_COMMAND_SEND( xTimer, xMessageID, xMessageValueValue, xReturn )
00420 #endif
00421
00422 #ifndef traceTIMER_EXPIRED
00423     #define traceTIMER_EXPIRED( pxTimer )
00424 #endif
00425
00426 #ifndef traceTIMER_COMMAND RECEIVED
00427     #define traceTIMER_COMMAND RECEIVED( pxTimer, xMessageID, xMessageValue )
00428 #endif
00429
00430 #ifndef configGENERATE_RUN_TIME_STATS
00431     #define configGENERATE_RUN_TIME_STATS 0
00432 #endif
00433
00434 #if ( configGENERATE_RUN_TIME_STATS == 1 )
00435
00436     #ifndef portCONFIGURE_TIMER_FOR_RUN_TIME_STATS
00437         #error If configGENERATE_RUN_TIME_STATS is defined then portCONFIGURE_TIMER_FOR_RUN_TIME_STATS must also be defined. portCONFIGURE_TIMER_FOR_RUN_TIME_STATS should call a port layer function to setup a peripheral timer/counter that can then be used as the run time counter time base.
00438     #endif /* portCONFIGURE_TIMER_FOR_RUN_TIME_STATS */
00439
00440     #ifndef portGET_RUN_TIME_COUNTER_VALUE
00441         #ifndef portALT_GET_RUN_TIME_COUNTER_VALUE
00442             #error If configGENERATE_RUN_TIME_STATS is defined then either portGET_RUN_TIME_COUNTER_VALUE or portALT_GET_RUN_TIME_COUNTER_VALUE must also be defined. See the examples provided and the FreeRTOS web site for more information.
00443         #endif /* portALT_GET_RUN_TIME_COUNTER_VALUE */
00444     #endif /* portGET_RUN_TIME_COUNTER_VALUE */
00445
00446 #endif /* configGENERATE_RUN_TIME_STATS */
00447
00448 #ifndef portCONFIGURE_TIMER_FOR_RUN_TIME_STATS
00449     #define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()
00450 #endif
00451
00452 #ifndef configUSE_MALLOC FAILED_HOOK
00453     #define configUSE_MALLOC FAILED_HOOK 0
00454 #endif
00455
00456 #ifndef portPRIVILEGE_BIT
00457     #define portPRIVILEGE_BIT ( ( unsigned portBASE_TYPE ) 0x00 )
00458 #endif
00459
00460 #ifndef portYIELD_WITHIN_API
00461     #define portYIELD_WITHIN_API portYIELD
00462 #endif
00463
00464 #ifndef pvPortMallocAligned
00465     #define pvPortMallocAligned( x, puxStackBuffer ) ( ( ( puxStackBuffer ) == NULL ) ? ( pvPortMalloc( ( x ) ) ) : ( puxStackBuffer ) )

```

```

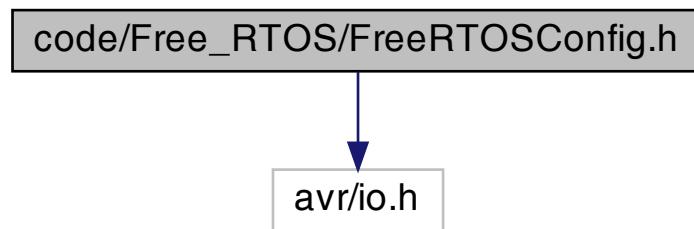
00466 #endif
00467
00468 #ifndef vPortFreeAligned
00469     #define vPortFreeAligned( pvBlockToFree ) vPortFree( pvBlockToFree )
00470 #endif
00471
00472 #endif /* INC_FREERTOS_H */
00473

```

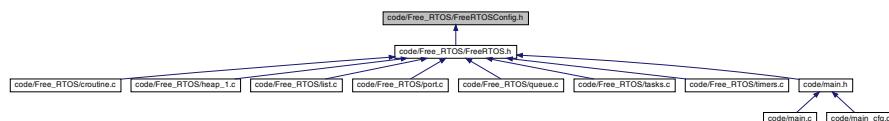
22.7 code/FreeRTOS/FreeRTOSConfig.h File Reference

#include <avr/io.h>

Include dependency graph for FreeRTOSConfig.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define configUSE_PREEMPTION 1
- #define configUSE_IDLE_HOOK 0
- #define configUSE_TICK_HOOK 0
- #define configCPU_CLOCK_HZ ((unsigned long) 8000000)
- #define configTICK_RATE_HZ ((portTickType) 8000)
- #define configMAX_PRIORITIES ((unsigned portBASE_TYPE) 10)
- #define configMINIMAL_STACK_SIZE ((unsigned short) 100)
- #define configTOTAL_HEAP_SIZE ((size_t) (1000))
- #define configMAX_TASK_NAME_LEN (8)
- #define configUSE_TRACE_FACILITY 0
- #define configUSE_16_BIT TICKS 1
- #define configIDLE_SHOULD_YIELD 0
- #define configQUEUE_REGISTRY_SIZE 0
- #define configUSE_COUNTING_SEMAPHORES 1
- #define configUSE_CO_ROUTINES 0

- #define configMAX_CO_ROUTINE_PRIORITIES (2)
- #define INCLUDE_vTaskPrioritySet 0
- #define INCLUDE_uxTaskPriorityGet 0
- #define INCLUDE_vTaskDelete 1
- #define INCLUDE_vTaskCleanUpResources 0
- #define INCLUDE_vTaskSuspend 0
- #define INCLUDE_vTaskDelayUntil 1
- #define INCLUDE_vTaskDelay 1

22.7.1 Macro Definition Documentation

22.7.1.1 configCPU_CLOCK_HZ #define configCPU_CLOCK_HZ ((unsigned long) 8000000)

Definition at line 74 of file [FreeRTOSConfig.h](#).

22.7.1.2 configIDLE_SHOULD_YIELD #define configIDLE_SHOULD_YIELD 0

Definition at line 82 of file [FreeRTOSConfig.h](#).

22.7.1.3 configMAX_CO_ROUTINE_PRIORITIES #define configMAX_CO_ROUTINE_PRIORITIES (2)

Definition at line 88 of file [FreeRTOSConfig.h](#).

22.7.1.4 configMAX_PRIORITIES #define configMAX_PRIORITIES ((unsigned portBASE_TYPE) 10)

Definition at line 76 of file [FreeRTOSConfig.h](#).

22.7.1.5 configMAX_TASK_NAME_LEN #define configMAX_TASK_NAME_LEN (8)

Definition at line 79 of file [FreeRTOSConfig.h](#).

22.7.1.6 configMINIMAL_STACK_SIZE #define configMINIMAL_STACK_SIZE ((unsigned short) 100)

Definition at line 77 of file [FreeRTOSConfig.h](#).

22.7.1.7 configQUEUE_REGISTRY_SIZE #define configQUEUE_REGISTRY_SIZE 0

Definition at line 83 of file [FreeRTOSConfig.h](#).

22.7.1.8 configTICK_RATE_HZ #define configTICK_RATE_HZ ((portTickType) 8000)

Definition at line 75 of file [FreeRTOSConfig.h](#).

22.7.1.9 configTOTAL_HEAP_SIZE #define configTOTAL_HEAP_SIZE ((size_t) (1000))

Definition at line 78 of file [FreeRTOSConfig.h](#).

22.7.1.10 configUSE_16_BIT_TICKS #define configUSE_16_BIT_TICKS 1

Definition at line 81 of file [FreeRTOSConfig.h](#).

22.7.1.11 configUSE_CO_ROUTINES #define configUSE_CO_ROUTINES 0

Definition at line 87 of file [FreeRTOSConfig.h](#).

22.7.1.12 configUSE_COUNTING_SEMAPHORES #define configUSE_COUNTING_SEMAPHORES 1

Definition at line 85 of file [FreeRTOSConfig.h](#).

22.7.1.13 configUSE_IDLE_HOOK #define configUSE_IDLE_HOOK 0

Definition at line 72 of file [FreeRTOSConfig.h](#).

22.7.1.14 configUSE_PREEMPTION #define configUSE_PREEMPTION 1

Definition at line 71 of file [FreeRTOSConfig.h](#).

22.7.1.15 configUSE_TICK_HOOK #define configUSE_TICK_HOOK 0

Definition at line 73 of file [FreeRTOSConfig.h](#).

22.7.1.16 configUSE_TRACE_FACILITY #define configUSE_TRACE_FACILITY 0

Definition at line 80 of file [FreeRTOSConfig.h](#).

22.7.1.17 INCLUDE_uxTaskPriorityGet #define INCLUDE_uxTaskPriorityGet 0

Definition at line 94 of file [FreeRTOSConfig.h](#).

22.7.1.18 INCLUDE_vTaskCleanUpResources #define INCLUDE_vTaskCleanUpResources 0

Definition at line 96 of file [FreeRTOSConfig.h](#).

22.7.1.19 INCLUDE_vTaskDelay #define INCLUDE_vTaskDelay 1

Definition at line 99 of file [FreeRTOSConfig.h](#).

22.7.1.20 INCLUDE_vTaskDelayUntil #define INCLUDE_vTaskDelayUntil 1

Definition at line 98 of file [FreeRTOSConfig.h](#).

22.7.1.21 INCLUDE_vTaskDelete #define INCLUDE_vTaskDelete 1

Definition at line 95 of file [FreeRTOSConfig.h](#).

22.7.1.22 INCLUDE_vTaskPrioritySet #define INCLUDE_vTaskPrioritySet 0

Definition at line 93 of file [FreeRTOSConfig.h](#).

22.7.1.23 INCLUDE_vTaskSuspend #define INCLUDE_vTaskSuspend 0

Definition at line 97 of file [FreeRTOSConfig.h](#).

22.8 FreeRTOSConfig.h

```

00001 /*
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005   ****
00006   *
00007   *   FreeRTOS tutorial books are available in pdf and paperback.
00008   *   Complete, revised, and edited pdf reference manuals are also
00009   *   available.
00010   *
00011   *   Purchasing FreeRTOS documentation will not only help you, by
00012   *   ensuring you get running as quickly as possible and with an
00013   *   in-depth knowledge of how to use FreeRTOS, it will also help
00014   *   the FreeRTOS project to continue with its mission of providing
00015   *   professional grade, cross platform, de facto standard solutions
00016   *   for microcontrollers - completely free of charge!
00017   *
00018   *   >> See http://www.FreeRTOS.org/Documentation for details. <<
00019   *
00020   *   Thank you for using FreeRTOS, and thank you for your support!
00021   *
00022   ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 >>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039 by writing to Richard Barry, contact details for whom are available on the
00040 FreeRTOS WEB site.
00041
00042 1 tab == 4 spaces!
00043
00044 http://www.FreeRTOS.org - Documentation, latest information, license and
00045 contact details.
00046
00047 http://www.SafeRTOS.com - A version that is certified for use in safety
00048 critical systems.
00049
00050 http://www.OpenRTOS.com - Commercial support, development, porting,
00051 licensing and training services.
00052 */
00053
00054 #ifndef FREERTOS_CONFIG_H
00055 #define FREERTOS_CONFIG_H
00056
00057 #include <avr/io.h>
00058
00059 /-----
00060   * Application specific definitions.
00061   *
00062   * These definitions should be adjusted for your particular hardware and
00063   * application requirements.
00064   *
00065   * THESE PARAMETERS ARE DESCRIBED WITHIN THE 'CONFIGURATION' SECTION OF THE
00066   * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org WEB SITE.
00067   *
00068   * See http://www.freertos.org/a00110.html.
00069 -----*/
00070
00071 #define configUSE_PREEMPTION      1
00072 #define configUSE_IDLE_HOOK       0
00073 #define configUSE_TICK_HOOK       0
00074 #define configCPU_CLOCK_HZ        ( ( unsigned long ) 8000000 )
00075 #define configTICK_RATE_HZ        ( ( portTickType ) 8000 )
00076 #define configMAX_PRIORITIES     ( ( unsigned portBASE_TYPE ) 10 )
00077 #define configMINIMAL_STACK_SIZE  ( ( unsigned short ) 100 )

```

```

00078 #define configTOTAL_HEAP_SIZE      ( (size_t) (1000) )
00079 #define configMAX_TASK_NAME_LEN    ( 8 )
00080 #define configUSE_TRACE_FACILITY   0
00081 #define configUSE_16_BIT_TICKS     1
00082 #define configIDLE_SHOULD_YIELD   0
00083 #define configQUEUE_REGISTRY_SIZE  0
00084
00085 #define configUSE_COUNTING_SEMAPHORES 1
00086 /* Co-routine definitions. */
00087 #define configUSE_CO_ROUTINES      0
00088 #define configMAX_CO_ROUTINE_PRIORITIES ( 2 )
00089
00090 /* Set the following definitions to 1 to include the API function, or zero
00091 to exclude the API function. */
00092
00093 #define INCLUDE_vTaskPrioritySet      0
00094 #define INCLUDE uxTaskPriorityGet      0
00095 #define INCLUDE_vTaskDelete         1
00096 #define INCLUDE_vTaskCleanUpResources 0
00097 #define INCLUDE_vTaskSuspend        0
00098 #define INCLUDE_vTaskDelayUntil     1
00099 #define INCLUDE_vTaskDelay        1
00100
00101
00102 #endif /* FREERTOS_CONFIG_H */

```

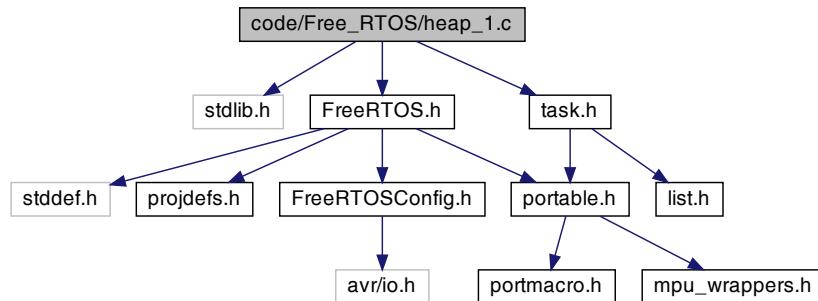
22.9 code/FreeRTOS/heap_1.c File Reference

```

#include <stdlib.h>
#include "FreeRTOS.h"
#include "task.h"

Include dependency graph for heap_1.c:

```



Data Structures

- union **xRTOS_HEAP**

Macros

- #define **MPU_WRAPPERS_INCLUDED_FROM_API_FILE**

Functions

- void * **pvPortMalloc** (size_t xWantedSize)
- void **vPortFree** (void *pv)
- void **vPortInitialiseBlocks** (void)
- size_t **xPortGetFreeHeapSize** (void)

22.9.1 Macro Definition Documentation

22.9.1.1 MPU_WRAPPERS_INCLUDED_FROM_API_FILE #define MPU_WRAPPERS_INCLUDED_FROM_API_←
FILE

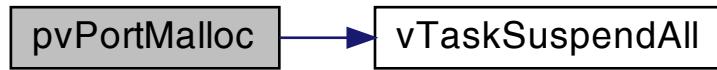
Definition at line 67 of file [heap_1.c](#).

22.9.2 Function Documentation

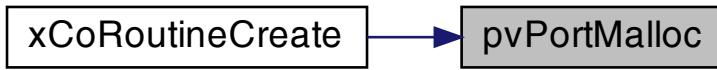
22.9.2.1 pvPortMalloc() void* pvPortMalloc (size_t xWantedSize)

Definition at line 89 of file [heap_1.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.9.2.2 vPortFree() void vPortFree (void * pv)

Definition at line 130 of file [heap_1.c](#).

Here is the caller graph for this function:



22.9.2.3 vPortInitialiseBlocks() void vPortInitialiseBlocks (void)

Definition at line 139 of file [heap_1.c](#).

22.9.2.4 xPortGetFreeHeapSize() size_t xPortGetFreeHeapSize (void)

Definition at line 146 of file [heap_1.c](#).

22.10 heap_1.c

```

00001 /*
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004 ****
00005 *
00006 */
00007 *   FreeRTOS tutorial books are available in pdf and paperback.
00008 *   Complete, revised, and edited pdf reference manuals are also
00009 *   available.
00010 *
00011 *   Purchasing FreeRTOS documentation will not only help you, by
00012 *   ensuring you get running as quickly as possible and with an
00013 *   in-depth knowledge of how to use FreeRTOS, it will also help
00014 *   the FreeRTOS project to continue with its mission of providing
00015 *   professional grade, cross platform, de facto standard solutions
00016 *   for microcontrollers - completely free of charge!
00017 *
00018 *   >> See http://www.FreeRTOS.org/Documentation for details. <<
00019 */
00020 *   Thank you for using FreeRTOS, and thank you for your support!
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 >>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to

```

```

00032     provide the source code for proprietary components outside of the FreeRTOS
00033     kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034     WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035     or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036     more details. You should have received a copy of the GNU General Public
00037     License and the FreeRTOS license exception along with FreeRTOS; if not it
00038     can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039     by writing to Richard Barry, contact details for whom are available on the
00040     FreeRTOS WEB site.
00041
00042     1 tab == 4 spaces!
00043
00044     http://www.FreeRTOS.org - Documentation, latest information, license and
00045     contact details.
00046
00047     http://www.SafeRTOS.com - A version that is certified for use in safety
00048     critical systems.
00049
00050     http://www.OpenRTOS.com - Commercial support, development, porting,
00051     licensing and training services.
00052 */
00053
00054
00055 /*
00056 * The simplest possible implementation of pvPortMalloc(). Note that this
00057 * implementation does NOT allow allocated memory to be freed again.
00058 *
00059 * See heap_2.c and heap_3.c for alternative implementations, and the memory
00060 * management pages of http://www.FreeRTOS.org for more information.
00061 */
00062 #include <stdlib.h>
00063
00064 /* Defining MPU_WRAPPERS_INCLUDED_FROM_API_FILE prevents task.h from redefining
00065 all the API functions to use the MPU wrappers. That should only be done when
00066 task.h is included from an application file. */
00067 #define MPU_WRAPPERS_INCLUDED_FROM_API_FILE
00068
00069 #include "FreeRTOS.h"
00070 #include "task.h"
00071
00072 #undef MPU_WRAPPERS_INCLUDED_FROM_API_FILE
00073
00074 /* Allocate the memory for the heap. The struct is used to force byte
00075 alignment without using any non-portable code. */
00076 static union xRTOS_HEAP
00077 {
00078     #if portBYTE_ALIGNMENT == 8
00079         volatile portDOUBLE dDummy;
00080     #else
00081         volatile unsigned long ulDummy;
00082     #endif
00083     unsigned char ucHeap[ configTOTAL_HEAP_SIZE ];
00084 } xHeap;
00085
00086 static size_t xNextFreeByte = ( size_t ) 0;
00087 /*-----*/
00088
00089 void *pvPortMalloc( size_t xWantedSize )
00090 {
00091     void *pvReturn = NULL;
00092
00093     /* Ensure that blocks are always aligned to the required number of bytes. */
00094     #if portBYTE_ALIGNMENT != 1
00095         if( xWantedSize & portBYTE_ALIGNMENT_MASK )
00096         {
00097             /* Byte alignment required. */
00098             xWantedSize += ( portBYTE_ALIGNMENT - ( xWantedSize & portBYTE_ALIGNMENT_MASK ) );
00099         }
00100     #endif
00101
00102     vTaskSuspendAll();
00103
00104     /* Check there is enough room left for the allocation. */
00105     if( ( ( xNextFreeByte + xWantedSize ) < configTOTAL_HEAP_SIZE ) &&
00106         ( ( xNextFreeByte + xWantedSize ) > xNextFreeByte ) )/* Check for overflow. */
00107     {
00108         /* Return the next free byte then increment the index past this
00109         block. */
00110         pvReturn = &( xHeap.ucHeap[ xNextFreeByte ] );
00111         xNextFreeByte += xWantedSize;
00112     }
00113 }
00114 xTaskResumeAll();
00115
00116 #if( configUSE_MALLOC_FAILED_HOOK == 1 )
00117 {
00118     if( pvReturn == NULL )

```

```

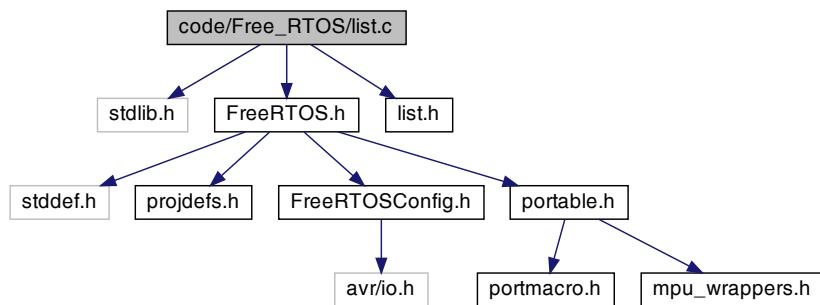
00119      {
00120          extern void vApplicationMallocFailedHook( void );
00121          vApplicationMallocFailedHook();
00122      }
00123  #endif
00124
00125  return pvReturn;
00126 }
00128 /*-----*/
00129
00130 void vPortFree( void *pv )
00131 {
00132     /* Memory cannot be freed using this scheme. See heap_2.c and heap_3.c
00133     for alternative implementations, and the memory management pages of
00134     http://www.FreeRTOS.org for more information. */
00135     ( void ) pv;
00136 }
00137 /*-----*/
00138
00139 void vPortInitialiseBlocks( void )
00140 {
00141     /* Only required when static memory is not cleared. */
00142     xNextFreeByte = ( size_t ) 0;
00143 }
00144 /*-----*/
00145
00146 size_t xPortGetFreeHeapSize( void )
00147 {
00148     return ( configTOTAL_HEAP_SIZE - xNextFreeByte );
00149 }
00150
00151
00152

```

22.11 code/FreeRTOS/list.c File Reference

```
#include <stdlib.h>
#include "FreeRTOS.h"
#include "list.h"

Include dependency graph for list.c:
```



Functions

- void **vListInitialise** (**xList** *pxList)
- void **vListInitialiselist** (**xListItem** *pxListItem)
- void **vListInsertEnd** (**xList** *pxList, **xListItem** *pxNewListItem)
- void **vListInsert** (**xList** *pxList, **xListItem** *pxNewListItem)
- void **vListRemove** (**xListItem** *pxItemToRemove)

22.11.1 Function Documentation

22.11.1.1 vListInitialise() void vListInitialise (
 xList * pxList)

Definition at line 63 of file [list.c](#).

22.11.1.2 vListInitialiseItem() void vListInitialiseItem (
 xListItem * pxItem)

Definition at line 83 of file [list.c](#).

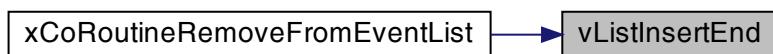
22.11.1.3 vListInsert() void vListInsert (
 xList * pxList,
 xListItem * pxNewListItem)

Definition at line 113 of file [list.c](#).

22.11.1.4 vListInsertEnd() void vListInsertEnd (
 xList * pxList,
 xListItem * pxNewListItem)

Definition at line 90 of file [list.c](#).

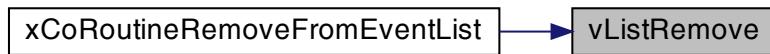
Here is the caller graph for this function:



```
22.11.1.5 vListRemove() void vListRemove (
    xListItem * pxItemToRemove )
```

Definition at line 170 of file [list.c](#).

Here is the caller graph for this function:



22.12 list.c

```

00001 /*
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005 ****
00006 *
00007 *   FreeRTOS tutorial books are available in pdf and paperback.
00008 *   Complete, revised, and edited pdf reference manuals are also
00009 *   available.
00010 *
00011 *   Purchasing FreeRTOS documentation will not only help you, by
00012 *   ensuring you get running as quickly as possible and with an
00013 *   in-depth knowledge of how to use FreeRTOS, it will also help
00014 *   the FreeRTOS project to continue with its mission of providing
00015 *   professional grade, cross platform, de facto standard solutions
00016 *   for microcontrollers - completely free of charge!
00017 *
00018 *   >>> See http://www.FreeRTOS.org/Documentation for details. <<
00019 *
00020 *   Thank you for using FreeRTOS, and thank you for your support!
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 >>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039 by writing to Richard Barry, contact details for whom are available on the
00040 FreeRTOS WEB site.
00041
00042 1 tab == 4 spaces!
00043
00044 http://www.FreeRTOS.org - Documentation, latest information, license and
00045 contact details.
00046
00047 http://www.SafeRTOS.com - A version that is certified for use in safety
00048 critical systems.
00049
00050 http://www.OpenRTOS.com - Commercial support, development, porting,
00051 licensing and training services.
00052 */
00053
00054
00055 #include <stdlib.h>
00056 #include "FreeRTOS.h"
00057 #include "list.h"
00058
00059 /**
00060 * PUBLIC LIST API documented in list.h

```

```

00061  *-----*/
00062
00063 void vListInitialise( xList *pxList )
00064 {
00065     /* The list structure contains a list item which is used to mark the
00066     end of the list. To initialise the list the list end is inserted
00067     as the only list entry. */
00068     pxList->pxIndex = ( xListItem * ) &( pxList->xListEnd );
00069
00070     /* The list end value is the highest possible value in the list to
00071     ensure it remains at the end of the list. */
00072     pxList->xListEnd.xItemValue = portMAX_DELAY;
00073
00074     /* The list end next and previous pointers point to itself so we know
00075     when the list is empty. */
00076     pxList->xListEnd.pxNext = ( xListItem * ) &( pxList->xListEnd );
00077     pxList->xListEnd.pxPrevious = ( xListItem * ) &( pxList->xListEnd );
00078
00079     pxList->uxNumberOfItems = ( unsigned portBASE_TYPE ) 0U;
00080 }
00081 /*-----*/
00082
00083 void vListInitialiseItem( xListItem *pxItem )
00084 {
00085     /* Make sure the list item is not recorded as being on a list. */
00086     pxItem->pvContainer = NULL;
00087 }
00088 /*-----*/
00089
00090 void vListInsertEnd( xList *pxList, xListItem *pxNewListItem )
00091 {
00092     volatile xListItem * pxIndex;
00093
00094     /* Insert a new list item into pxList, but rather than sort the list,
00095     makes the new list item the last item to be removed by a call to
00096     pvListGetOwnerOfNextEntry. This means it has to be the item pointed to by
00097     the pxIndex member. */
00098     pxIndex = pxList->pxIndex;
00099
00100    pxNewListItem->pxNext = pxIndex->pxNext;
00101    pxNewListItem->pxPrevious = pxList->pxIndex;
00102    pxIndex->pxNext->pxPrevious = ( volatile xListItem * ) pxNewListItem;
00103    pxIndex->pxNext = ( volatile xListItem * ) pxNewListItem;
00104    pxList->pxIndex = ( volatile xListItem * ) pxNewListItem;
00105
00106    /* Remember which list the item is in. */
00107    pxNewListItem->pvContainer = ( void * ) pxList;
00108
00109    ( pxList->uxNumberOfItems )++;
00110 }
00111 /*-----*/
00112
00113 void vListInsert( xList *pxList, xListItem *pxNewListItem )
00114 {
00115     volatile xListItem *pxIterator;
00116     portTickType xValueOfInsertion;
00117
00118     /* Insert the new list item into the list, sorted in ulListItem order. */
00119     xValueOfInsertion = pxNewListItem->xItemValue;
00120
00121     /* If the list already contains a list item with the same item value then
00122     the new list item should be placed after it. This ensures that TCB's which
00123     are stored in ready lists (all of which have the same ulListItem value)
00124     get an equal share of the CPU. However, if the xItemValue is the same as
00125     the back marker the iteration loop below will not end. This means we need
00126     to guard against this by checking the value first and modifying the
00127     algorithm slightly if necessary. */
00128     if( xValueOfInsertion == portMAX_DELAY )
00129     {
00130         pxIterator = pxList->xListEnd.pxPrevious;
00131     }
00132     else
00133     {
00134         /* *** NOTE ****
00135         If you find your application is crashing here then likely causes are:
00136             1) Stack overflow -
00137                 see http://www.freertos.org/Stacks-and-stack-overflow-checking.html
00138             2) Incorrect interrupt priority assignment, especially on Cortex-M3
00139                 parts where numerically high priority values denote low actual
00140                 interrupt priorities, which can seem counter intuitive. See
00141                 configMAX_SYSCALL_INTERRUPT_PRIORITY on http://www.freertos.org/a00110.html
00142             3) Calling an API function from within a critical section or when
00143                 the scheduler is suspended.
00144             4) Using a queue or semaphore before it has been initialised or
00145                 before the scheduler has been started (are interrupts firing
00146                 before vTaskStartScheduler() has been called?).
00147                 See http://www.freertos.org/FAQHelp.html for more tips.
```

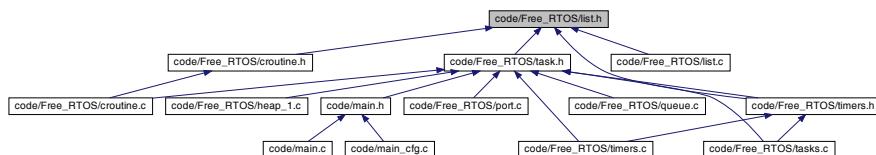
```

00148     ****
00149
00150     for( pxIterator = ( xListItem * ) &( pxList->xListEnd ); pxIterator->pxNext->xItemValue <=
00151         xValueOfInsertion; pxIterator = pxIterator->pxNext )
00152     {
00153         /* There is nothing to do here, we are just iterating to the
00154            wanted insertion position. */
00155     }
00156
00157     pxNewItem->pxNext = pxIterator->pxNext;
00158     pxNewItem->pxPrevious = ( volatile xListItem * ) pxNewItem;
00159     pxNewItem->pxPrevious = pxIterator;
00160     pxIterator->pxNext = ( volatile xListItem * ) pxNewItem;
00161
00162     /* Remember which list the item is in. This allows fast removal of the
00163        item later. */
00164     pxNewItem->pvContainer = ( void * ) pxList;
00165
00166     ( pxList->uxNumberOfItems )++;
00167 }
00168 -----
00169
00170 void vListRemove( xListItem *pxItemToRemove )
00171 {
00172     xList * pxList;
00173
00174     pxItemToRemove->pxNext->pxPrevious = pxItemToRemove->pxPrevious;
00175     pxItemToRemove->pxPrevious->pxNext = pxItemToRemove->pxNext;
00176
00177     /* The list item knows which list it is in. Obtain the list from the list
00178        item. */
00179     pxList = ( xList * ) pxItemToRemove->pvContainer;
00180
00181     /* Make sure the index is left pointing to a valid item. */
00182     if( pxList->pxIndex == pxItemToRemove )
00183     {
00184         pxList->pxIndex = pxItemToRemove->pxPrevious;
00185     }
00186
00187     pxItemToRemove->pvContainer = NULL;
00188     ( pxList->uxNumberOfItems )--;
00189 }
00190 -----
00191

```

22.13 code/FreeRTOS/list.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct `xLIST_ITEM`
- struct `xMINI_LIST_ITEM`
- struct `xLIST`

Macros

- `#define listSET_LIST_ITEM_OWNER(pxListItem, pxOwner) (pxListItem)->pvOwner = (void *) (pxOwner)`

- `#define listSET_LIST_ITEM_VALUE(pxListItem, xValue) (pxListItem)->xItemValue = (xValue)`
- `#define listGET_LIST_ITEM_VALUE(pxListItem) ((pxListItem)->xItemValue)`
- `#define listGET_ITEM_VALUE_OF_HEAD_ENTRY(pxList) ((& (pxList)->xListEnd)->pxNext->xItemValue)`
- `#define listLIST_IS_EMPTY(pxList) ((pxList)->uxNumberOfItems == (unsigned portBASE_TYPE) 0)`
- `#define listCURRENT_LIST_LENGTH(pxList) ((pxList)->uxNumberOfItems)`
- `#define listGET_OWNER_OF_NEXT_ENTRY(pxTCB, pxList)`
- `#define listGET_OWNER_OF_HEAD_ENTRY(pxList) ((& (pxList)->xListEnd)->pxNext->pvOwner)`
- `#define listIS_CONTAINED_WITHIN(pxList, pxListItem) ((pxListItem)->pvContainer == (void *) (pxList))`

Typedefs

- `typedef struct xLIST_ITEM xListItem`
- `typedef struct xMINI_LIST_ITEM xMiniListItem`
- `typedef struct xLIST xList`

Functions

- `void vListInitialise (xList *pxList)`
- `void vListInitialiseItem (xListItem *pxItem)`
- `void vListInsert (xList *pxList, xListItem *pxNewListItem)`
- `void vListInsertEnd (xList *pxList, xListItem *pxNewListItem)`
- `void vListRemove (xListItem *pxItemToRemove)`

22.13.1 Macro Definition Documentation

22.13.1.1 listCURRENT_LIST_LENGTH `#define listCURRENT_LIST_LENGTH(pxList) ((pxList)->uxNumberOfItems)`

Definition at line 169 of file [list.h](#).

22.13.1.2 listGET_ITEM_VALUE_OF_HEAD_ENTRY `#define listGET_ITEM_VALUE_OF_HEAD_ENTRY(pxList) ((& (pxList)->xListEnd)->pxNext->xItemValue)`

Definition at line 155 of file [list.h](#).

22.13.1.3 listGET_LIST_ITEM_VALUE `#define listGET_LIST_ITEM_VALUE(pxListItem) ((pxListItem)->xItemValue)`

Definition at line 146 of file [list.h](#).

```
22.13.1.4 listGET_OWNER_OF_HEAD_ENTRY #define listGET_OWNER_OF_HEAD_ENTRY( pxList ) ( (&( ( pxList )->xListEnd ))->pxNext->pvOwner )
```

Definition at line 220 of file [list.h](#).

```
22.13.1.5 listGET_OWNER_OF_NEXT_ENTRY #define listGET_OWNER_OF_NEXT_ENTRY( pxTCB, pxList )
```

Value:

```
{ xList * const pxConstList = ( pxList );
  /* Increment the index to the next item and return the item, ensuring */
  /* we don't return the marker used at the end of the list. */
  ( pxConstList )->pxIndex = ( pxConstList )->pxIndex->pxNext;
  if( ( pxConstList )->pxIndex == ( xListItem * ) && ( pxConstList )->xListEnd ) {
    ( pxConstList )->pxIndex = ( pxConstList )->pxIndex->pxNext;
  }
  ( pxTCB ) = ( pxConstList )->pxIndex->pvOwner;
}
```

Definition at line 190 of file [list.h](#).

```
22.13.1.6 listIS_CONTAINED_WITHIN #define listIS_CONTAINED_WITHIN(
  pxList,
  pxListItem ) ( ( pxListItem )->pvContainer == ( void * ) ( pxList ) )
```

Definition at line 232 of file [list.h](#).

```
22.13.1.7 listLIST_IS_EMPTY #define listLIST_IS_EMPTY(
  pxList ) ( ( pxList )->uxNumberOfItems == ( unsigned portBASE_TYPE ) 0 )
```

Definition at line 164 of file [list.h](#).

```
22.13.1.8 listSET_LIST_ITEM_OWNER #define listSET_LIST_ITEM_OWNER(
  pxListItem,
  pxOwner ) ( pxListItem )->pvOwner = ( void * ) ( pxOwner )
```

Definition at line 127 of file [list.h](#).

```
22.13.1.9 listSET_LIST_ITEM_VALUE #define listSET_LIST_ITEM_VALUE(
  pxListItem,
  xValue ) ( pxListItem )->xItemValue = ( xValue )
```

Definition at line 136 of file [list.h](#).

22.13.2 Typedef Documentation

22.13.2.1 xList `typedef struct xLIST xList`

22.13.2.2 xListItem `typedef struct xLIST_ITEM xListItem`

Definition at line 1 of file [list.h](#).

22.13.2.3 xMiniListItem `typedef struct xMINI_LIST_ITEM xMiniListItem`

Definition at line 1 of file [list.h](#).

22.13.3 Function Documentation

22.13.3.1 vListInitialise() `void vListInitialise (`
 `xList * pxList)`

Definition at line 63 of file [list.c](#).

22.13.3.2 vListInitialiseItem() `void vListInitialiseItem (`
 `xListItem * pxItem)`

Definition at line 83 of file [list.c](#).

22.13.3.3 vListInsert() `void vListInsert (`
 `xList * pxList,`
 `xListItem * pxNewListItem)`

Definition at line 113 of file [list.c](#).

```
22.13.3.4 vListInsertEnd() void vListInsertEnd (
    xList * pxList,
    xListItem * pxNewListItem )
```

Definition at line 90 of file [list.c](#).

Here is the caller graph for this function:



```
22.13.3.5 vListRemove() void vListRemove (
    xListItem * pxItemToRemove )
```

Definition at line 170 of file [list.c](#).

Here is the caller graph for this function:



22.14 list.h

```
00001 /*
00002     FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004 ****
00005 *
00006 *      FreeRTOS tutorial books are available in pdf and paperback.
00007 *      Complete, revised, and edited pdf reference manuals are also
00008 *      available.
00009 *
00010 *
00011 *      Purchasing FreeRTOS documentation will not only help you, by
00012 *      ensuring you get running as quickly as possible and with an
00013 *      in-depth knowledge of how to use FreeRTOS, it will also help
00014 *      the FreeRTOS project to continue with its mission of providing
00015 *      professional grade, cross platform, de facto standard solutions
00016 *      for microcontrollers - completely free of charge!
00017 *
00018 *      >> See http://www.FreeRTOS.org/Documentation for details. <<
00019 *
00020 *      Thank you for using FreeRTOS, and thank you for your support!
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
```

```

00030     >>NOTE<< The modification to the GPL is included to allow you to
00031     distribute a combined work that includes FreeRTOS without being obliged to
00032     provide the source code for proprietary components outside of the FreeRTOS
00033     kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034     WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035     or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036     more details. You should have received a copy of the GNU General Public
00037     License and the FreeRTOS license exception along with FreeRTOS; if not it
00038     can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039     by writing to Richard Barry, contact details for whom are available on the
00040     FreeRTOS WEB site.
00041
00042     1 tab == 4 spaces!
00043
00044     http://www.FreeRTOS.org - Documentation, latest information, license and
00045     contact details.
00046
00047     http://www.SafeRTOS.com - A version that is certified for use in safety
00048     critical systems.
00049
00050     http://www.OpenRTOS.com - Commercial support, development, porting,
00051     licensing and training services.
00052 */
00053 /*
00054 */
00055 * This is the list implementation used by the scheduler. While it is tailored
00056 * heavily for the schedulers needs, it is also available for use by
00057 * application code.
00058 *
00059 * xLists can only store pointers to xListItems. Each xListItem contains a
00060 * numeric value (xItemValue). Most of the time the lists are sorted in
00061 * descending item value order.
00062 *
00063 * Lists are created already containing one list item. The value of this
00064 * item is the maximum possible that can be stored, it is therefore always at
00065 * the end of the list and acts as a marker. The list member pxHead always
00066 * points to this marker - even though it is at the tail of the list. This
00067 * is because the tail contains a wrap back pointer to the true head of
00068 * the list.
00069 *
00070 * In addition to its value, each list item contains a pointer to the next
00071 * item in the list (pxNext), a pointer to the list it is in (pxContainer)
00072 * and a pointer to back to the object that contains it. These latter two
00073 * pointers are included for efficiency of list manipulation. There is
00074 * effectively a two way link between the object containing the list item and
00075 * the list item itself.
00076 *
00077 *
00078 * \page ListIntroduction List Implementation
00079 * \ingroup FreeRTOSIntro
00080 */
00081
00082
00083 #ifndef LIST_H
00084 #define LIST_H
00085
00086 #ifdef __cplusplus
00087 extern "C" {
00088 #endif
00089 /*
00090 * Definition of the only type of object that a list can contain.
00091 */
00092 struct xLIST_ITEM
00093 {
00094     portTickType xItemValue;           /*< The value being listed. In most cases this is used to
00095                                         sort the list in descending order. */
00096     volatile struct xLIST_ITEM * pxNext;    /*< Pointer to the next xListItem in the list. */
00097     volatile struct xLIST_ITEM * pxPrevious; /*< Pointer to the previous xListItem in the list. */
00098     void * pvOwner;                   /*< Pointer to the object (normally a TCB) that contains
00099                                         the list item. There is therefore a two way link between the object containing the list item and the
00100                                         list item itself. */
00101     void * pvContainer;              /*< Pointer to the list in which this list item is placed
00102                                         (if any). */
00103 };
00104 typedef struct xLIST_ITEM xListItem;    /* For some reason lint wants this as two separate
00105                                         definitions. */
00106
00107 struct xMINI_LIST_ITEM
00108 {
00109     portTickType xItemValue;
00110     volatile struct xLIST_ITEM *pxNext;
00111     volatile struct xLIST_ITEM *pxPrevious;
00112 };
00113 typedef struct xMINI_LIST_ITEM xMiniListItem;
00114
00115 /*
00116 * Definition of the type of queue used by the scheduler.

```

```

00112 */
00113 typedef struct xLIST
00114 {
00115     volatile unsigned portBASE_TYPE uxNumberOfItems;
00116     volatile xListItem * pxIndex;           /*< Used to walk through the list. Points to the last
00117     item returned by a call to pvListGetOwnerOfNextEntry () . */
00118     volatile xMiniListItem xListEnd;        /*< List item that contains the maximum possible item value
00119     meaning it is always at the end of the list and is therefore used as a marker. */
00120 } xList;
00121 */
00122 * Access macro to set the owner of a list item. The owner of a list item
00123 * is the object (usually a TCB) that contains the list item.
00124 *
00125 * \page listSET_LIST_ITEM_OWNER listSET_LIST_ITEM_OWNER
00126 * \ingroup LinkedList
00127 */
00128 #define listSET_LIST_ITEM_OWNER( pxListItem, pxOwner )      ( pxListItem )->pvOwner = ( void * ) ( pxOwner )
00129 */
00130 * Access macro to set the value of the list item. In most cases the value is
00131 * used to sort the list in descending order.
00132 *
00133 * \page listSET_LIST_ITEM_VALUE listSET_LIST_ITEM_VALUE
00134 * \ingroup LinkedList
00135 */
00136 #define listSET_LIST_ITEM_VALUE( pxListItem, xValue )       ( pxListItem )->xItemValue = ( xValue )
00137 */
00138 * Access macro to retrieve the value of the list item. The value can
00139 * represent anything - for example a the priority of a task, or the time at
00140 * which a task should be unblocked.
00141 *
00142 *
00143 * \page listGET_LIST_ITEM_VALUE listGET_LIST_ITEM_VALUE
00144 * \ingroup LinkedList
00145 */
00146 #define listGET_LIST_ITEM_VALUE( pxListItem )             ( ( pxListItem )->xItemValue )
00147 */
00148 * Access macro to retrieve the value of the list item at the head of a given
00149 * list.
00150 *
00151 *
00152 * \page listGET_LIST_ITEM_VALUE listGET_LIST_ITEM_VALUE
00153 * \ingroup LinkedList
00154 */
00155 #define listGET_ITEM_VALUE_OF_HEAD_ENTRY( pxList )         ( ( &( ( pxList )->xListEnd
00156     ) )->pxNext->xItemValue )
00157 */
00158 * Access macro to determine if a list contains any items. The macro will
00159 * only have the value true if the list is empty.
00160 *
00161 * \page listLIST_IS_EMPTY listLIST_IS_EMPTY
00162 * \ingroup LinkedList
00163 */
00164 #define listLIST_IS_EMPTY( pxList )                      ( ( pxList )->uxNumberOfItems == ( unsigned
00165     portBASE_TYPE ) 0 )
00166 */
00167 * Access macro to return the number of items in the list.
00168 */
00169 #define listCURRENT_LIST_LENGTH( pxList )                ( ( pxList )->uxNumberOfItems )
00170 */
00171 * Access function to obtain the owner of the next entry in a list.
00172 *
00173 *
00174 * The list member pxIndex is used to walk through a list. Calling
00175 * listGET_OWNER_OF_NEXT_ENTRY increments pxIndex to the next item in the list
00176 * and returns that entries pxOwner parameter. Using multiple calls to this
00177 * function it is therefore possible to move through every item contained in
00178 * a list.
00179 *
00180 * The pxOwner parameter of a list item is a pointer to the object that owns
00181 * the list item. In the scheduler this is normally a task control block.
00182 * The pxOwner parameter effectively creates a two way link between the list
00183 * item and its owner.
00184 *
00185 * @param pxList The list from which the next item owner is to be returned.
00186 *
00187 * \page listGET_OWNER_OF_NEXT_ENTRY listGET_OWNER_OF_NEXT_ENTRY
00188 * \ingroup LinkedList
00189 */
00190 #define listGET_OWNER_OF_NEXT_ENTRY( pxTCB, pxList )
00191 {
00192     xList * const pxConstList = ( pxList );
00193     /* Increment the index to the next item and return the item, ensuring */

```

```

00194     /* we don't return the marker used at the end of the list. */
00195     ( pxConstList )->pxIndex = ( pxConstList )->pxIndex->pxNext;
00196     if( ( pxConstList )->pxIndex == ( xListItem * ) & ( ( pxConstList )->xListEnd ) )
00197     {
00198         ( pxConstList )->pxIndex = ( pxConstList )->pxIndex->pxNext;
00199     }
00200     ( pTCB ) = ( pxConstList )->pxIndex->pvOwner;
00201 }
00202
00203
00204 /*
00205 * Access function to obtain the owner of the first entry in a list. Lists
00206 * are normally sorted in ascending item value order.
00207 *
00208 * This function returns the pxOwner member of the first item in the list.
00209 * The pxOwner parameter of a list item is a pointer to the object that owns
00210 * the list item. In the scheduler this is normally a task control block.
00211 * The pxOwner parameter effectively creates a two way link between the list
00212 * item and its owner.
00213 *
00214 * @param pxList The list from which the owner of the head item is to be
00215 * returned.
00216 *
00217 * \page listGET_OWNER_OF_HEAD_ENTRY listGET_OWNER_OF_HEAD_ENTRY
00218 * \ingroup LinkedList
00219 */
00220 #define listGET_OWNER_OF_HEAD_ENTRY( pxList ) ( ( ( pxList )->xListEnd )->pxNext->pvOwner )
00221
00222 /*
00223 * Check to see if a list item is within a list. The list item maintains a
00224 * "container" pointer that points to the list it is in. All this macro does
00225 * is check to see if the container and the list match.
00226 *
00227 * @param pxList The list we want to know if the list item is within.
00228 * @param pxListItem The list item we want to know if is in the list.
00229 * @return pdTRUE is the list item is in the list, otherwise pdFALSE.
00230 * pointer against
00231 */
00232 #define listIS_CONTAINED_WITHIN( pxList, pxListItem ) ( ( pxListItem )->pvContainer == ( void * ) ( pxList ) )
00233
00234 /*
00235 * Must be called before a list is used! This initialises all the members
00236 * of the list structure and inserts the xListEnd item into the list as a
00237 * marker to the back of the list.
00238 *
00239 * @param pxList Pointer to the list being initialised.
00240 *
00241 * \page vListInitialise vListInitialise
00242 * \ingroup LinkedList
00243 */
00244 void vListInitialise( xList *pxList );
00245
00246 /*
00247 * Must be called before a list item is used. This sets the list container to
00248 * null so the item does not think that it is already contained in a list.
00249 *
00250 * @param pxItem Pointer to the list item being initialised.
00251 *
00252 * \page vListInitialiseItem vListInitialiseItem
00253 * \ingroup LinkedList
00254 */
00255 void vListInitialiseItem( xListItem *pxItem );
00256
00257 /*
00258 * Insert a list item into a list. The item will be inserted into the list in
00259 * a position determined by its item value (descending item value order).
00260 *
00261 * @param pxList The list into which the item is to be inserted.
00262 *
00263 * @param pxNewItem The item to that is to be placed in the list.
00264 *
00265 * \page vListInsert vListInsert
00266 * \ingroup LinkedList
00267 */
00268 void vListInsert( xList *pxList, xListItem *pxNewItem );
00269
00270 /*
00271 * Insert a list item into a list. The item will be inserted in a position
00272 * such that it will be the last item within the list returned by multiple
00273 * calls to listGET_OWNER_OF_NEXT_ENTRY.
00274 *
00275 * The list member pvIndex is used to walk through a list. Calling
00276 * listGET_OWNER_OF_NEXT_ENTRY increments pvIndex to the next item in the list.
00277 * Placing an item in a list using vListInsertEnd effectively places the item
00278 * in the list position pointed to by pvIndex. This means that every other
00279 * item within the list will be returned by listGET_OWNER_OF_NEXT_ENTRY before

```

```

00280 * the pvIndex parameter again points to the item being inserted.
00281 *
00282 * @param pxList The list into which the item is to be inserted.
00283 *
00284 * @param pxNewListItem The list item to be inserted into the list.
00285 *
00286 * \page vListInsertEnd vListInsertEnd
00287 * \ingroup LinkedList
00288 */
00289 void vListInsertEnd( xList *pxList, xListItem *pxNewListItem );
00290
00291 /*
00292 * Remove an item from a list. The list item has a pointer to the list that
00293 * it is in, so only the list item need be passed into the function.
00294 *
00295 * @param vListRemove The item to be removed. The item will remove itself from
00296 * the list pointed to by its pxContainer parameter.
00297 *
00298 * \page vListRemove vListRemove
00299 * \ingroup LinkedList
00300 */
00301 void vListRemove( xListItem *pxItemToRemove );
00302
00303 #ifdef __cplusplus
00304 }
00305 #endif
00306
00307 #endif
00308

```

22.15 code/FreeRTOS/macros.h File Reference

set and clear macros and get high nibble and low nibble .

Macros

- #define HIGH_NIBBLE(hVar, var) hVar= (var & 0xf0)
- #define LOW_NIBBLE(lVar, var) lVar= (var<<4)
- #define CLR_BIT(var, bit) var &= (~(1<<bit))
- #define SET_BIT(var, bit) var |= (1<<bit)
- #define TOG_BIT(var, bit) var ^= (1<<bit)

22.15.1 Detailed Description

set and clear macros and get high nibble and low nibble .

Author

Islam Mohamed.

Definition in file [macros.h](#).

22.15.2 Macro Definition Documentation

22.15.2.1 CLR_BIT #define CLR_BIT(
 var,
 bit) var &= (~(1<<bit))

Definition at line 15 of file [macros.h](#).

22.15.2.2 HIGH_NIBBLE #define HIGH_NIBBLE(
 hVar,
 var) hVar= (var & 0xf0)

Definition at line 11 of file [macros.h](#).

22.15.2.3 LOW_NIBBLE #define LOW_NIBBLE(
 lVar,
 var) lVar= (var<<4)

Definition at line 13 of file [macros.h](#).

22.15.2.4 SET_BIT #define SET_BIT(
 var,
 bit) var |= (1<<bit)

Definition at line 17 of file [macros.h](#).

22.15.2.5 TOG_BIT #define TOG_BIT(
 var,
 bit) var ^= (1<<bit)

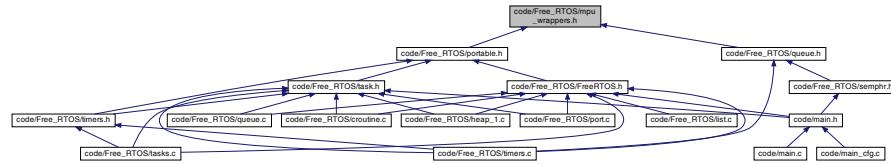
Definition at line 19 of file [macros.h](#).

22.16 macros.h

```
00001
00008 #ifndef _MACROS_H
00009 #define _MACROS_H
00010
00011 #define HIGH_NIBBLE(hVar,var) hVar= (var & 0xf0) // hVar will hold value of high nibble of var
00012
00013 #define LOW_NIBBLE(lVar,var) lVar= (var<<4) // lVar will hold the value of low nibble of var
00014
00015 #define CLR_BIT(var, bit) var &= (~(1<<bit))
00016
00017 #define SET_BIT(var, bit) var |= (1<<bit)
00018
00019 #define TOG_BIT(var, bit) var ^= (1<<bit)
00020
00021 #endif
00022
```

22.17 code/FreeRTOS/mpu_wrappers.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- `#define PRIVILEGED_FUNCTION`
- `#define PRIVILEGED_DATA`
- `#define portUSING_MPUSWRAPPERS 0`

22.17.1 Macro Definition Documentation

22.17.1.1 portUSING_MPUSWRAPPERS `#define portUSING_MPUSWRAPPERS 0`

Definition at line 129 of file [mpu_wrappers.h](#).

22.17.1.2 PRIVILEGED_DATA `#define PRIVILEGED_DATA`

Definition at line 128 of file [mpu_wrappers.h](#).

22.17.1.3 PRIVILEGED_FUNCTION `#define PRIVILEGED_FUNCTION`

Definition at line 127 of file [mpu_wrappers.h](#).

22.18 mpu_wrappers.h

```

00001 /*
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005 ****
00006 *
00007 *   FreeRTOS tutorial books are available in pdf and paperback.
00008 *   Complete, revised, and edited pdf reference manuals are also
00009 *   available.
00010 *
00011 *   Purchasing FreeRTOS documentation will not only help you, by
00012 *   ensuring you get running as quickly as possible and with an
00013 *   in-depth knowledge of how to use FreeRTOS, it will also help
00014 *   the FreeRTOS project to continue with its mission of providing
00015 *   professional grade, cross platform, de facto standard solutions
00016 *   for microcontrollers - completely free of charge!
00017 *
00018 *   >> See http://www.FreeRTOS.org/Documentation for details. <<
00019 *
00020 *   Thank you for using FreeRTOS, and thank you for your support!
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 >>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039 by writing to Richard Barry, contact details for whom are available on the
00040 FreeRTOS WEB site.
00041
00042 1 tab == 4 spaces!
00043
00044 http://www.FreeRTOS.org - Documentation, latest information, license and
00045 contact details.
00046
00047 http://www.SafeRTOS.com - A version that is certified for use in safety
00048 critical systems.
00049
00050 http://www.OpenRTOS.com - Commercial support, development, porting,
00051 licensing and training services.
00052 */
00053
00054 #ifndef MPU_WRAPPERS_H
00055 #define MPU_WRAPPERS_H
00056
00057 /* This file redefines API functions to be called through a wrapper macro, but
00058 only for ports that are using the MPU. */
00059 #ifdef portUSING_MPUSWRAPPERS
00060
00061 /* MPU_WRAPPERS_INCLUDED_FROM_API_FILE will be defined when this file is
00062 included from queue.c or task.c to prevent it from having an effect within
00063 those files. */
00064 #ifndef MPU_WRAPPERS_INCLUDED_FROM_API_FILE
00065
00066     #define xTaskGenericCreate           MPU_xTaskGenericCreate
00067     #define vTaskAllocateMPURegions      MPU_vTaskAllocateMPURegions
00068     #define vTaskDelete                  MPU_vTaskDelete
00069     #define vTaskDelayUntil              MPU_vTaskDelayUntil
00070     #define vTaskDelay                  MPU_vTaskDelay
00071     #define uxTaskPriorityGet            MPU_uxTaskPriorityGet
00072     #define vTaskPrioritySet              MPU_vTaskPrioritySet
00073     #define vTaskSuspend                 MPU_vTaskSuspend
00074     #define xTaskIsTaskSuspended         MPU_xTaskIsTaskSuspended
00075     #define vTaskResume                  MPU_vTaskResume
00076     #define vTaskSuspendAll              MPU_vTaskSuspendAll
00077     #define xTaskResumeAll              MPU_xTaskResumeAll
00078     #define xTaskGetTickCount             MPU_xTaskGetTickCount
00079     #define uxTaskGetNumberOfTasks        MPU_uxTaskGetNumberOfTasks
00080     #define vTaskList                   MPU_vTaskList
00081     #define vTaskGetRunTimeStats          MPU_vTaskGetRunTimeStats
00082     #define vTaskStartTrace              MPU_vTaskStartTrace
00083     #define ulTaskEndTrace              MPU_ulTaskEndTrace
00084     #define vTaskSetApplicationTaskTag    MPU_vTaskSetApplicationTaskTag
00085     #define xTaskGetApplicationTaskTag    MPU_xTaskGetApplicationTaskTag

```

```

00086     #define xTaskCallApplicationTaskHook      MPU_xTaskCallApplicationTaskHook
00087     #define uxTaskGetStackHighWaterMark      MPU_uxTaskGetStackHighWaterMark
00088     #define xTaskGetCurrentTaskHandle       MPU_xTaskGetCurrentTaskHandle
00089     #define xTaskGetSchedulerState          MPU_xTaskGetSchedulerState
00090
00091     #define xQueueCreate                  MPU_xQueueCreate
00092     #define xQueueCreateMutex             MPU_xQueueCreateMutex
00093     #define xQueueGiveMutexRecursive      MPU_xQueueGiveMutexRecursive
00094     #define xQueueTakeMutexRecursive      MPU_xQueueTakeMutexRecursive
00095     #define xQueueCreateCountingSemaphore  MPU_xQueueCreateCountingSemaphore
00096     #define xQueueGenericSend            MPU_xQueueGenericSend
00097     #define xQueueAltGenericSend         MPU_xQueueAltGenericSend
00098     #define xQueueAltGenericReceive      MPU_xQueueAltGenericReceive
00099     #define xQueueGenericReceive         MPU_xQueueGenericReceive
00100     #define uxQueueMessagesWaiting       MPU_uxQueueMessagesWaiting
00101     #define vQueueDelete                 MPU_vQueueDelete
00102
00103     #define pvPortMalloc                MPU_pvPortMalloc
00104     #define vPortFree                   MPU_vPortFree
00105     #define xPortGetFreeHeapSize        MPU_xPortGetFreeHeapSize
00106     #define vPortInitialiseBlocks       MPU_vPortInitialiseBlocks
00107
00108     #if configQUEUE_REGISTRY_SIZE > 0
00109         #define vQueueAddToRegistry        MPU_vQueueAddToRegistry
00110         #define vQueueUnregisterQueue      MPU_vQueueUnregisterQueue
00111     #endif
00112
00113     /* Remove the privileged function macro. */
00114     #define PRIVILEGED_FUNCTION
00115
00116     #else /* MPU_WRAPPERS_INCLUDED_FROM_API_FILE */
00117
00118     /* Ensure API functions go in the privileged execution section. */
00119     #define PRIVILEGED_FUNCTION __attribute__((section("privileged_functions")))
00120     #define PRIVILEGED_DATA __attribute__((section("privileged_data")))
00121     // #define PRIVILEGED_DATA
00122
00123 #endif /* MPU_WRAPPERS_INCLUDED_FROM_API_FILE */
00124
00125 #else /* portUSING_MPU_WRAPPERS */
00126
00127     #define PRIVILEGED_FUNCTION
00128     #define PRIVILEGED_DATA
00129     #define portUSING_MPU_WRAPPERS 0
00130
00131 #endif /* portUSING_MPU_WRAPPERS */
00132
00133
00134 #endif /* MPU_WRAPPERS_H */
00135

```

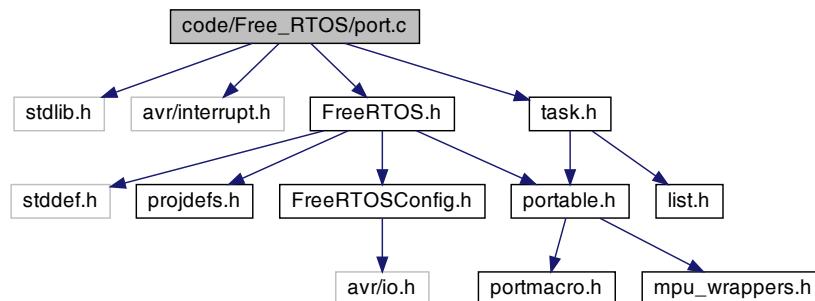
22.19 code/FreeRTOS/port.c File Reference

```

#include <stdlib.h>
#include <avr/interrupt.h>
#include "FreeRTOS.h"
#include "task.h"

Include dependency graph for port.c:

```



Macros

- #define portFLAGS_INT_ENABLED ((portSTACK_TYPE) 0x80)
- #define portCLEAR_COUNTER_ON_MATCH ((unsigned char) 0x08)
- #define portPRESCALE_64 ((unsigned char) 0x03)
- #define portCLOCK_PRESCALER ((unsigned long) 64)
- #define portCOMPARE_MATCH_A_INTERRUPT_ENABLE ((unsigned char) 0x10)
- #define portSAVE_CONTEXT()
- #define portRESTORE_CONTEXT()

TypeDefs

- typedef void tskTCB

Functions

- portSTACK_TYPE * pxPortInitialiseStack (portSTACK_TYPE *pxTopOfStack, pdTASK_CODE pxCode, void *pvParameters)
- portBASE_TYPE xPortStartScheduler (void)
- void vPortEndScheduler (void)
- void vPortYield (void)
- void vPortYieldFromTick (void)
- void SIG_OUTPUT_COMPARE1A (void)

Variables

- volatile tskTCB *volatile pxCurrentTCB

22.19.1 Macro Definition Documentation

22.19.1.1 portCLEAR_COUNTER_ON_MATCH #define portCLEAR_COUNTER_ON_MATCH ((unsigned char) 0x08)

Definition at line 77 of file [port.c](#).

22.19.1.2 portCLOCK_PRESCALER #define portCLOCK_PRESCALER ((unsigned long) 64)

Definition at line 79 of file [port.c](#).

22.19.1.3 portCOMPARE_MATCH_A_INTERRUPT_ENABLE #define portCOMPARE_MATCH_A_INTERRUPT_ENABLE ((unsigned char) 0x10)

Definition at line 80 of file [port.c](#).

22.19.1.4 portFLAGS_INT_ENABLED #define portFLAGS_INT_ENABLED ((portSTACK_TYPE) 0x80)

Definition at line 74 of file [port.c](#).

22.19.1.5 portPRESCALE_64 #define portPRESCALE_64 ((unsigned char) 0x03)

Definition at line 78 of file [port.c](#).

22.19.1.6 portRESTORE_CONTEXT #define portRESTORE_CONTEXT()

Definition at line 157 of file [port.c](#).

22.19.1.7 portSAVE_CONTEXT #define portSAVE_CONTEXT()

Definition at line 107 of file [port.c](#).

22.19.2 Typedef Documentation

22.19.2.1 tskTCB typedef void tskTCB

Definition at line 86 of file [port.c](#).

22.19.3 Function Documentation

22.19.3.1 pxPortInitialiseStack() portSTACK_TYPE* pxPortInitialiseStack (portSTACK_TYPE * pxTopOfStack,
pdTASK_CODE pxCode,
void * pvParameters)

Definition at line 211 of file [port.c](#).

22.19.3.2 SIG_OUTPUT_COMPARE1A() void SIG_OUTPUT_COMPARE1A (void)

Definition at line 429 of file [port.c](#).

22.19.3.3 vPortEndScheduler() void vPortEndScheduler (void)

Definition at line 343 of file [port.c](#).

22.19.3.4 vPortYield() void vPortYield (void)

Definition at line 354 of file [port.c](#).

22.19.3.5 vPortYieldFromTick() void vPortYieldFromTick (void)

Definition at line 371 of file [port.c](#).

22.19.3.6 xPortStartScheduler() portBASE_TYPE xPortStartScheduler (void)

Definition at line 326 of file [port.c](#).

22.19.4 Variable Documentation

22.19.4.1 pxCurrentTCB volatile tskTCB* volatile pxCurrentTCB [extern]

Definition at line 130 of file [tasks.c](#).

22.20 port.c

```
00001 /*  
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.  
00003  
00004  
00005   *****  
00006   *  
00007   *   FreeRTOS tutorial books are available in pdf and paperback.  
00008   *   Complete, revised, and edited pdf reference manuals are also  
00009   *   available.  
00010   *  
00011   *   Purchasing FreeRTOS documentation will not only help you, by  
00012   *   ensuring you get running as quickly as possible and with an  
00013   *   in-depth knowledge of how to use FreeRTOS, it will also help  
00014   *   the FreeRTOS project to continue with its mission of providing  
00015   *   professional grade, cross platform, de facto standard solutions  
00016   *   for microcontrollers - completely free of charge!  
00017   *  
00018   *   >> See http://www.FreeRTOS.org/Documentation for details. <<  
00019   *  
00020   *   Thank you for using FreeRTOS, and thank you for your support!  
00021   *  
00022   *****  
00023
```

```

00024
00025     This file is part of the FreeRTOS distribution.
00026
00027     FreeRTOS is free software; you can redistribute it and/or modify it under
00028     the terms of the GNU General Public License (version 2) as published by the
00029     Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030     >>NOTE<< The modification to the GPL is included to allow you to
00031     distribute a combined work that includes FreeRTOS without being obliged to
00032     provide the source code for proprietary components outside of the FreeRTOS
00033     kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034     WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035     or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036     more details. You should have received a copy of the GNU General Public
00037     License and the FreeRTOS license exception along with FreeRTOS; if not it
00038     can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039     by writing to Richard Barry, contact details for whom are available on the
00040     FreeRTOS WEB site.
00041
00042     1 tab == 4 spaces!
00043
00044     http://www.FreeRTOS.org - Documentation, latest information, license and
00045     contact details.
00046
00047     http://www.SafeRTOS.com - A version that is certified for use in safety
00048     critical systems.
00049
00050     http://www.OpenRTOS.com - Commercial support, development, porting,
00051     licensing and training services.
00052 */
00053 /*
00054 */
00055
00056 Changes from V2.6.0
00057
00058     + AVR port - Replaced the inb() and outb() functions with direct memory
00059     access. This allows the port to be built with the 20050414 build of
00060     WinAVR.
00061 */
00062
00063 #include <stdlib.h>
00064 #include <avr/interrupt.h>
00065
00066 #include "FreeRTOS.h"
00067 #include "task.h"
00068
00069 /*****
00070     * Implementation of functions defined in portable.h for the AVR port.
00071     *****/
00072
00073 /* Start tasks with interrupts enables. */
00074 #define portFLAGS_INT_ENABLED           ( ( portSTACK_TYPE ) 0x80 )
00075
00076 /* Hardware constants for timer 1. */
00077 #define portCLEAR_COUNTER_ON_MATCH      ( ( unsigned char ) 0x08 )
00078 #define portPRESCALE_64                 ( ( unsigned char ) 0x03 )
00079 #define portCLOCK_PRESCALER            ( ( unsigned long ) 64 )
00080 #define portCOMPARE_MATCH_A_INTERRUPT_ENABLE ( ( unsigned char ) 0x10 )
00081
00082 /*****
00083
00084 /* We require the address of the pxCurrentTCB variable, but don't want to know
00085 any details of its type. */
00086 typedef void tskTCB;
00087 extern volatile tskTCB * volatile pxCurrentTCB;
00088
00089 /*****
00090
00091 /*
00092     * Macro to save all the general purpose registers, the save the stack pointer
00093     * into the TCB.
00094     *
00095     * The first thing we do is save the flags then disable interrupts. This is to
00096     * guard our stack against having a context switch interrupt after we have already
00097     * pushed the registers onto the stack - causing the 32 registers to be on the
00098     * stack twice.
00099     *
00100    * r1 is set to zero as the compiler expects it to be thus, however some
00101    * of the math routines make use of R1.
00102    *
00103    * The interrupts will have been disabled during the call to portSAVE_CONTEXT()
00104    * so we need not worry about reading/writing to the stack pointer.
00105    */
00106
00107 #define portSAVE_CONTEXT()
00108     asm volatile ( "push    r0          \n\t"
00109                  "in     r0, __SREG__   \n\t"
00110                  "cli                           \n\t" );

```

```

00111      "push    r0          \n\t" \
00112      "push    r1          \n\t" \
00113      "clr     r1          \n\t" \
00114      "push    r2          \n\t" \
00115      "push    r3          \n\t" \
00116      "push    r4          \n\t" \
00117      "push    r5          \n\t" \
00118      "push    r6          \n\t" \
00119      "push    r7          \n\t" \
00120      "push    r8          \n\t" \
00121      "push    r9          \n\t" \
00122      "push    r10         \n\t" \
00123      "push    r11         \n\t" \
00124      "push    r12         \n\t" \
00125      "push    r13         \n\t" \
00126      "push    r14         \n\t" \
00127      "push    r15         \n\t" \
00128      "push    r16         \n\t" \
00129      "push    r17         \n\t" \
00130      "push    r18         \n\t" \
00131      "push    r19         \n\t" \
00132      "push    r20         \n\t" \
00133      "push    r21         \n\t" \
00134      "push    r22         \n\t" \
00135      "push    r23         \n\t" \
00136      "push    r24         \n\t" \
00137      "push    r25         \n\t" \
00138      "push    r26         \n\t" \
00139      "push    r27         \n\t" \
00140      "push    r28         \n\t" \
00141      "push    r29         \n\t" \
00142      "push    r30         \n\t" \
00143      "push    r31         \n\t" \
00144      "lds     r26, pxCurrentTCB \n\t" \
00145      "lds     r27, pxCurrentTCB + 1 \n\t" \
00146      "in      r0, 0x3d        \n\t" \
00147      "st      x+, r0          \n\t" \
00148      "in      r0, 0x3e        \n\t" \
00149      "st      x+, r0          \n\t" \
00150      ); \
00151 \
00152 /* 
00153 * Opposite to portSAVE_CONTEXT(). Interrupts will have been disabled during
00154 * the context save so we can write to the stack pointer.
00155 */
00156
00157 #define portRESTORE_CONTEXT()
00158     asm volatile ( "lds    r26, pxCurrentTCB    \n\t" \
00159                 "lds    r27, pxCurrentTCB + 1 \n\t" \
00160                 "ld     r28, x+           \n\t" \
00161                 "out   __SP_L__, r28       \n\t" \
00162                 "ld     r29, x+           \n\t" \
00163                 "out   __SP_H__, r29       \n\t" \
00164                 "pop   r31             \n\t" \
00165                 "pop   r30             \n\t" \
00166                 "pop   r29             \n\t" \
00167                 "pop   r28             \n\t" \
00168                 "pop   r27             \n\t" \
00169                 "pop   r26             \n\t" \
00170                 "pop   r25             \n\t" \
00171                 "pop   r24             \n\t" \
00172                 "pop   r23             \n\t" \
00173                 "pop   r22             \n\t" \
00174                 "pop   r21             \n\t" \
00175                 "pop   r20             \n\t" \
00176                 "pop   r19             \n\t" \
00177                 "pop   r18             \n\t" \
00178                 "pop   r17             \n\t" \
00179                 "pop   r16             \n\t" \
00180                 "pop   r15             \n\t" \
00181                 "pop   r14             \n\t" \
00182                 "pop   r13             \n\t" \
00183                 "pop   r12             \n\t" \
00184                 "pop   r11             \n\t" \
00185                 "pop   r10             \n\t" \
00186                 "pop   r9              \n\t" \
00187                 "pop   r8              \n\t" \
00188                 "pop   r7              \n\t" \
00189                 "pop   r6              \n\t" \
00190                 "pop   r5              \n\t" \
00191                 "pop   r4              \n\t" \
00192                 "pop   r3              \n\t" \
00193                 "pop   r2              \n\t" \
00194                 "pop   r1              \n\t" \
00195                 "pop   r0              \n\t" \
00196                 "out   __SREG__, r0       \n\t" \
00197                 "pop   r0              \n\t" \

```

```

00198         );
00199
00200 /*-----*/
00201
00202 /*
00203  * Perform hardware setup to enable ticks from timer 1, compare match A.
00204 */
00205 static void prvSetupTimerInterrupt( void );
00206 /*-----*/
00207
00208 /*
00209  * See header file for description.
00210 */
00211 portSTACK_TYPE *pxPortInitialiseStack( portSTACK_TYPE *pxTopOfStack, pdTASK_CODE pxCode, void
00212   *pvParameters )
00213 {
00214   unsigned short usAddress;
00215
00216   /* Place a few bytes of known values on the bottom of the stack.
00217    * This is just useful for debugging. */
00218
00219   *pxTopOfStack = 0x11;
00220   pxTopOfStack--;
00221   *pxTopOfStack = 0x22;
00222   pxTopOfStack--;
00223   *pxTopOfStack = 0x33;
00224   pxTopOfStack--;
00225
00226   /* Simulate how the stack would look after a call to vPortYield() generated by
00227    * the compiler. */
00228
00229   /*lint -e950 -e611 -e923 Lint doesn't like this much - but nothing I can do about it. */
00230
00231   /* The start of the task code will be popped off the stack last, so place
00232    * it on first. */
00233   usAddress = ( unsigned short ) pxCode;
00234   *pxTopOfStack = ( portSTACK_TYPE ) ( usAddress & ( unsigned short ) 0x00ff );
00235   pxTopOfStack--;
00236
00237   usAddress >= 8;
00238   *pxTopOfStack = ( portSTACK_TYPE ) ( usAddress & ( unsigned short ) 0x00ff );
00239   pxTopOfStack--;
00240
00241   /* Next simulate the stack as if after a call to portSAVE_CONTEXT().
00242    * portSAVE_CONTEXT places the flags on the stack immediately after r0
00243    * to ensure the interrupts get disabled as soon as possible, and so ensuring
00244    * the stack use is minimal should a context switch interrupt occur. */
00245   *pxTopOfStack = ( portSTACK_TYPE ) 0x00; /* R0 */
00246   pxTopOfStack--;
00247   *pxTopOfStack = portFLAGS_INT_ENABLED;
00248   pxTopOfStack--;
00249
00250   /* Now the remaining registers.  The compiler expects R1 to be 0. */
00251   *pxTopOfStack = ( portSTACK_TYPE ) 0x00; /* R1 */
00252   pxTopOfStack--;
00253   *pxTopOfStack = ( portSTACK_TYPE ) 0x02; /* R2 */
00254   pxTopOfStack--;
00255   *pxTopOfStack = ( portSTACK_TYPE ) 0x03; /* R3 */
00256   pxTopOfStack--;
00257   *pxTopOfStack = ( portSTACK_TYPE ) 0x04; /* R4 */
00258   pxTopOfStack--;
00259   *pxTopOfStack = ( portSTACK_TYPE ) 0x05; /* R5 */
00260   pxTopOfStack--;
00261   *pxTopOfStack = ( portSTACK_TYPE ) 0x06; /* R6 */
00262   pxTopOfStack--;
00263   *pxTopOfStack = ( portSTACK_TYPE ) 0x07; /* R7 */
00264   pxTopOfStack--;
00265   *pxTopOfStack = ( portSTACK_TYPE ) 0x08; /* R8 */
00266   pxTopOfStack--;
00267   *pxTopOfStack = ( portSTACK_TYPE ) 0x09; /* R9 */
00268   pxTopOfStack--;
00269   *pxTopOfStack = ( portSTACK_TYPE ) 0x10; /* R10 */
00270   pxTopOfStack--;
00271   *pxTopOfStack = ( portSTACK_TYPE ) 0x11; /* R11 */
00272   pxTopOfStack--;
00273   *pxTopOfStack = ( portSTACK_TYPE ) 0x12; /* R12 */
00274   pxTopOfStack--;
00275   *pxTopOfStack = ( portSTACK_TYPE ) 0x13; /* R13 */
00276   pxTopOfStack--;
00277   *pxTopOfStack = ( portSTACK_TYPE ) 0x14; /* R14 */
00278   pxTopOfStack--;
00279   *pxTopOfStack = ( portSTACK_TYPE ) 0x15; /* R15 */
00280   pxTopOfStack--;
00281   *pxTopOfStack = ( portSTACK_TYPE ) 0x16; /* R16 */
00282   pxTopOfStack--;
00283   *pxTopOfStack = ( portSTACK_TYPE ) 0x17; /* R17 */

```

```

00284     pxTopOfStack--;
00285     *pxTopOfStack = ( portSTACK_TYPE ) 0x18; /* R18 */
00286     pxTopOfStack--;
00287     *pxTopOfStack = ( portSTACK_TYPE ) 0x19; /* R19 */
00288     pxTopOfStack--;
00289     *pxTopOfStack = ( portSTACK_TYPE ) 0x20; /* R20 */
00290     pxTopOfStack--;
00291     *pxTopOfStack = ( portSTACK_TYPE ) 0x21; /* R21 */
00292     pxTopOfStack--;
00293     *pxTopOfStack = ( portSTACK_TYPE ) 0x22; /* R22 */
00294     pxTopOfStack--;
00295     *pxTopOfStack = ( portSTACK_TYPE ) 0x23; /* R23 */
00296     pxTopOfStack--;
00297
00298     /* Place the parameter on the stack in the expected location. */
00299     usAddress = ( unsigned short ) pvParameters;
00300     *pxTopOfStack = ( portSTACK_TYPE ) ( usAddress & ( unsigned short ) 0x00ff );
00301     pxTopOfStack--;
00302
00303     usAddress >= 8;
00304     *pxTopOfStack = ( portSTACK_TYPE ) ( usAddress & ( unsigned short ) 0x00ff );
00305     pxTopOfStack--;
00306
00307     *pxTopOfStack = ( portSTACK_TYPE ) 0x26; /* R26 X */
00308     pxTopOfStack--;
00309     *pxTopOfStack = ( portSTACK_TYPE ) 0x27; /* R27 */
00310     pxTopOfStack--;
00311     *pxTopOfStack = ( portSTACK_TYPE ) 0x28; /* R28 Y */
00312     pxTopOfStack--;
00313     *pxTopOfStack = ( portSTACK_TYPE ) 0x29; /* R29 */
00314     pxTopOfStack--;
00315     *pxTopOfStack = ( portSTACK_TYPE ) 0x30; /* R30 Z */
00316     pxTopOfStack--;
00317     *pxTopOfStack = ( portSTACK_TYPE ) 0x31; /* R31 */
00318     pxTopOfStack--;
00319
00320     /*lint +e950 +e611 +e923 */
00321
00322     return pxTopOfStack;
00323 }
00324 /*-----*/
00325
00326 portBASE_TYPE xPortStartScheduler( void )
00327 {
00328     /* Setup the hardware to generate the tick. */
00329     prvSetupTimerInterrupt();
00330
00331     /* Restore the context of the first task that is going to run. */
00332     portRESTORE_CONTEXT();
00333
00334     /* Simulate a function call end as generated by the compiler. We will now
00335      jump to the start of the task the context of which we have just restored. */
00336     asm volatile ( "ret" );
00337
00338     /* Should not get here. */
00339     return pdTRUE;
00340 }
00341 /*-----*/
00342
00343 void vPortEndScheduler( void )
00344 {
00345     /* It is unlikely that the AVR port will get stopped. If required simply
00346      disable the tick interrupt here. */
00347 }
00348 /*-----*/
00349
00350 /*
00351  * Manual context switch. The first thing we do is save the registers so we
00352  * can use a naked attribute.
00353 */
00354 void vPortYield( void ) __attribute__ ( ( naked ) );
00355 void vPortYield( void )
00356 {
00357     portSAVE_CONTEXT();
00358     vTaskSwitchContext();
00359     portRESTORE_CONTEXT();
00360
00361     asm volatile ( "ret" );
00362 }
00363 /*-----*/
00364
00365 /*
00366  * Context switch function used by the tick. This must be identical to
00367  * vPortYield() from the call to vTaskSwitchContext() onwards. The only
00368  * difference from vPortYield() is the tick count is incremented as the
00369  * call comes from the tick ISR.
00370 */

```

```

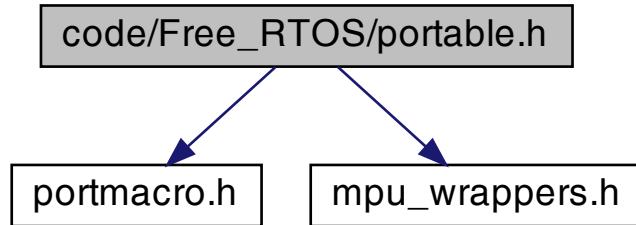
00371 void vPortYieldFromTick( void ) __attribute__ ( ( naked ) );
00372 void vPortYieldFromTick( void )
00373 {
00374     portSAVE_CONTEXT();
00375     vTaskIncrementTick();
00376     vTaskSwitchContext();
00377     portRESTORE_CONTEXT();
00378
00379     asm volatile ( "ret" );
00380 }
00381 /*-----*/
00382
00383 /*
00384  * Setup timer 1 compare match A to generate a tick interrupt.
00385 */
00386 static void prvSetupTimerInterrupt( void )
00387 {
00388     unsigned long ulCompareMatch;
00389     unsigned char ucHighByte, ucLowByte;
00390
00391     /* Using 16bit timer 1 to generate the tick.  Correct fuses must be
00392      selected for the configCPU_CLOCK_HZ clock. */
00393
00394     ulCompareMatch = configCPU_CLOCK_HZ / configTICK_RATE_HZ;
00395
00396     /* We only have 16 bits so have to scale to get our required tick rate. */
00397     ulCompareMatch /= portCLOCK_PRESCALER;
00398
00399     /* Adjust for correct value. */
00400     ulCompareMatch -= ( unsigned long ) 1;
00401
00402     /* Setup compare match value for compare match A.  Interrupts are disabled
00403      before this is called so we need not worry here. */
00404     ucLowByte = ( unsigned char ) ( ulCompareMatch & ( unsigned long ) 0xff );
00405     ulCompareMatch >>= 8;
00406     ucHighByte = ( unsigned char ) ( ulCompareMatch & ( unsigned long ) 0xff );
00407     OCR1AH = ucHighByte;
00408     OCR1AL = ucLowByte;
00409
00410     /* Setup clock source and compare match behaviour. */
00411     ucLowByte = portCLEAR_COUNTER_ON_MATCH | portPRESCALE_64;
00412     TCCR1B = ucLowByte;
00413
00414     /* Enable the interrupt - this is okay as interrupt are currently globally
00415      disabled. */
00416     ucLowByte = TIMSK;
00417     ucLowByte |= portCOMPARE_MATCH_A_INTERRUPT_ENABLE;
00418     TIMSK = ucLowByte;
00419 }
00420 /*-----*/
00421
00422 #if configUSE_PREEMPTION == 1
00423
00424     /*
00425      * Tick ISR for preemptive scheduler.  We can use a naked attribute as
00426      * the context is saved at the start of vPortYieldFromTick().  The tick
00427      * count is incremented after the context is saved.
00428      */
00429     void SIG_OUTPUT_COMPARE1A( void ) __attribute__ ( ( signal, naked ) );
00430     void SIG_OUTPUT_COMPARE1A( void )
00431     {
00432         vPortYieldFromTick();
00433         asm volatile ( "reti" );
00434     }
00435 #else
00436
00437     /*
00438      * Tick ISR for the cooperative scheduler.  All this does is increment the
00439      * tick count.  We don't need to switch context, this can only be done by
00440      * manual calls to taskYIELD();
00441      */
00442     void SIG_OUTPUT_COMPARE1A( void ) __attribute__ ( ( signal ) );
00443     void SIG_OUTPUT_COMPARE1A( void )
00444     {
00445         vTaskIncrementTick();
00446     }
00447 #endif
00448
00449
00450

```

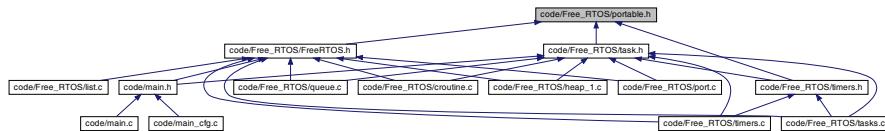
22.21 code/FreeRTOS/portable.h File Reference

```
#include "portmacro.h"
```

```
#include "mpu_wrappers.h"
Include dependency graph for portable.h:
```



This graph shows which files directly or indirectly include this file:



Macros

- `#define portBYTE_ALIGNMENT_MASK (0x0000)`
- `#define portNUM_CONFIGURABLE_REGIONS 1`

Functions

- `portSTACK_TYPE * pxPortInitialiseStack (portSTACK_TYPE *pxTopOfStack, pdTASK_CODE pxCode, void *pvParameters)`
- `void * pvPortMalloc (size_t xSize) PRIVILEGED_FUNCTION`
- `void vPortFree (void *pv) PRIVILEGED_FUNCTION`
- `void vPortInitialiseBlocks (void) PRIVILEGED_FUNCTION`
- `size_t xPortGetFreeHeapSize (void) PRIVILEGED_FUNCTION`
- `portBASE_TYPE xPortStartScheduler (void) PRIVILEGED_FUNCTION`
- `void vPortEndScheduler (void) PRIVILEGED_FUNCTION`

22.21.1 Macro Definition Documentation

22.21.1.1 portBYTE_ALIGNMENT_MASK `#define portBYTE_ALIGNMENT_MASK (0x0000)`

Definition at line 323 of file `portable.h`.

22.21.1.2 portNUM_CONFIGURABLE_REGIONS #define portNUM_CONFIGURABLE_REGIONS 1

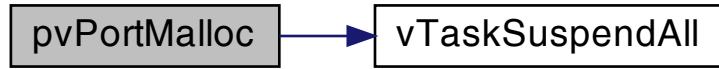
Definition at line 331 of file [portable.h](#).

22.21.2 Function Documentation

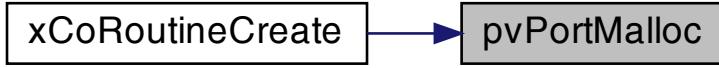
22.21.2.1 pvPortMalloc() void* pvPortMalloc (size_t xSize)

Definition at line 89 of file [heap_1.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.21.2.2 pxPortInitialiseStack() portSTACK_TYPE* pxPortInitialiseStack (portSTACK_TYPE * pxTopOfStack, pdTASK_CODE pxCode, void * pvParameters)

Definition at line 211 of file [port.c](#).

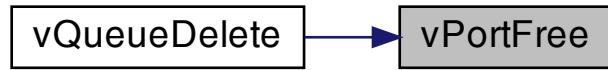
```
22.21.2.3 vPortEndScheduler() void vPortEndScheduler (
    void )
```

Definition at line 343 of file [port.c](#).

```
22.21.2.4 vPortFree() void vPortFree (
    void * pv )
```

Definition at line 130 of file [heap_1.c](#).

Here is the caller graph for this function:



```
22.21.2.5 vPortInitialiseBlocks() void vPortInitialiseBlocks (
    void )
```

Definition at line 139 of file [heap_1.c](#).

```
22.21.2.6 xPortGetFreeHeapSize() size_t xPortGetFreeHeapSize (
    void )
```

Definition at line 146 of file [heap_1.c](#).

```
22.21.2.7 xPortStartScheduler() portBASE_TYPE xPortStartScheduler (
    void )
```

Definition at line 326 of file [port.c](#).

22.22 portable.h

```

00001 /*
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005 ****
00006 *
00007 *   FreeRTOS tutorial books are available in pdf and paperback.
00008 *   Complete, revised, and edited pdf reference manuals are also
00009 *   available.
00010 *
00011 *   Purchasing FreeRTOS documentation will not only help you, by
00012 *   ensuring you get running as quickly as possible and with an
00013 *   in-depth knowledge of how to use FreeRTOS, it will also help
00014 *   the FreeRTOS project to continue with its mission of providing
00015 *   professional grade, cross platform, de facto standard solutions
00016 *   for microcontrollers - completely free of charge!
00017 *
00018 *   >>> See http://www.FreeRTOS.org/Documentation for details. <<
00019 *
00020 *   Thank you for using FreeRTOS, and thank you for your support!
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 >>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039 by writing to Richard Barry, contact details for whom are available on the
00040 FreeRTOS WEB site.
00041
00042 1 tab == 4 spaces!
00043
00044 http://www.FreeRTOS.org - Documentation, latest information, license and
00045 contact details.
00046
00047 http://www.SafeRTOS.com - A version that is certified for use in safety
00048 critical systems.
00049
00050 http://www.OpenRTOS.com - Commercial support, development, porting,
00051 licensing and training services.
00052 */
00053
00054 -----
00055 * Portable layer API. Each function must be defined for each port.
00056 -----
00057
00058 #ifndef PORTABLE_H
00059 #define PORTABLE_H
00060
00061 /* Include the macro file relevant to the port being used. */
00062
00063 #ifdef OPEN_WATCOM_INDUSTRIAL_PC_PORT
00064     #include "..\Source\portable\owatcom\16bitdos\pc\portmacro.h"
00065     typedef void ( __interrupt __far *pxISR )();
00066 #endif
00067
00068 #ifdef OPEN_WATCOM_FLASH_LITE_186_PORT
00069     #include "..\Source\portable\owatcom\16bitdos\flsh186\portmacro.h"
00070     typedef void ( __interrupt __far *pxISR )();
00071 #endif
00072
00073 #ifdef GCC_MEGA_AVR
00074     #include "../portable/GCC/ATMega323/portmacro.h"
00075 #endif
00076
00077 #ifdef IAR_MEGA_AVR
00078     #include "../portable/IAR/ATMega323/portmacro.h"
00079 #endif
00080
00081 #ifdef MPLAB_PIC24_PORT
00082     #include "..\Source\portable\MPLAB\PIC24_dsPIC\portmacro.h"
00083 #endif
00084
00085 #ifdef MPLAB_DSPIC_PORT

```

```
00086     #include "..\..\Source\portable\MPLAB\PIC24_dsPIC\portmacro.h"
00087 #endif
00088
00089 #ifdef MPLAB_PIC18F_PORT
00090     #include "..\..\Source\portable\MPLAB\PIC18F\portmacro.h"
00091 #endif
00092
00093 #ifdef MPLAB_PIC32MX_PORT
00094     #include "..\..\Source\portable\MPLAB\PIC32MX\portmacro.h"
00095 #endif
00096
00097 #ifdef _FEDPICC
00098     #include "libFreeRTOS/Include/portmacro.h"
00099 #endif
00100
00101 #ifdef SDCC_CYGNAL
00102     #include "../../Source/portable/SDCC/Cygnal/portmacro.h"
00103 #endif
00104
00105 #ifdef GCC_ARM7
00106     #include "../../Source/portable/GCC/ARM7_LPC2000/portmacro.h"
00107 #endif
00108
00109 #ifdef GCC_ARM7_ECLIPSE
00110     #include "portmacro.h"
00111 #endif
00112
00113 #ifdef ROWLEY_LPC23xx
00114     #include "../../Source/portable/GCC/ARM7_LPC23xx/portmacro.h"
00115 #endif
00116
00117 #ifdef IAR_MSP430
00118     #include "..\..\Source\portable\IAR\MSP430\portmacro.h"
00119 #endif
00120
00121 #ifdef GCC_MSP430
00122     #include "../../Source/portable/GCC/MSP430F449/portmacro.h"
00123 #endif
00124
00125 #ifdef ROWLEY_MSP430
00126     #include "../../Source/portable/Rowley/MSP430F449/portmacro.h"
00127 #endif
00128
00129 #ifdef ARM7_LPC21xx_KEIL_RVDS
00130     #include "..\..\Source\portable\RVDS\ARM7_LPC21xx\portmacro.h"
00131 #endif
00132
00133 #ifdef SAM7_GCC
00134     #include "../../Source/portable/GCC/ARM7_AT91SAM7S/portmacro.h"
00135 #endif
00136
00137 #ifdef SAM7_IAR
00138     #include "..\..\Source\portable\IAR\AtmelSAM7S64\portmacro.h"
00139 #endif
00140
00141 #ifdef SAM9XE_IAR
00142     #include "..\..\Source\portable\IAR\AtmelSAM9XE\portmacro.h"
00143 #endif
00144
00145 #ifdef LPC2000_IAR
00146     #include "..\..\Source\portable\IAR\LPC2000\portmacro.h"
00147 #endif
00148
00149 #ifdef STR71X_IAR
00150     #include "..\..\Source\portable\IAR\STR71x\portmacro.h"
00151 #endif
00152
00153 #ifdef STR75X_IAR
00154     #include "..\..\Source\portable\IAR\STR75x\portmacro.h"
00155 #endif
00156
00157 #ifdef STR75X_GCC
00158     #include "../../Source\portable\GCC\STR75x\portmacro.h"
00159 #endif
00160
00161 #ifdef STR91X_IAR
00162     #include "..\..\Source\portable\IAR\STR91x\portmacro.h"
00163 #endif
00164
00165 #ifdef GCC_H8S
00166     #include "../../Source\portable\GCC\H8S2329\portmacro.h"
00167 #endif
00168
00169 #ifdef GCC_AT91FR40008
00170     #include "../../Source\portable\GCC\ARM7_AT91FR40008\portmacro.h"
00171 #endif
00172
```

```

00173 #ifdef RVDS_ARMCM3_LM3S102
00174     #include "../../Source/portable/RVDS/ARM_CM3/portmacro.h"
00175 #endif
00176
00177 #ifdef GCC_ARMCM3_LM3S102
00178     #include "../../Source/portable/GCC/ARM_CM3/portmacro.h"
00179 #endif
00180
00181 #ifdef GCC_ARMCM3
00182     #include "../../Source/portable/GCC/ARM_CM3/portmacro.h"
00183 #endif
00184
00185 #ifdef IAR_ARM_CM3
00186     #include "../../Source/portable/IAR/ARM_CM3/portmacro.h"
00187 #endif
00188
00189 #ifdef IAR_ARMCM3_LM
00190     #include "../../Source/portable/IAR/ARM_CM3/portmacro.h"
00191 #endif
00192
00193 #ifdef HCS12_CODE_WARRIOR
00194     #include "../../Source/portable/CodeWarrior/HCS12/portmacro.h"
00195 #endif
00196
00197 #ifdef MICROBLAZE_GCC
00198     #include "../../Source/portable/GCC/MicroBlaze/portmacro.h"
00199 #endif
00200
00201 #ifdef TERN_EE
00202     #include "..\..\Source\portable\Paradigm\Tern_EE\small\portmacro.h"
00203 #endif
00204
00205 #ifdef GCC_HCS12
00206     #include "../../Source/portable/GCC/HCS12/portmacro.h"
00207 #endif
00208
00209 #ifdef GCC_MCF5235
00210     #include "../../Source/portable/GCC/MCF5235/portmacro.h"
00211 #endif
00212
00213 #ifdef COLDFIRE_V2_GCC
00214     #include "../../Source/portable/GCC/ColdFire_V2/portmacro.h"
00215 #endif
00216
00217 #ifdef COLDFIRE_V2_CODEWARRIOR
00218     #include "../../Source/portable/CodeWarrior/ColdFire_V2/portmacro.h"
00219 #endif
00220
00221 #ifdef GCC_PPC405
00222     #include "../../Source/portable/GCC/PPC405_Xilinx/portmacro.h"
00223 #endif
00224
00225 #ifdef GCC_PPC440
00226     #include "../../Source/portable/GCC/PPC440_Xilinx/portmacro.h"
00227 #endif
00228
00229 #ifdef _16FX_SOFTUNE
00230     #include "..\..\Source\portable\Softune\MB96340\portmacro.h"
00231 #endif
00232
00233 #ifdef BCC_INDUSTRIAL_PC_PORT
00234     /* A short file name has to be used in place of the normal
00235        FreeRTOSConfig.h when using the Borland compiler. */
00236     #include "frconfig.h"
00237     #include "..\portable\BCC\16BitDOS\PC\prtmacro.h"
00238     typedef void ( __interrupt __far *pxISR )();
00239 #endif
00240
00241 #ifdef BCC_FLASH_LITE_186_PORT
00242     /* A short file name has to be used in place of the normal
00243        FreeRTOSConfig.h when using the Borland compiler. */
00244     #include "frconfig.h"
00245     #include "..\portable\BCC\16BitDOS\flsh186\prtmacro.h"
00246     typedef void ( __interrupt __far *pxISR )();
00247 #endif
00248
00249 #ifdef __GNUC__
00250     #ifdef __AVR32_AVR32A__
00251         #include "portmacro.h"
00252     #endif
00253 #endif
00254
00255 #ifdef __ICCAVR32__
00256     #ifdef __CORE__
00257         #if __CORE__ == __AVR32A__
00258             #include "portmacro.h"
00259         #endif

```

```
00260     #endif
00261 #endif
00262
00263 #ifdef __91467D
00264     #include "portmacro.h"
00265 #endif
00266
00267 #ifdef __96340
00268     #include "portmacro.h"
00269 #endif
00270
00271
00272 #ifdef __IAR_V850ES_Fx3__
00273     #include "../../Source/portable/IAR/V850ES/portmacro.h"
00274 #endif
00275
00276 #ifdef __IAR_V850ES_Jx3__
00277     #include "../../Source/portable/IAR/V850ES/portmacro.h"
00278 #endif
00279
00280 #ifdef __IAR_V850ES_Jx3_L__
00281     #include "../../Source/portable/IAR/V850ES/portmacro.h"
00282 #endif
00283
00284 #ifdef __IAR_V850ES_Jx2__
00285     #include "../../Source/portable/IAR/V850ES/portmacro.h"
00286 #endif
00287
00288 #ifdef __IAR_V850ES_Hx2__
00289     #include "../../Source/portable/IAR/V850ES/portmacro.h"
00290 #endif
00291
00292 #ifdef __IAR_78K0R_Kx3__
00293     #include "../../Source/portable/IAR/78K0R/portmacro.h"
00294 #endif
00295
00296 #ifdef __IAR_78K0R_Kx3L__
00297     #include "../../Source/portable/IAR/78K0R/portmacro.h"
00298 #endif
00299
00300 /* Catch all to ensure portmacro.h is included in the build. Newer demos
00301 have the path as part of the project options, rather than as relative from
00302 the project location. If portENTER_CRITICAL() has not been defined then
00303 portmacro.h has not yet been included - as every portmacro.h provides a
00304 portENTER_CRITICAL() definition. Check the demo application for your demo
00305 to find the path to the correct portmacro.h file. */
00306 #ifndef portENTER_CRITICAL
00307     #include "portmacro.h"
00308 #endif
00309
00310 #if portBYTE_ALIGNMENT == 8
00311     #define portBYTE_ALIGNMENT_MASK ( 0x0007 )
00312 #endif
00313
00314 #if portBYTE_ALIGNMENT == 4
00315     #define portBYTE_ALIGNMENT_MASK ( 0x0003 )
00316 #endif
00317
00318 #if portBYTE_ALIGNMENT == 2
00319     #define portBYTE_ALIGNMENT_MASK ( 0x0001 )
00320 #endif
00321
00322 #if portBYTE_ALIGNMENT == 1
00323     #define portBYTE_ALIGNMENT_MASK ( 0x0000 )
00324 #endif
00325
00326 #ifndef portBYTE_ALIGNMENT_MASK
00327     #error "Invalid portBYTE_ALIGNMENT definition"
00328 #endif
00329
00330 #ifndef portNUM_CONFIGURABLE_REGIONS
00331     #define portNUM_CONFIGURABLE_REGIONS 1
00332 #endif
00333
00334 #ifdef __cplusplus
00335 extern "C" {
00336 #endif
00337
00338 #include "mpu_wrappers.h"
00339
00340 /*
00341 * Setup the stack of a new task so it is ready to be placed under the
00342 * scheduler control. The registers have to be placed on the stack in
00343 * the order that the port expects to find them.
00344 *
00345 */
00346 #if( portUSING_MPU_WRAPPERS == 1 )
```

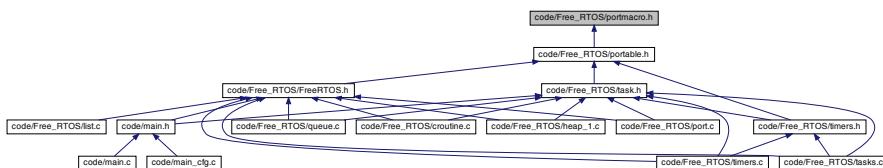
```

00347     portSTACK_TYPE *pxPortInitialiseStack( portSTACK_TYPE *pxTopOfStack, pdTASK_CODE pxCode, void
00348     *pvParameters, portBASE_TYPE xRunPrivileged ) PRIVILEGED_FUNCTION;
00349     portSTACK_TYPE *pxPortInitialiseStack( portSTACK_TYPE *pxTopOfStack, pdTASK_CODE pxCode, void
00350     *pvParameters );
00351 #endif
00352 /*
00353  * Map to the memory management routines required for the port.
00354 */
00355 void *pvPortMalloc( size_t xSize ) PRIVILEGED_FUNCTION;
00356 void vPortFree( void *pv ) PRIVILEGED_FUNCTION;
00357 void vPortInitialiseBlocks( void ) PRIVILEGED_FUNCTION;
00358 size_t xPortGetFreeHeapSize( void ) PRIVILEGED_FUNCTION;
00359
00360 /*
00361  * Setup the hardware ready for the scheduler to take control. This generally
00362  * sets up a tick interrupt and sets timers for the correct tick frequency.
00363 */
00364 portBASE_TYPE xPortStartScheduler( void ) PRIVILEGED_FUNCTION;
00365
00366 /*
00367  * Undo any hardware/ISR setup that was performed by xPortStartScheduler() so
00368  * the hardware is left in its original condition after the scheduler stops
00369  * executing.
00370 */
00371 void vPortEndScheduler( void ) PRIVILEGED_FUNCTION;
00372
00373 /*
00374  * The structures and methods of manipulating the MPU are contained within the
00375  * port layer.
00376 *
00377  * Fills the xMPUSettings structure with the memory region information
00378  * contained in xRegions.
00379 */
00380 #if( portUSING_MPUs == 1 )
00381     struct xMEMORY_REGION;
00382     void vPortStoreTaskMPUSettings( xMPU_SETTINGS *xMPUSettings, const struct xMEMORY_REGION * const
00383     xRegions, portSTACK_TYPE *pxBottomOfStack, unsigned short usStackDepth ) PRIVILEGED_FUNCTION;
00384 #endif
00385 #ifdef __cplusplus
00386 }
00387#endif
00388 #endif /* PORTABLE_H */
00390

```

22.23 code/FreeRTOS/portmacro.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define portCHAR char
- #define portFLOAT float
- #define portDOUBLE double
- #define portLONG long
- #define portSHORT int
- #define portSTACK_TYPE unsigned portCHAR
- #define portBASE_TYPE char

- #define portMAX_DELAY (portTickType) 0xffffffff
- #define portENTER_CRITICAL()
- #define portEXIT_CRITICAL()
- #define portDISABLE_INTERRUPTS() asm volatile ("cli" ::);
- #define portENABLE_INTERRUPTS() asm volatile ("sei" ::);
- #define portSTACK_GROWTH (-1)
- #define portTICK_RATE_MS ((portTickType) 1000 / configTICK_RATE_HZ)
- #define portBYTE_ALIGNMENT 1
- #define portNOP() asm volatile ("nop");
- #define portYIELD() vPortYield()
- #define portTASK_FUNCTION_PROTO(vFunction, pvParameters) void vFunction(void *pvParameters)
- #define portTASK_FUNCTION(vFunction, pvParameters) void vFunction(void *pvParameters)

TypeDefs

- typedef unsigned portLONG portTickType

Functions

- void vPortYield (void) __attribute__((naked))

22.23.1 Macro Definition Documentation

22.23.1.1 portBASE_TYPE #define portBASE_TYPE char

Definition at line 85 of file [portmacro.h](#).

22.23.1.2 portBYTE_ALIGNMENT #define portBYTE_ALIGNMENT 1

Definition at line 111 of file [portmacro.h](#).

22.23.1.3 portCHAR #define portCHAR char

Definition at line 79 of file [portmacro.h](#).

22.23.1.4 portDISABLE_INTERRUPTS #define portDISABLE_INTERRUPTS() asm volatile ("cli" ::);

Definition at line 104 of file [portmacro.h](#).

22.23.1.5 portDOUBLE #define portDOUBLE double

Definition at line 81 of file [portmacro.h](#).

22.23.1.6 portENABLE_INTERRUPTS #define portENABLE_INTERRUPTS() asm volatile ("sei" ::);

Definition at line 105 of file [portmacro.h](#).

22.23.1.7 portENTER_CRITICAL #define portENTER_CRITICAL()

Value:

```
asm volatile ( "in    __tmp_reg__, __SREG__" :: );      \
asm volatile ( "cli" :: );                                \
asm volatile ( "push  __tmp_reg__" :: )
```

Definition at line 97 of file [portmacro.h](#).

22.23.1.8 portEXIT_CRITICAL #define portEXIT_CRITICAL()

Value:

```
asm volatile ( "pop   __tmp_reg__" :: );      \
asm volatile ( "out   __SREG__, __tmp_reg__" :: )
```

Definition at line 101 of file [portmacro.h](#).

22.23.1.9 portFLOAT #define portFLOAT float

Definition at line 80 of file [portmacro.h](#).

22.23.1.10 portLONG #define portLONG long

Definition at line 82 of file [portmacro.h](#).

22.23.1.11 portMAX_DELAY #define portMAX_DELAY (portTickType) 0xffffffff

Definition at line 92 of file [portmacro.h](#).

22.23.1.12 portNOP #define portNOP() asm volatile ("nop");

Definition at line 112 of file [portmacro.h](#).

22.23.1.13 portSHORT #define portSHORT int

Definition at line 83 of file [portmacro.h](#).

22.23.1.14 portSTACK_GROWTH #define portSTACK_GROWTH (-1)

Definition at line 109 of file [portmacro.h](#).

22.23.1.15 portSTACK_TYPE #define portSTACK_TYPE unsigned [portCHAR](#)

Definition at line 84 of file [portmacro.h](#).

22.23.1.16 portTASK_FUNCTION #define portTASK_FUNCTION(

vFunction,
pvParameters) void *vFunction(void *pvParameters)*

Definition at line 122 of file [portmacro.h](#).

22.23.1.17 portTASK_FUNCTION_PROTO #define portTASK_FUNCTION_PROTO(

vFunction,
pvParameters) void *vFunction(void *pvParameters)*

Definition at line 121 of file [portmacro.h](#).

22.23.1.18 portTICK_RATE_MS #define portTICK_RATE_MS (([portTickType](#)) 1000 / [configTICK_RATE_HZ](#))

Definition at line 110 of file [portmacro.h](#).

22.23.1.19 portYIELD #define portYIELD() [vPortYield\(\)](#)

Definition at line 117 of file [portmacro.h](#).

22.23.2 Typedef Documentation

22.23.2.1 portTickType `typedef unsigned portLONG portTickType`

Definition at line 91 of file [portmacro.h](#).

22.23.3 Function Documentation

22.23.3.1 vPortYield() `void vPortYield (void)`

Definition at line 354 of file [port.c](#).

22.24 portmacro.h

```

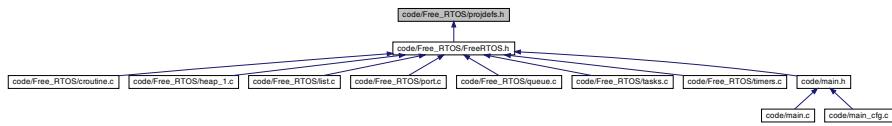
00001 /*
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005 ****
00006 *
00007 *   FreeRTOS tutorial books are available in pdf and paperback.
00008 *   Complete, revised, and edited pdf reference manuals are also
00009 *   available.
00010 *
00011 *   Purchasing FreeRTOS documentation will not only help you, by
00012 *   ensuring you get running as quickly as possible and with an
00013 *   in-depth knowledge of how to use FreeRTOS, it will also help
00014 *   the FreeRTOS project to continue with its mission of providing
00015 *   professional grade, cross platform, de facto standard solutions
00016 *   for microcontrollers - completely free of charge!
00017 *
00018 *   >> See http://www.FreeRTOS.org/Documentation for details. <<
00019 *
00020 *   Thank you for using FreeRTOS, and thank you for your support!
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 >>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039 by writing to Richard Barry, contact details for whom are available on the
00040 FreeRTOS WEB site.
00041
00042 1 tab == 4 spaces!
00043
00044 http://www.FreeRTOS.org - Documentation, latest information, license and
00045 contact details.
00046
00047 http://www.SafeRTOS.com - A version that is certified for use in safety
00048 critical systems.
00049
00050 http://www.OpenRTOS.com - Commercial support, development, porting,
00051 licensing and training services.

```

```
00052 */
00053
00054 /*
00055 Changes from V1.2.3
00056
00057     + portCPU_CLOSK_HZ definition changed to 8MHz base 10, previously it
00058         base 16.
00059 */
00060
00061 #ifndef PORTMACRO_H
00062 #define PORTMACRO_H
00063
00064 #ifdef __cplusplus
00065 extern "C" {
00066 #endif
00067
00068 /-----
00069 * Port specific definitions.
00070 *
00071 * The settings in this file configure FreeRTOS correctly for the
00072 * given hardware and compiler.
00073 *
00074 * These settings should not be altered.
00075 -----
00076 */
00077
00078 /* Type definitions. */
00079 #define portCHAR          char
00080 #define portFLOAT          float
00081 #define portDOUBLE         double
00082 #define portLONG           long
00083 #define portSHORT          int
00084 #define portSTACK_TYPE     unsigned portCHAR
00085 #define portBASE_TYPE      char
00086
00087 #if( configUSE_16_BIT_TICKS == 1 )
00088     typedef unsigned portSHORT portTickType;
00089     #define portMAX_DELAY ( portTickType ) 0xffff
00090 #else
00091     typedef unsigned portLONG portTickType;
00092     #define portMAX_DELAY ( portTickType ) 0xffffffff
00093 #endif
00094 /-----
00095
00096 /* Critical section management. */
00097 #define portENTER_CRITICAL()          asm volatile ( "in    __tmp_reg__, __SREG__" :: ); \
00098                                         asm volatile ( "cli" :: ); \
00099                                         asm volatile ( "push   __tmp_reg__" :: )
00100
00101 #define portEXIT_CRITICAL()         asm volatile ( "pop   __tmp_reg__" :: );
00102                                         asm volatile ( "out   __SREG__, __tmp_reg__" :: )
00103
00104 #define portDISABLE_INTERRUPTS()    asm volatile ( "cli" :: );
00105 #define portENABLE_INTERRUPTS()     asm volatile ( "sei" :: );
00106 /-----
00107
00108 /* Architecture specifics. */
00109 #define portSTACK_GROWTH          ( -1 )
00110 #define portTICK_RATE_MS          ( ( portTickType ) 1000 / configTICK_RATE_HZ )
00111 #define portBYTE_ALIGNMENT        1
00112 #define portNOP()                asm volatile ( "nop" );
00113 /-----
00114
00115 /* Kernel utilities. */
00116 extern void vPortYield( void ) __attribute__ ( ( naked ) );
00117 #define portYIELD()              vPortYield()
00118 /-----
00119
00120 /* Task function macros as described on the FreeRTOS.org WEB site. */
00121 #define portTASK_FUNCTION_PROTO( vFunction, pvParameters ) void vFunction( void *pvParameters )
00122 #define portTASK_FUNCTION( vFunction, pvParameters ) void vFunction( void *pvParameters )
00123
00124 #ifdef __cplusplus
00125 }
00126 #endif
00127
00128 #endif /* PORTMACRO_H */
00129
```

22.25 code/FreeRTOS/projdefs.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define pdTRUE (1)
- #define pdFALSE (0)
- #define pdPASS (1)
- #define pdFAIL (0)
- #define errQUEUE_EMPTY (0)
- #define errQUEUE_FULL (0)
- #define errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY (-1)
- #define errNO_TASK_TO_RUN (-2)
- #define errQUEUE_BLOCKED (-4)
- #define errQUEUE_YIELD (-5)

TypeDefs

- typedef void(* pdTASK_CODE) (void *)

22.25.1 Macro Definition Documentation

22.25.1.1 errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY #define errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY (-1)

Definition at line 69 of file [projdefs.h](#).

22.25.1.2 errNO_TASK_TO_RUN #define errNO_TASK_TO_RUN (-2)

Definition at line 70 of file [projdefs.h](#).

22.25.1.3 errQUEUE_BLOCKED #define errQUEUE_BLOCKED (-4)

Definition at line 71 of file [projdefs.h](#).

22.25.1.4 errQUEUE_EMPTY #define errQUEUE_EMPTY (0)

Definition at line 65 of file [projdefs.h](#).

22.25.1.5 errQUEUE_FULL #define errQUEUE_FULL (0)

Definition at line 66 of file [projdefs.h](#).

22.25.1.6 errQUEUE_YIELD #define errQUEUE_YIELD (-5)

Definition at line 72 of file [projdefs.h](#).

22.25.1.7 pdFAIL #define pdFAIL (0)

Definition at line 64 of file [projdefs.h](#).

22.25.1.8 pdFALSE #define pdFALSE (0)

Definition at line 61 of file [projdefs.h](#).

22.25.1.9 pdPASS #define pdPASS (1)

Definition at line 63 of file [projdefs.h](#).

22.25.1.10 pdTRUE #define pdTRUE (1)

Definition at line 60 of file [projdefs.h](#).

22.25.2 Typedef Documentation

22.25.2.1 pdTASK_CODE typedef void(* pdTASK_CODE) (void *)

Definition at line 58 of file [projdefs.h](#).

22.26 projdefs.h

```

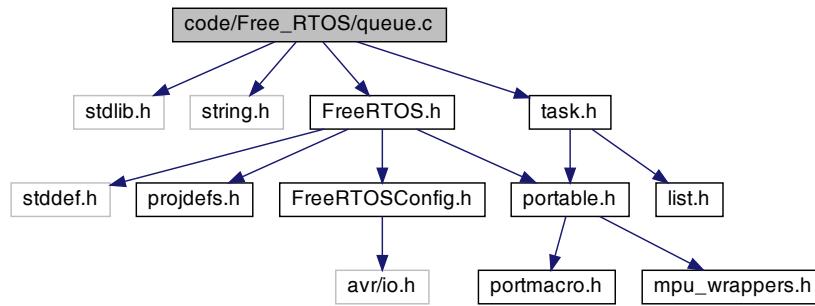
00001 /*
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005 ****
00006 *
00007 *   FreeRTOS tutorial books are available in pdf and paperback.
00008 *   Complete, revised, and edited pdf reference manuals are also
00009 *   available.
00010 *
00011 *   Purchasing FreeRTOS documentation will not only help you, by
00012 *   ensuring you get running as quickly as possible and with an
00013 *   in-depth knowledge of how to use FreeRTOS, it will also help
00014 *   the FreeRTOS project to continue with its mission of providing
00015 *   professional grade, cross platform, de facto standard solutions
00016 *   for microcontrollers - completely free of charge!
00017 *
00018 *   >>> See http://www.FreeRTOS.org/Documentation for details. <<
00019 *
00020 *   Thank you for using FreeRTOS, and thank you for your support!
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 >>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039 by writing to Richard Barry, contact details for whom are available on the
00040 FreeRTOS WEB site.
00041
00042 1 tab == 4 spaces!
00043
00044 http://www.FreeRTOS.org - Documentation, latest information, license and
00045 contact details.
00046
00047 http://www.SafeRTOS.com - A version that is certified for use in safety
00048 critical systems.
00049
00050 http://www.OpenRTOS.com - Commercial support, development, porting,
00051 licensing and training services.
00052 */
00053
00054 #ifndef PROJDEFS_H
00055 #define PROJDEFS_H
00056
00057 /* Defines the prototype to which task functions must conform. */
00058 typedef void (*pdTASK_CODE)( void * );
00059
00060 #define pdTRUE      ( 1 )
00061 #define pdFALSE     ( 0 )
00062
00063 #define pdPASS      ( 1 )
00064 #define pdFAIL      ( 0 )
00065 #define errQUEUE_EMPTY    ( 0 )
00066 #define errQUEUE_FULL     ( 0 )
00067
00068 /* Error definitions. */
00069 #define errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY ( -1 )
00070 #define errNO_TASK_TO_RUN        ( -2 )
00071 #define errQUEUE_BLOCKED        ( -4 )
00072 #define errQUEUE_YIELD          ( -5 )
00073
00074 #endif /* PROJDEFS_H */
00075
00076
00077

```

22.27 code/FreeRTOS/queue.c File Reference

```
#include <stdlib.h>
#include <string.h>
```

```
#include "FreeRTOS.h"
#include "task.h"
Include dependency graph for queue.c:
```



Data Structures

- struct [QueueDefinition](#)

Macros

- #define MPU_WRAPPERS_INCLUDED_FROM_API_FILE
- #define queueUNLOCKED ((signed portBASE_TYPE) -1)
- #define queueLOCKED_UNMODIFIED ((signed portBASE_TYPE) 0)
- #define queueERRONEOUS_UNBLOCK (-1)
- #define queueSEND_TO_BACK (0)
- #define queueSEND_TO_FRONT (1)
- #define pxMutexHolder pcTail
- #define uxQueueType pcHead
- #define uxRecursiveCallCount pcReadFrom
- #define queueQUEUE_IS_MUTEX NULL
- #define queueSemaphore_QUEUE_ITEM_LENGTH ((unsigned portBASE_TYPE) 0)
- #define queueDONT_BLOCK ((portTickType) 0U)
- #define queueMUTEX_GIVE_BLOCK_TIME ((portTickType) 0U)
- #define prvLockQueue(pxQueue)

Typedefs

- typedef struct [QueueDefinition](#) xQUEUE
- typedef xQUEUE * xQueueHandle

Functions

- `xQueueHandle xQueueCreate (unsigned portBASE_TYPE uxQueueLength, unsigned portBASE_TYPE uxItemSize) PRIVILEGED_FUNCTION`
- `signed portBASE_TYPE xQueueGenericSend (xQueueHandle xQueue, const void *const pvItemToQueue, portTickType xTicksToWait, portBASE_TYPE xCopyPosition) PRIVILEGED_FUNCTION`
- `unsigned portBASE_TYPE uxQueueMessagesWaiting (const xQueueHandle pxQueue) PRIVILEGED_FUNCTION`
- `void vQueueDelete (xQueueHandle xQueue) PRIVILEGED_FUNCTION`
- `signed portBASE_TYPE xQueueGenericSendFromISR (xQueueHandle pxQueue, const void *const pvItemToQueue, signed portBASE_TYPE *pxHigherPriorityTaskWoken, portBASE_TYPE xCopyPosition) PRIVILEGED_FUNCTION`
- `signed portBASE_TYPE xQueueGenericReceive (xQueueHandle pxQueue, void *const pvBuffer, portTickType xTicksToWait, portBASE_TYPE xJustPeeking) PRIVILEGED_FUNCTION`
- `signed portBASE_TYPE xQueueReceiveFromISR (xQueueHandle pxQueue, void *const pvBuffer, signed portBASE_TYPE *pxTaskWoken) PRIVILEGED_FUNCTION`
- `xQueueHandle xQueueCreateMutex (void)`
- `xQueueHandle xQueueCreateCountingSemaphore (unsigned portBASE_TYPE uxCountValue, unsigned portBASE_TYPE uxInitialCount)`
- `unsigned portBASE_TYPE uxQueueMessagesWaitingFromISR (const xQueueHandle pxQueue)`
- `signed portBASE_TYPE xQueueIsQueueEmptyFromISR (const xQueueHandle pxQueue)`
- `signed portBASE_TYPE xQueueIsQueueFullFromISR (const xQueueHandle pxQueue)`

22.27.1 Macro Definition Documentation

22.27.1.1 MPU_WRAPPERS_INCLUDED_FROM_API_FILE #define MPU_WRAPPERS_INCLUDED_FROM_API_FILE

Definition at line 60 of file [queue.c](#).

22.27.1.2 prvLockQueue #define prvLockQueue (
 pxQueue)

Value:

```
taskENTER_CRITICAL();
{
    if( ( pxQueue )->xRxLock == queueUNLOCKED )
    {
        ( pxQueue )->xRxLock = queueLOCKED_UNMODIFIED;
    }
    if( ( pxQueue )->xTxLock == queueUNLOCKED )
    {
        ( pxQueue )->xTxLock = queueLOCKED_UNMODIFIED;
    }
}
taskEXIT_CRITICAL();
```

22.27.1.3 pxMutexHolder #define pxMutexHolder pcTail

Definition at line 86 of file [queue.c](#).

22.27.1.4 queueDONT_BLOCK #define queueDONT_BLOCK ((portTickType) 0U)

Definition at line 94 of file [queue.c](#).

22.27.1.5 queueERRONEOUS_UNBLOCK #define queueERRONEOUS_UNBLOCK (-1)

Definition at line 79 of file [queue.c](#).

22.27.1.6 queueLOCKED_UNMODIFIED #define queueLOCKED_UNMODIFIED ((signed portBASE_TYPE) 0)

Definition at line 77 of file [queue.c](#).

22.27.1.7 queueMUTEX_GIVE_BLOCK_TIME #define queueMUTEX_GIVE_BLOCK_TIME ((portTickType) 0U)

Definition at line 95 of file [queue.c](#).

22.27.1.8 queueQUEUE_IS_MUTEX #define queueQUEUE_IS_MUTEX NULL

Definition at line 89 of file [queue.c](#).

22.27.1.9 queueSEMAPHORE_QUEUE_ITEM_LENGTH #define queueSEMAPHORE_QUEUE_ITEM_LENGTH ((unsigned portBASE_TYPE) 0)

Definition at line 93 of file [queue.c](#).

22.27.1.10 queueSEND_TO_BACK #define queueSEND_TO_BACK (0)

Definition at line 82 of file [queue.c](#).

22.27.1.11 queueSEND_TO_FRONT #define queueSEND_TO_FRONT (1)

Definition at line 83 of file [queue.c](#).

22.27.1.12 queueUNLOCKED #define queueUNLOCKED ((signed portBASE_TYPE) -1)

Definition at line 76 of file [queue.c](#).

22.27.1.13 uxQueueType #define uxQueueType pcHead

Definition at line 87 of file [queue.c](#).

22.27.1.14 uxRecursiveCallCount #define uxRecursiveCallCount pcReadFrom

Definition at line 88 of file [queue.c](#).

22.27.2 Typedef Documentation

22.27.2.1 xQUEUE typedef struct QueueDefinition xQUEUE

22.27.2.2 xQueueHandle typedef xQUEUE* xQueueHandle

Definition at line 127 of file [queue.c](#).

22.27.3 Function Documentation

22.27.3.1 uxQueueMessagesWaiting() unsigned portBASE_TYPE uxQueueMessagesWaiting (const xQueueHandle pxQueue)

Definition at line 1051 of file [queue.c](#).

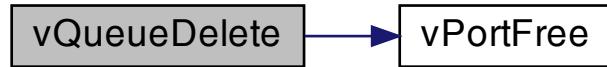
22.27.3.2 uxQueueMessagesWaitingFromISR() unsigned portBASE_TYPE uxQueueMessagesWaiting← FromISR (const xQueueHandle pxQueue)

Definition at line 1065 of file [queue.c](#).

```
22.27.3.3 vQueueDelete() void vQueueDelete (
    xQueueHandle xQueue )
```

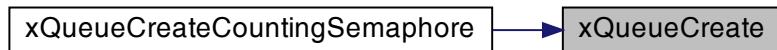
Definition at line 1077 of file [queue.c](#).

Here is the call graph for this function:



```
22.27.3.4 xQueueCreate() xQueueHandle xQueueCreate (
    unsigned portBASE_TYPE uxQueueLength,
    unsigned portBASE_TYPE uxItemSize )
```

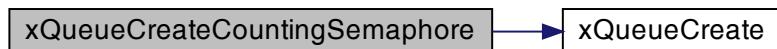
Here is the caller graph for this function:



```
22.27.3.5 xQueueCreateCountingSemaphore() xQueueHandle xQueueCreateCountingSemaphore (
    unsigned portBASE_TYPE uxCountValue,
    unsigned portBASE_TYPE uxInitialCount )
```

Definition at line 440 of file [queue.c](#).

Here is the call graph for this function:



22.27.3.6 xQueueCreateMutex() `signed portBASE_TYPE xQueueCreateMutex (`
 `void)`

Definition at line 141 of file [queue.c](#).

22.27.3.7 xQueueGenericReceive() `signed portBASE_TYPE xQueueGenericReceive (`
 `xQueueHandle pxQueue,`
 `void *const pvBuffer,`
 `portTickType xTicksToWait,`
 `portBASE_TYPE xJustPeeking)`

Definition at line 846 of file [queue.c](#).

22.27.3.8 xQueueGenericSend() `signed portBASE_TYPE xQueueGenericSend (`
 `xQueueHandle xQueue,`
 `const void *const pvItemToQueue,`
 `portTickType xTicksToWait,`
 `portBASE_TYPE xCopyPosition)`

Definition at line 464 of file [queue.c](#).

22.27.3.9 xQueueGenericSendFromISR() `signed portBASE_TYPE xQueueGenericSendFromISR (`
 `xQueueHandle pxQueue,`
 `const void *const pvItemToQueue,`
 `signed portBASE_TYPE * pxHigherPriorityTaskWoken,`
 `portBASE_TYPE xCopyPosition)`

Definition at line 789 of file [queue.c](#).

22.27.3.10 xQueueIsQueueEmptyFromISR() `signed portBASE_TYPE xQueueIsQueueEmptyFromISR (`
 `const xQueueHandle pxQueue)`

Definition at line 1216 of file [queue.c](#).

22.27.3.11 xQueueIsQueueFullFromISR() `signed portBASE_TYPE xQueueIsQueueFullFromISR (`
 `const xQueueHandle pxQueue)`

Definition at line 1239 of file [queue.c](#).

```
22.27.3.12 xQueueReceiveFromISR() signed portBASE_TYPE xQueueReceiveFromISR (
    xQueueHandle pxQueue,
    void *const pvBuffer,
    signed portBASE_TYPE * pxTaskWoken )
```

Definition at line 996 of file [queue.c](#).

22.28 queue.c

```
00001 /*
00002     FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005 ****
00006 *
00007 *     FreeRTOS tutorial books are available in pdf and paperback.
00008 *     Complete, revised, and edited pdf reference manuals are also
00009 *     available.
00010 *
00011 *     Purchasing FreeRTOS documentation will not only help you, by
00012 *     ensuring you get running as quickly as possible and with an
00013 *     in-depth knowledge of how to use FreeRTOS, it will also help
00014 *     the FreeRTOS project to continue with its mission of providing
00015 *     professional grade, cross platform, de facto standard solutions
00016 *     for microcontrollers - completely free of charge!
00017 *
00018 *     >> See http://www.FreeRTOS.org/Documentation for details. <<
00019 *
00020 *     Thank you for using FreeRTOS, and thank you for your support!
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 >>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039 by writing to Richard Barry, contact details for whom are available on the
00040 FreeRTOS WEB site.
00041
00042 1 tab == 4 spaces!
00043
00044 http://www.FreeRTOS.org - Documentation, latest information, license and
00045 contact details.
00046
00047 http://www.SafeRTOS.com - A version that is certified for use in safety
00048 critical systems.
00049
00050 http://www.OpenRTOS.com - Commercial support, development, porting,
00051 licensing and training services.
00052 */
00053
00054 #include <stdlib.h>
00055 #include <string.h>
00056
00057 /* Defining MPU_WRAPPERS_INCLUDED_FROM_API_FILE prevents task.h from redefining
00058 all the API functions to use the MPU wrappers. That should only be done when
00059 task.h is included from an application file. */
00060 #define MPU_WRAPPERS_INCLUDED_FROM_API_FILE
00061
00062 #include "FreeRTOS.h"
00063 #include "task.h"
00064
00065 #if ( configUSE_CO_ROUTINES == 1 )
00066     #include "croutine.h"
00067 #endif
00068
00069 #undef MPU_WRAPPERS_INCLUDED_FROM_API_FILE
00070
00071 /*****
00072 * PUBLIC LIST API documented in list.h
00073 ******/
```

```

00074
00075 /* Constants used with the cRxLock and cTxLock structure members. */
00076 #define queueUNLOCKED          ( ( signed portBASE_TYPE ) -1 )
00077 #define queueLOCKED_UNMODIFIED ( ( signed portBASE_TYPE ) 0 )
00078
00079 #define queueERRONEOUS_UNBLOCK ( -1 )
00080
00081 /* For internal use only. */
00082 #define queueSEND_TO_BACK      ( 0 )
00083 #define queueSEND_TO_FRONT     ( 1 )
00084
00085 /* Effectively make a union out of the xQUEUE structure. */
00086 #define pxMutexHolder          pcTail
00087 #define uxQueueType            pcHead
00088 #define uxRecursiveCallCount   pcReadFrom
00089 #define queueQUEUE_IS_MUTEX    NULL
00090
00091 /* Semaphores do not actually store or copy data, so have an items size of
00092 zero. */
00093 #define queueSEMAPHORE_QUEUE_ITEM_LENGTH ( ( unsigned portBASE_TYPE ) 0 )
00094 #define queueDONT_BLOCK         ( ( portTickType ) 0U )
00095 #define queueMUTEX_GIVE_BLOCK_TIME ( ( portTickType ) 0U )
00096
00097 /*
00098 * Definition of the queue used by the scheduler.
00099 * Items are queued by copy, not reference.
00100 */
00101 typedef struct QueueDefinition
00102 {
00103     signed char *pcHead;           /*< Points to the beginning of the queue storage area. */
00104     signed char *pcTail;          /*< Points to the byte at the end of the queue storage area.
00105     Once more byte is allocated than necessary to store the queue items, this is used as a marker. */
00106     signed char *pcWriteTo;        /*< Points to the free next place in the storage area. */
00107     signed char *pcReadFrom;      /*< Points to the last place that a queued item was read from.
00108 */
00109     xList xTasksWaitingToSend;    /*< List of tasks that are blocked waiting to post onto
00110     this queue. Stored in priority order. */
00111     xList xTasksWaitingToReceive; /*< List of tasks that are blocked waiting to read from
00112     this queue. Stored in priority order. */
00113     volatile unsigned portBASE_TYPE uxMessagesWaiting; /*< The number of items currently in the queue.
00114     */
00115     unsigned portBASE_TYPE uxLength; /*< The length of the queue defined as the number of items
00116     it will hold, not the number of bytes. */
00117     unsigned portBASE_TYPE uxItemSize; /*< The size of each items that the queue will hold. */
00118     signed portBASE_TYPE xRxLock;   /*< Stores the number of items received from the queue
00119     (removed from the queue) while the queue was locked. Set to queueUNLOCKED when the queue is not
00120     locked. */
00121     signed portBASE_TYPE xTxLock;   /*< Stores the number of items transmitted to the queue
00122     (added to the queue) while the queue was locked. Set to queueUNLOCKED when the queue is not locked.
00123 */
00124     * Inside this file xQueueHandle is a pointer to a xQUEUE structure.
00125     * To keep the definition private the API header file defines it as a
00126     * pointer to void.
00127     typedef xQUEUE * xQueueHandle;
00128
00129 /*
00130 * Prototypes for public functions are included here so we don't have to
00131 * include the API header file (as it defines xQueueHandle differently). These
00132 * functions are documented in the API header file.
00133 */
00134 xQueueHandle xQueueCreate( unsigned portBASE_TYPE uxQueueLength, unsigned portBASE_TYPE uxItemSize )
00135     PRIVILEGED_FUNCTION;
00136     signed portBASE_TYPE xQueueGenericSend( xQueueHandle xQueue, const void * const pvItemToQueue,
00137     portTickType xTicksToWait, portBASE_TYPE xCopyPosition ) PRIVILEGED_FUNCTION;
00138     unsigned portBASE_TYPE uxQueueMessagesWaiting( const xQueueHandle pxQueue ) PRIVILEGED_FUNCTION;
00139     void vQueueDelete( xQueueHandle xQueue ) PRIVILEGED_FUNCTION;
00140     signed portBASE_TYPE xQueueGenericSendFromISR( xQueueHandle pxQueue, const void * const pvItemToQueue,
00141     signed portBASE_TYPE *pxHigherPriorityTaskWoken, portBASE_TYPE xCopyPosition ) PRIVILEGED_FUNCTION;
00142     signed portBASE_TYPE xQueueGenericReceive( xQueueHandle pxQueue, void * const pvBuffer, portTickType
00143     xTicksToWait, portBASE_TYPE xJustPeeking ) PRIVILEGED_FUNCTION;
00144     signed portBASE_TYPE xQueueReceiveFromISR( xQueueHandle pxQueue, void * const pvBuffer, signed
00145     portBASE_TYPE *pxTaskWoken ) PRIVILEGED_FUNCTION;
00146     xQueueHandle xQueueCreateMutex( void ) PRIVILEGED_FUNCTION;
00147     xQueueHandle xQueueCreateCountingSemaphore( unsigned portBASE_TYPE uxCountValue, unsigned
00148     portBASE_TYPE uxInitialCount ) PRIVILEGED_FUNCTION;
00149     portBASE_TYPE xQueueTakeMutexRecursive( xQueueHandle xMutex, portTickType xBlockTime )
00150     PRIVILEGED_FUNCTION;

```

```

00144 portBASE_TYPE xQueueGiveMutexRecursive( xQueueHandle xMutex ) PRIVILEGED_FUNCTION;
00145 signed portBASE_TYPE xQueueAltGenericSend( xQueueHandle pxQueue, const void * const pvItemToQueue,
00146     portTickType xTicksToWait, portBASE_TYPE xCopyPosition ) PRIVILEGED_FUNCTION;
00147 signed portBASE_TYPE xQueueAltGenericReceive( xQueueHandle pxQueue, void * const pvBuffer,
00148     portTickType xTicksToWait, portBASE_TYPE xJustPeeking ) PRIVILEGED_FUNCTION;
00149 signed portBASE_TYPE xQueueIsEmptyFromISR( const xQueueHandle pxQueue ) PRIVILEGED_FUNCTION;
00150 signed portBASE_TYPE xQueueIsQueueFullFromISR( const xQueueHandle pxQueue ) PRIVILEGED_FUNCTION;
00151 unsigned portBASE_TYPE uxQueueMessagesWaitingFromISR( const xQueueHandle pxQueue )
00152     PRIVILEGED_FUNCTION;
00153 void vQueueWaitForMessageRestricted( xQueueHandle pxQueue, portTickType xTicksToWait )
00154     PRIVILEGED_FUNCTION;
00155 /*
00156  * Co-routine queue functions differ from task queue functions. Co-routines are
00157  * an optional component.
00158 */
00159 #if configUSE_CO_ROUTINES == 1
00160     signed portBASE_TYPE xQueueCRSendFromISR( xQueueHandle pxQueue, const void *pvItemToQueue, signed
00161         portBASE_TYPE xCoRoutinePreviouslyWoken ) PRIVILEGED_FUNCTION;
00162     signed portBASE_TYPE xQueueCRReceiveFromISR( xQueueHandle pxQueue, void *pvBuffer, signed
00163         portBASE_TYPE *pxTaskWoken ) PRIVILEGED_FUNCTION;
00164     signed portBASE_TYPE xQueueCRSend( xQueueHandle pxQueue, const void *pvItemToQueue, portTickType
00165         xTicksToWait ) PRIVILEGED_FUNCTION;
00166     signed portBASE_TYPE xQueueCRReceive( xQueueHandle pxQueue, void *pvBuffer, portTickType
00167         xTicksToWait ) PRIVILEGED_FUNCTION;
00168 #endif
00169 /*
00170  * The queue registry is just a means for kernel aware debuggers to locate
00171  * queue structures. It has no other purpose so is an optional component.
00172 */
00173 #if configQUEUE_REGISTRY_SIZE > 0
00174     /* The type stored within the queue registry array. This allows a name
00175      to be assigned to each queue making kernel aware debugging a little
00176      more user friendly. */
00177     typedef struct QUEUE_REGISTRY_ITEM
00178     {
00179         signed char *pcQueueName;
00180         xQueueHandle xHandle;
00181     } xQueueRegistryItem;
00182     /* The queue registry is simply an array of xQueueRegistryItem structures.
00183      The pcQueueName member of a structure being NULL is indicative of the
00184      array position being vacant. */
00185     xQueueRegistryItem xQueueRegistry[ configQUEUE_REGISTRY_SIZE ];
00186     /* Removes a queue from the registry by simply setting the pcQueueName
00187      member to NULL. */
00188     static void vQueueUnregisterQueue( xQueueHandle xQueue ) PRIVILEGED_FUNCTION;
00189     void vQueueAddToRegistry( xQueueHandle xQueue, signed char *pcQueueName ) PRIVILEGED_FUNCTION;
00190 #endif
00191 /*
00192  * Unlocks a queue locked by a call to prvLockQueue. Locking a queue does not
00193  * prevent an ISR from adding or removing items to the queue, but does prevent
00194  * an ISR from removing tasks from the queue event lists. If an ISR finds a
00195  * queue is locked it will instead increment the appropriate queue lock count
00196  * to indicate that a task may require unblocking. When the queue is unlocked
00197  * these lock counts are inspected, and the appropriate action taken.
00198 */
00199 static void prvUnlockQueue( xQueueHandle pxQueue ) PRIVILEGED_FUNCTION;
00200 /*
00201  * Uses a critical section to determine if there is any data in a queue.
00202  * @return pdTRUE if the queue contains no items, otherwise pdFALSE.
00203 */
00204 static signed portBASE_TYPE prvIsQueueEmpty( const xQueueHandle pxQueue ) PRIVILEGED_FUNCTION;
00205 /*
00206  * Uses a critical section to determine if there is any space in a queue.
00207  * @return pdTRUE if there is no space, otherwise pdFALSE;
00208 */
00209 static signed portBASE_TYPE prvIsQueueFull( const xQueueHandle pxQueue ) PRIVILEGED_FUNCTION;
00210 /*
00211  * Copies an item into the queue, either at the front of the queue or the
00212  * back of the queue.
00213 */
00214 static void prvCopyDataToQueue( xQUEUE *pxQueue, const void *pvItemToQueue, portBASE_TYPE xPosition )
00215     PRIVILEGED_FUNCTION;
00216 /*
00217  * Copies an item out of a queue.
00218 */
00219 static void prvCopyDataFromQueue( xQUEUE *pxQueue, void *pvItemFromQueue, portBASE_TYPE xPosition )
00220     PRIVILEGED_FUNCTION;

```

```

00222 static void prvCopyDataFromQueue( xQUEUE * const pxQueue, const void *pvBuffer ) PRIVILEGED_FUNCTION;
00223 /*****
00224 */
00225 /*
00226 * Macro to mark a queue as locked. Locking a queue prevents an ISR from
00227 * accessing the queue event lists.
00228 */
00229 #define prvLockQueue( pxQueue )
00230     taskENTER_CRITICAL();
00231     {
00232         if( ( pxQueue )->xRxLock == queueUNLOCKED )
00233         {
00234             ( pxQueue )->xRxLock = queueLOCKED_UNMODIFIED;
00235         }
00236         if( ( pxQueue )->xTxLock == queueUNLOCKED )
00237         {
00238             ( pxQueue )->xTxLock = queueLOCKED_UNMODIFIED;
00239         }
00240     }
00241     taskEXIT_CRITICAL()
00242 /*****
00243 */
00244
00245 /**
00246 * PUBLIC QUEUE MANAGEMENT API documented in queue.h
00247 */
00248
00249 xQueueHandle xQueueCreate( unsigned portBASE_TYPE uxQueueLength, unsigned portBASE_TYPE uxItemSize )
00250 {
00251     xQUEUE *pxNewQueue;
00252     size_t xQueueSizeInBytes;
00253     xQueueHandle xReturn = NULL;
00254
00255     /* Allocate the new queue structure. */
00256     if( uxQueueLength > ( unsigned portBASE_TYPE ) 0 )
00257     {
00258         pxNewQueue = ( xQUEUE * ) pvPortMalloc( sizeof( xQUEUE ) );
00259         if( pxNewQueue != NULL )
00260         {
00261             /* Create the list of pointers to queue items. The queue is one byte
00262             longer than asked for to make wrap checking easier/faster. */
00263             xQueueSizeInBytes = ( size_t ) ( uxQueueLength * uxItemSize ) + ( size_t ) 1;
00264
00265             pxNewQueue->pcHead = ( signed char * ) pvPortMalloc( xQueueSizeInBytes );
00266             if( pxNewQueue->pcHead != NULL )
00267             {
00268                 /* Initialise the queue members as described above where the
00269                 queue type is defined. */
00270                 pxNewQueue->pcTail = pxNewQueue->pcHead + ( uxQueueLength * uxItemSize );
00271                 pxNewQueue->uxMessagesWaiting = ( unsigned portBASE_TYPE ) 0U;
00272                 pxNewQueue->pcWriteTo = pxNewQueue->pcHead;
00273                 pxNewQueue->pcReadFrom = pxNewQueue->pcHead + ( ( uxQueueLength - ( unsigned
00274                     portBASE_TYPE ) 1U ) * uxItemSize );
00275                 pxNewQueue->uxLength = uxQueueLength;
00276                 pxNewQueue->uxItemSize = uxItemSize;
00277                 pxNewQueue->xRxLock = queueUNLOCKED;
00278                 pxNewQueue->xTxLock = queueUNLOCKED;
00279
00280                 /* Likewise ensure the event queues start with the correct state. */
00281                 vListInitialise( &( pxNewQueue->xTasksWaitingToSend ) );
00282                 vListInitialise( &( pxNewQueue->xTasksWaitingToReceive ) );
00283
00284                 traceQUEUE_CREATE( pxNewQueue );
00285                 xReturn = pxNewQueue;
00286             }
00287             else
00288             {
00289                 traceQUEUE_CREATE_FAILED();
00290                 vPortFree( pxNewQueue );
00291             }
00292         }
00293         configASSERT( xReturn );
00294     }
00295     return xReturn;
00296 }
00297 /*****
00298 */
00299
00300 #if ( configUSE_MUTEXES == 1 )
00301
00302     xQueueHandle xQueueCreateMutex( void )
00303     {
00304         xQUEUE *pxNewQueue;
00305
00306         /* Allocate the new queue structure. */
00307         pxNewQueue = ( xQUEUE * ) pvPortMalloc( sizeof( xQUEUE ) );

```

```

00308     if( pxNewQueue != NULL )
00309     {
00310         /* Information required for priority inheritance. */
00311         pxNewQueue->pxMutexHolder = NULL;
00312         pxNewQueue->uxQueueType = queueQUEUE_IS_MUTEX;
00313
00314         /* Queues used as a mutex no data is actually copied into or out
00315         of the queue. */
00316         pxNewQueue->pcWriteTo = NULL;
00317         pxNewQueue->pcReadFrom = NULL;
00318
00319         /* Each mutex has a length of 1 (like a binary semaphore) and
00320         an item size of 0 as nothing is actually copied into or out
00321         of the mutex. */
00322         pxNewQueue->uxMessagesWaiting = ( unsigned portBASE_TYPE ) 0U;
00323         pxNewQueue->uxLength = ( unsigned portBASE_TYPE ) 1U;
00324         pxNewQueue->uxItemSize = ( unsigned portBASE_TYPE ) 0U;
00325         pxNewQueue->xRxLock = queueUNLOCKED;
00326         pxNewQueue->xTxLock = queueUNLOCKED;
00327
00328         /* Ensure the event queues start with the correct state. */
00329         vListInitialise( &( pxNewQueue->xTasksWaitingToSend ) );
00330         vListInitialise( &( pxNewQueue->xTasksWaitingToReceive ) );
00331
00332         /* Start with the semaphore in the expected state. */
00333         xQueueGenericSend( pxNewQueue, NULL, ( portTickType ) 0U, queueSEND_TO_BACK );
00334
00335         traceCREATE_MUTEX( pxNewQueue );
00336     }
00337     else
00338     {
00339         traceCREATE_MUTEX_FAILED();
00340     }
00341
00342     configASSERT( pxNewQueue );
00343     return pxNewQueue;
00344 }
00345
00346 #endif /* configUSE_MUTEXES */
00347 /*-----*/
00348
00349 #if configUSE_RECURSIVE_MUTEXES == 1
00350
00351     portBASE_TYPE xQueueGiveMutexRecursive( xQueueHandle pxMutex )
00352     {
00353         portBASE_TYPE xReturn;
00354
00355         configASSERT( pxMutex );
00356
00357         /* If this is the task that holds the mutex then pxMutexHolder will not
00358         change outside of this task. If this task does not hold the mutex then
00359         pxMutexHolder can never coincidentally equal the tasks handle, and as
00360         this is the only condition we are interested in it does not matter if
00361         pxMutexHolder is accessed simultaneously by another task. Therefore no
00362         mutual exclusion is required to test the pxMutexHolder variable. */
00363         if( pxMutex->pxMutexHolder == xTaskGetCurrentTaskHandle() )
00364         {
00365             traceGIVE_MUTEX_RECURSIVE( pxMutex );
00366
00367             /* uxRecursiveCallCount cannot be zero if pxMutexHolder is equal to
00368             the task handle, therefore no underflow check is required. Also,
00369             uxRecursiveCallCount is only modified by the mutex holder, and as
00370             there can only be one, no mutual exclusion is required to modify the
00371             uxRecursiveCallCount member. */
00372             ( pxMutex->uxRecursiveCallCount )--;
00373
00374             /* Have we unwound the call count? */
00375             if( pxMutex->uxRecursiveCallCount == 0 )
00376             {
00377                 /* Return the mutex. This will automatically unblock any other
00378                 task that might be waiting to access the mutex. */
00379                 xQueueGenericSend( pxMutex, NULL, queueMutex_GIVE_BLOCK_TIME, queueSEND_TO_BACK );
00380             }
00381
00382             xReturn = pdPASS;
00383         }
00384     else
00385     {
00386         /* We cannot give the mutex because we are not the holder. */
00387         xReturn = pdFAIL;
00388
00389         traceGIVE_MUTEX_RECURSIVE_FAILED( pxMutex );
00390     }
00391
00392     return xReturn;
00393 }
00394

```

```

00395 #endif /* configUSE_RECURSIVE_MUTEXES */
00396 /*-----*/
00397
00398 #if configUSE_RECURSIVE_MUTEXES == 1
00399
00400     portBASE_TYPE xQueueTakeMutexRecursive( xQueueHandle pxMutex, portTickType xBlockTime )
00401     {
00402         portBASE_TYPE xReturn;
00403
00404         configASSERT( pxMutex );
00405
00406         /* Comments regarding mutual exclusion as per those within
00407         xQueueGiveMutexRecursive(). */
00408
00409         traceTAKE_MUTEX_RECURSIVE( pxMutex );
00410
00411         if( pxMutex->pxMutexHolder == xTaskGetCurrentTaskHandle() )
00412         {
00413             ( pxMutex->uxRecursiveCallCount )++;
00414             xReturn = pdPASS;
00415         }
00416         else
00417         {
00418             xReturn = xQueueGenericReceive( pxMutex, NULL, xBlockTime, pdFALSE );
00419
00420             /* pdPASS will only be returned if we successfully obtained the mutex,
00421             we may have blocked to reach here. */
00422             if( xReturn == pdPASS )
00423             {
00424                 ( pxMutex->uxRecursiveCallCount )++;
00425             }
00426             else
00427             {
00428                 traceTAKE_MUTEX_RECURSIVE_FAILED( pxMutex );
00429             }
00430         }
00431
00432         return xReturn;
00433     }
00434
00435 #endif /* configUSE_RECURSIVE_MUTEXES */
00436 /*-----*/
00437
00438 #if configUSE_COUNTING_SEMAPHORES == 1
00439
00440     xQueueHandle xQueueCreateCountingSemaphore( unsigned portBASE_TYPE uxCountValue, unsigned
00441         portBASE_TYPE uxInitialCount )
00442     {
00443         xQueueHandle pxHandle;
00444
00445         pxHandle = xQueueCreate( ( unsigned portBASE_TYPE ) uxCountValue,
00446             queueSEMAPHORE_QUEUE_ITEM_LENGTH );
00447
00448         if( pxHandle != NULL )
00449         {
00450             pxHandle->uxMessagesWaiting = uxInitialCount;
00451             traceCREATE_COUNTING_SEMAPHORE();
00452         }
00453         else
00454         {
00455             traceCREATE_COUNTING_SEMAPHORE_FAILED();
00456         }
00457
00458         configASSERT( pxHandle );
00459         return pxHandle;
00460     }
00461
00462 #endif /* configUSE_COUNTING_SEMAPHORES */
00463 /*-----*/
00464 signed portBASE_TYPE xQueueGenericSend( xQueueHandle pxQueue, const void * const pvItemToQueue,
00465     portTickType xTicksToWait, portBASE_TYPE xCopyPosition )
00466 {
00467     signed portBASE_TYPE xEntryTimeSet = pdFALSE;
00468     xTimeOutType xTimeOut;
00469
00470     configASSERT( pxQueue );
00471     configASSERT( !( ( pvItemToQueue == NULL ) && ( pxQueue->uxItemSize != ( unsigned portBASE_TYPE )
00472         0U ) ) );
00473
00474     /* This function relaxes the coding standard somewhat to allow return
00475     statements within the function itself. This is done in the interest
00476     of execution time efficiency. */
00477     for( ; ; )
00478     {
00479         taskENTER_CRITICAL();

```

```

00478      {
00479          /* Is there room on the queue now? To be running we must be
00480          the highest priority task wanting to access the queue. */
00481          if( pxQueue->uxMessagesWaiting < pxQueue->uxLength )
00482          {
00483              traceQUEUE_SEND( pxQueue );
00484              prvCopyDataToQueue( pxQueue, pvItemToQueue, xCopyPosition );
00485
00486              /* If there was a task waiting for data to arrive on the
00487              queue then unblock it now. */
00488              if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToReceive ) ) == pdFALSE )
00489              {
00490                  if( xTaskRemoveFromEventList( &( pxQueue->xTasksWaitingToReceive ) ) == pdTRUE )
00491                  {
00492                      /* The unblocked task has a priority higher than
00493                      our own so yield immediately. Yes it is ok to do
00494                      this from within the critical section - the kernel
00495                      takes care of that. */
00496                      portYIELD_WITHIN_API();
00497                  }
00498              }
00499
00500              taskEXIT_CRITICAL();
00501
00502              /* Return to the original privilege level before exiting the
00503              function. */
00504              return pdPASS;
00505
00506          }  

00507          else
00508          {
00509              if( xTicksToWait == ( portTickType ) 0 )
00510              {
00511                  /* The queue was full and no block time is specified (or
00512                  the block time has expired) so leave now. */
00513                  taskEXIT_CRITICAL();
00514
00515                  /* Return to the original privilege level before exiting
00516                  the function. */
00517                  traceQUEUE_SEND_FAILED( pxQueue );
00518                  return errQUEUE_FULL;
00519              }
00520              else if( xEntryTimeSet == pdFALSE )
00521              {
00522                  /* The queue was full and a block time was specified so
00523                  configure the timeout structure. */
00524                  vTaskSetTimeOutState( &xTimeOut );
00525                  xEntryTimeSet = pdTRUE;
00526              }
00527          }
00528          taskEXIT_CRITICAL();
00529
00530          /* Interrupts and other tasks can send to and receive from the queue
00531          now the critical section has been exited. */
00532
00533          vTaskSuspendAll();
00534          prvLockQueue( pxQueue );
00535
00536          /* Update the timeout state to see if it has expired yet. */
00537          if( xTaskCheckForTimeOut( &xTimeOut, &xTicksToWait ) == pdFALSE )
00538          {
00539              if( prvIsQueueFull( pxQueue ) != pdFALSE )
00540              {
00541                  traceBLOCKING_ON_QUEUE_SEND( pxQueue );
00542                  vTaskPlaceOnEventList( &( pxQueue->xTasksWaitingToSend ), xTicksToWait );
00543
00544                  /* Unlocking the queue means queue events can effect the
00545                  event list. It is possible that interrupts occurring now
00546                  remove this task from the event list again - but as the
00547                  scheduler is suspended the task will go onto the pending
00548                  ready last instead of the actual ready list. */
00549                  prvUnlockQueue( pxQueue );
00550
00551                  /* Resuming the scheduler will move tasks from the pending
00552                  ready list into the ready list - so it is feasible that this
00553                  task is already in a ready list before it yields - in which
00554                  case the yield will not cause a context switch unless there
00555                  is also a higher priority task in the pending ready list. */
00556                  if( xTaskResumeAll() == pdFALSE )
00557                  {
00558                      portYIELD_WITHIN_API();
00559                  }
00560
00561          }  

00562          else
00563          {
00564              /* Try again. */
00565              prvUnlockQueue( pxQueue );

```

```

00565             ( void ) xTaskResumeAll();
00566         }
00567     }
00568     else
00569     {
00570         /* The timeout has expired. */
00571         prvUnlockQueue( pxQueue );
00572         ( void ) xTaskResumeAll();
00573
00574         /* Return to the original privilege level before exiting the
00575         function. */
00576         traceQUEUE_SEND_FAILED( pxQueue );
00577         return errQUEUE_FULL;
00578     }
00579 }
00580 */
00581 /*-----*/
00582
00583 #if configUSE_ALTERNATIVE_API == 1
00584
00585     signed portBASE_TYPE xQueueAltGenericSend( xQueueHandle pxQueue, const void * const pvItemToQueue,
00586     portTickType xTicksToWait, portBASE_TYPE xCopyPosition )
00587     {
00588         signed portBASE_TYPE xEntryTimeSet = pdFALSE;
00589         xTimeOutType xTimeOut;
00590
00591         configASSERT( pxQueue );
00592         configASSERT( !( ( pvItemToQueue == NULL ) && ( pxQueue->uxItemSize != ( unsigned
00593             portBASE_TYPE ) 0U ) ) );
00594
00595         for( ; ; )
00596         {
00597             taskENTER_CRITICAL();
00598
00599             /* Is there room on the queue now? To be running we must be
00600             the highest priority task wanting to access the queue. */
00601             if( pxQueue->uxMessagesWaiting < pxQueue->uxLength )
00602             {
00603                 traceQUEUE_SEND( pxQueue );
00604                 prvCopyDataToQueue( pxQueue, pvItemToQueue, xCopyPosition );
00605
00606                 /* If there was a task waiting for data to arrive on the
00607                 queue then unblock it now. */
00608                 if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToReceive ) ) == pdFALSE )
00609                 {
00610                     if( xTaskRemoveFromEventList( &( pxQueue->xTasksWaitingToReceive ) ) == pdTRUE
00611
00612                     {
00613                         /* The unblocked task has a priority higher than
00614                         our own so yield immediately. */
00615                         portYIELD_WITHIN_API();
00616
00617                     }
00618
00619                 }
00620             }
00621             taskEXIT_CRITICAL();
00622             return pdPASS;
00623         }
00624     }
00625     else
00626     {
00627         if( xTicksToWait == ( portTickType ) 0 )
00628         {
00629             taskEXIT_CRITICAL();
00630             return errQUEUE_FULL;
00631         }
00632     }
00633     taskEXIT_CRITICAL();
00634
00635     taskENTER_CRITICAL();
00636
00637     if( xTaskCheckForTimeOut( &xTimeOut, &xTicksToWait ) == pdFALSE )
00638     {
00639         if( prvIsQueueFull( pxQueue ) != pdFALSE )
00640         {
00641             traceBLOCKING_ON_QUEUE_SEND( pxQueue );
00642             vTaskPlaceOnEventList( &( pxQueue->xTasksWaitingToSend ), xTicksToWait );
00643             portYIELD_WITHIN_API();
00644         }
00645     }
00646     else
00647     {
00648         taskEXIT_CRITICAL();

```

```

00649             traceQUEUE_SEND_FAILED( pxQueue );
00650             return errQUEUE_FULL;
00651         }
00652     }
00653     taskEXIT_CRITICAL();
00654 }
00655 }
00656
00657 #endif /* configUSE_ALTERNATIVE_API */
00658 /*-----*/
00659
00660 #if configUSE_ALTERNATIVE_API == 1
00661
00662     signed portBASE_TYPE xQueueAltGenericReceive( xQueueHandle pxQueue, void * const pvBuffer,
00663             portTickType xTicksToWait, portBASE_TYPE xJustPeeking )
00664     {
00665         signed portBASE_TYPE xEntryTimeSet = pdFALSE;
00666         xTimeOutType xTimeOut;
00667         signed char *pcOriginalReadPosition;
00668
00669         configASSERT( pxQueue );
00670         configASSERT( !( ( pvBuffer == NULL ) && ( pxQueue->uxItemSize != ( unsigned portBASE_TYPE ) 0U ) ) );
00671         for( ; ; )
00672         {
00673             taskENTER_CRITICAL();
00674             {
00675                 if( pxQueue->uxMessagesWaiting > ( unsigned portBASE_TYPE ) 0 )
00676                 {
00677                     /* Remember our read position in case we are just peeking. */
00678                     pcOriginalReadPosition = pxQueue->pcReadFrom;
00679
00680                     prvCopyDataFromQueue( pxQueue, pvBuffer );
00681
00682                     if( xJustPeeking == pdFALSE )
00683                     {
00684                         traceQUEUE_RECEIVE( pxQueue );
00685
00686                         /* We are actually removing data. */
00687                         --( pxQueue->uxMessagesWaiting );
00688
00689                         #if ( configUSE_MUTEXES == 1 )
00690                         {
00691                             if( pxQueue->uxQueueType == queueQUEUE_IS_MUTEX )
00692                             {
00693                                 /* Record the information required to implement
00694                                 priority inheritance should it become necessary. */
00695                                 pxQueue->pxMutexHolder = xTaskGetCurrentTaskHandle();
00696                             }
00697                         }
00698                         #endif
00699
00700                         if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToSend ) ) == pdFALSE )
00701                         {
00702                             if( xTaskRemoveFromEventList( &( pxQueue->xTasksWaitingToSend ) ) ==
00703                                 pdTRUE )
00704                             {
00705                                 portYIELD_WITHIN_API();
00706                             }
00707                         }
00708                         else
00709                         {
00710                             traceQUEUE_PEEK( pxQueue );
00711
00712                             /* We are not removing the data, so reset our read
00713                             pointer. */
00714                             pxQueue->pcReadFrom = pcOriginalReadPosition;
00715
00716                             /* The data is being left in the queue, so see if there are
00717                             any other tasks waiting for the data. */
00718                             if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToReceive ) ) == pdFALSE )
00719                             {
00720                                 /* Tasks that are removed from the event list will get added to
00721                                 the pending ready list as the scheduler is still suspended. */
00722                                 if( xTaskRemoveFromEventList( &( pxQueue->xTasksWaitingToReceive ) ) !=
00723                                     pdFALSE )
00724                                     {
00725                                         /* The task waiting has a higher priority than this task. */
00726                                         portYIELD_WITHIN_API();
00727                                     }
00728                             }
00729                         }
00730
00731             taskEXIT_CRITICAL();

```

```

00732         return pdPASS;
00733     }
00734     else
00735     {
00736         if( xTicksToWait == ( portTickType ) 0 )
00737         {
00738             taskEXIT_CRITICAL();
00739             traceQUEUE_RECEIVE_FAILED( pxQueue );
00740             return errQUEUE_EMPTY;
00741         }
00742         else if( xEntryTimeSet == pdFALSE )
00743         {
00744             vTaskSetTimeOutState( &xTimeOut );
00745             xEntryTimeSet = pdTRUE;
00746         }
00747     }
00748 }
00749 taskEXIT_CRITICAL();
00750
00751 taskENTER_CRITICAL();
00752 {
00753     if( xTaskCheckForTimeOut( &xTimeOut, &xTicksToWait ) == pdFALSE )
00754     {
00755         if( prvIsQueueEmpty( pxQueue ) != pdFALSE )
00756         {
00757             traceBLOCKING_ON_QUEUE_RECEIVE( pxQueue );
00758
00759             #if ( configUSE_MUTEXES == 1 )
00760             {
00761                 if( pxQueue->uxQueueType == queueQUEUE_IS_MUTEX )
00762                 {
00763                     portENTER_CRITICAL();
00764                     vTaskPriorityInherit( ( void * ) pxQueue->pMutexHolder );
00765                     portEXIT_CRITICAL();
00766                 }
00767             }
00768         #endif
00769
00770         vTaskPlaceOnEventList( &( pxQueue->xTasksWaitingToReceive ), xTicksToWait );
00771         portYIELD_WITHIN_API();
00772     }
00773     else
00774     {
00775         taskEXIT_CRITICAL();
00776         traceQUEUE_RECEIVE_FAILED( pxQueue );
00777         return errQUEUE_EMPTY;
00778     }
00779 }
00780 }
00781 taskEXIT_CRITICAL();
00782 }
00783 }
00784
00785 #endif /* configUSE_ALTERNATIVE_API */
00786 /***** Generic API Implementation *****/
00787
00788
00789 signed portBASE_TYPE xQueueGenericSendFromISR( xQueueHandle pxQueue, const void * const pvItemToQueue,
00790         signed portBASE_TYPE *pxHigherPriorityTaskWoken, portBASE_TYPE xCopyPosition )
00791 {
00792     signed portBASE_TYPE xReturn;
00793     unsigned portBASE_TYPE uxSavedInterruptStatus;
00794     configASSERT( pxQueue );
00795     configASSERT( pxHigherPriorityTaskWoken );
00796     configASSERT( !( ( pvItemToQueue == NULL ) && ( pxQueue->uxItemSize != ( unsigned portBASE_TYPE ) 0U ) ) );
00797
00798     /* Similar to xQueueGenericSend, except we don't block if there is no room
00799     in the queue. Also we don't directly wake a task that was blocked on a
00800     queue read, instead we return a flag to say whether a context switch is
00801     required or not (i.e. has a task with a higher priority than us been woken
00802     by this post). */
00803     uxSavedInterruptStatus = portSET_INTERRUPT_MASK_FROM_ISR();
00804     {
00805         if( pxQueue->uxMessagesWaiting < pxQueue->uxLength )
00806         {
00807             traceQUEUE_SEND_FROM_ISR( pxQueue );
00808
00809             prvCopyDataToQueue( pxQueue, pvItemToQueue, xCopyPosition );
00810
00811             /* If the queue is locked we do not alter the event list. This will
00812             be done when the queue is unlocked later. */
00813             if( pxQueue->xTxLock == queueUNLOCKED )
00814             {
00815                 if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToReceive ) ) == pdFALSE )
00816                 {

```

```

00817             if( xTaskRemoveFromEventList( &( pxQueue->xTasksWaitingToReceive ) ) != pdFALSE )
00818             {
00819                 /* The task waiting has a higher priority so record that a
00820                 context switch is required. */
00821                 *pxHigherPriorityTaskWoken = pdTRUE;
00822             }
00823         }
00824     }
00825     else
00826     {
00827         /* Increment the lock count so the task that unlocks the queue
00828         knows that data was posted while it was locked. */
00829         ++( pxQueue->xTxLock );
00830     }
00831
00832     xReturn = pdPASS;
00833 }
00834 else
00835 {
00836     traceQUEUE_SEND_FROM_ISR_FAILED( pxQueue );
00837     xReturn = errQUEUE_FULL;
00838 }
00839 }
00840 portCLEAR_INTERRUPT_MASK_FROM_ISR( uxSavedInterruptStatus );
00841
00842 return xReturn;
00843 }
00844 /*****
```

```

00845
00846 signed portBASE_TYPE xQueueGenericReceive( xQueueHandle pxQueue, void * const pvBuffer, portTickType
00847     xTicksToWait, portBASE_TYPE xJustPeeking )
00848 {
00849     signed portBASE_TYPE xEntryTimeSet = pdFALSE;
00850     xTimeOutType xTimeOut;
00851     signed char *pcOriginalReadPosition;
00852
00853     configASSERT( pxQueue );
00854     configASSERT( !( ( pvBuffer == NULL ) && ( pxQueue->uxItemSize != ( unsigned portBASE_TYPE ) 0U ) ) );
00855
00856     /* This function relaxes the coding standard somewhat to allow return
00857     statements within the function itself. This is done in the interest
00858     of execution time efficiency. */
00859
00860     for( ; ; )
00861     {
00862         taskENTER_CRITICAL();
00863         {
00864             /* Is there data in the queue now? To be running we must be
00865             the highest priority task wanting to access the queue. */
00866             if( pxQueue->uxMessagesWaiting > ( unsigned portBASE_TYPE ) 0 )
00867             {
00868                 /* Remember our read position in case we are just peeking. */
00869                 pcOriginalReadPosition = pxQueue->pcReadFrom;
00870
00871                 prvCopyDataFromQueue( pxQueue, pvBuffer );
00872
00873                 if( xJustPeeking == pdFALSE )
00874                 {
00875                     traceQUEUE_RECEIVE( pxQueue );
00876
00877                     /* We are actually removing data. */
00878                     --( pxQueue->uxMessagesWaiting );
00879
00880                     #if ( configUSE_MUTEXES == 1 )
00881                     {
00882                         if( pxQueue->uxQueueType == queueQUEUE_IS_MUTEX )
00883                         {
00884                             /* Record the information required to implement
00885                             priority inheritance should it become necessary. */
00886                             pxQueue->pxMutexHolder = xTaskGetCurrentTaskHandle();
00887                         }
00888                     }
00889                     #endif
00890
00891                     if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToSend ) ) == pdFALSE )
00892                     {
00893                         if( xTaskRemoveFromEventList( &( pxQueue->xTasksWaitingToSend ) ) == pdTRUE )
00894                         {
00895                             portYIELD_WITHIN_API();
00896                         }
00897                     }
00898                 }
00899             }
00900             traceQUEUE_PEEK( pxQueue );
00901 }
```

```

00902     /* We are not removing the data, so reset our read
00903     pointer. */
00904     pxQueue->pcReadFrom = pcOriginalReadPosition;
00905
00906     /* The data is being left in the queue, so see if there are
00907     any other tasks waiting for the data. */
00908     if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToReceive ) ) == pdFALSE )
00909     {
00910         /* Tasks that are removed from the event list will get added to
00911         the pending ready list as the scheduler is still suspended. */
00912         if( xTaskRemoveFromEventList( &( pxQueue->xTasksWaitingToReceive ) ) !=
00913             pdFALSE )
00914         {
00915             /* The task waiting has a higher priority than this task. */
00916             portYIELD_WITHIN_API();
00917         }
00918     }
00919
00920     taskEXIT_CRITICAL();
00921     return pdPASS;
00922 }
00923 else
00924 {
00925     if( xTicksToWait == ( portTickType ) 0 )
00926     {
00927         /* The queue was empty and no block time is specified (or
00928         the block time has expired) so leave now. */
00929         taskEXIT_CRITICAL();
00930         traceQUEUE_RECEIVE_FAILED( pxQueue );
00931         return errQUEUE_EMPTY;
00932     }
00933     else if( xEntryTimeSet == pdFALSE )
00934     {
00935         /* The queue was empty and a block time was specified so
00936         configure the timeout structure. */
00937         vTaskSetTimeOutState( &xTimeOut );
00938         xEntryTimeSet = pdTRUE;
00939     }
00940 }
00941 }
00942 }
00943 taskEXIT_CRITICAL();
00944
00945 /* Interrupts and other tasks can send to and receive from the queue
00946 now the critical section has been exited. */
00947
00948 vTaskSuspendAll();
00949 prvLockQueue( pxQueue );
00950
00951 /* Update the timeout state to see if it has expired yet. */
00952 if( xTaskCheckForTimeOut( &xTimeOut, &xTicksToWait ) == pdFALSE )
00953 {
00954     if( prvIsQueueEmpty( pxQueue ) != pdFALSE )
00955     {
00956         traceBLOCKING_ON_QUEUE_RECEIVE( pxQueue );
00957
00958         #if ( configUSE_MUTEXES == 1 )
00959         {
00960             if( pxQueue->uxQueueType == queueQUEUE_IS_MUTEX )
00961             {
00962                 portENTER_CRITICAL();
00963                 {
00964                     vTaskPriorityInherit( ( void * ) pxQueue->pxMutexHolder );
00965                 }
00966                 portEXIT_CRITICAL();
00967             }
00968         }
00969     #endif
00970
00971     vTaskPlaceOnEventList( &( pxQueue->xTasksWaitingToReceive ), xTicksToWait );
00972     prvUnlockQueue( pxQueue );
00973     if( xTaskResumeAll() == pdFALSE )
00974     {
00975         portYIELD_WITHIN_API();
00976     }
00977 }
00978 else
00979 {
00980     /* Try again. */
00981     prvUnlockQueue( pxQueue );
00982     ( void ) xTaskResumeAll();
00983 }
00984 else
00985 {
00986     prvUnlockQueue( pxQueue );

```

```

00988     ( void ) xTaskResumeAll();
00989     traceQUEUE_RECEIVE_FAILED( pxQueue );
00990     return errQUEUE_EMPTY;
00991 }
00992 }
00993 }
00994 /*-----*/
00995
00996 signed portBASE_TYPE xQueueReceiveFromISR( xQueueHandle pxQueue, void * const pvBuffer, signed
00997 portBASE_TYPE *pxTaskWoken )
00998 {
00999     signed portBASE_TYPE xReturn;
01000     unsigned portBASE_TYPE uxSavedInterruptStatus;
01001     configASSERT( pxQueue );
01002     configASSERT( pxTaskWoken );
01003     configASSERT( !( ( pvBuffer == NULL ) && ( pxQueue->uxItemSize != ( unsigned portBASE_TYPE ) 0U ) )
01004 );
01005     uxSavedInterruptStatus = portSET_INTERRUPT_MASK_FROM_ISR();
01006     {
01007         /* We cannot block from an ISR, so check there is data available. */
01008         if( pxQueue->uxMessagesWaiting > ( unsigned portBASE_TYPE ) 0 )
01009         {
01010             traceQUEUE_RECEIVE_FROM_ISR( pxQueue );
01011
01012             prvCopyDataFromQueue( pxQueue, pvBuffer );
01013             --( pxQueue->uxMessagesWaiting );
01014
01015             /* If the queue is locked we will not modify the event list. Instead
01016             we update the lock count so the task that unlocks the queue will know
01017             that an ISR has removed data while the queue was locked. */
01018             if( pxQueue->xRxLock == queueUNLOCKED )
01019             {
01020                 if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToSend ) ) == pdFALSE )
01021                 {
01022                     if( xTaskRemoveFromEventList( &( pxQueue->xTasksWaitingToSend ) ) != pdFALSE )
01023                     {
01024                         /* The task waiting has a higher priority than us so
01025                         force a context switch. */
01026                         *pxTaskWoken = pdTRUE;
01027                     }
01028                 }
01029             }
01030         }
01031     }
01032     /* Increment the lock count so the task that unlocks the queue
01033     knows that data was removed while it was locked. */
01034     +( pxQueue->xRxLock );
01035
01036     xReturn = pdPASS;
01037 }
01038 else
01039 {
01040     xReturn = pdFAIL;
01041     traceQUEUE_RECEIVE_FROM_ISR FAILED( pxQueue );
01042 }
01043 }
01044 }
01045 portCLEAR_INTERRUPT_MASK_FROM_ISR( uxSavedInterruptStatus );
01046
01047 return xReturn;
01048 }
01049 /*-----*/
01050
01051 unsigned portBASE_TYPE uxQueueMessagesWaiting( const xQueueHandle pxQueue )
01052 {
01053     unsigned portBASE_TYPE uxReturn;
01054
01055     configASSERT( pxQueue );
01056
01057     taskENTER_CRITICAL();
01058     uxReturn = pxQueue->uxMessagesWaiting;
01059     taskEXIT_CRITICAL();
01060
01061     return uxReturn;
01062 }
01063 /*-----*/
01064
01065 unsigned portBASE_TYPE uxQueueMessagesWaitingFromISR( const xQueueHandle pxQueue )
01066 {
01067     unsigned portBASE_TYPE uxReturn;
01068
01069     configASSERT( pxQueue );
01070
01071     uxReturn = pxQueue->uxMessagesWaiting;
01072

```

```

01073     return uxReturn;
01074 }
01075 /*-----*/
01076
01077 void vQueueDelete( xQueueHandle pxQueue )
01078 {
01079     configASSERT( pxQueue );
01080
01081     traceQUEUE_DELETE( pxQueue );
01082     vQueueUnregisterQueue( pxQueue );
01083     vPortFree( pxQueue->pcHead );
01084     vPortFree( pxQueue );
01085 }
01086 /*-----*/
01087
01088 static void prvCopyDataToQueue( xQUEUE *pxQueue, const void *pvItemToQueue, portBASE_TYPE xPosition )
01089 {
01090     if( pxQueue->uxItemSize == ( unsigned ) portBASE_TYPE ) 0 )
01091     {
01092         #if ( configUSE_MUTEXES == 1 )
01093         {
01094             if( pxQueue->uxQueueType == queueQUEUE_IS_MUTEX )
01095             {
01096                 /* The mutex is no longer being held. */
01097                 vTaskPriorityDisinherit( ( void * ) pxQueue->pxMutexHolder );
01098                 pxQueue->pxMutexHolder = NULL;
01099             }
01100         }
01101         #endif
01102     }
01103     else if( xPosition == queueSEND_TO_BACK )
01104     {
01105         memcpy( ( void * ) pxQueue->pcWriteTo, pvItemToQueue, ( unsigned ) pxQueue->uxItemSize );
01106         pxQueue->pcWriteTo += pxQueue->uxItemSize;
01107         if( pxQueue->pcWriteTo >= pxQueue->pcTail )
01108         {
01109             pxQueue->pcWriteTo = pxQueue->pcHead;
01110         }
01111     }
01112     else
01113     {
01114         memcpy( ( void * ) pxQueue->pcReadFrom, pvItemToQueue, ( unsigned ) pxQueue->uxItemSize );
01115         pxQueue->pcReadFrom -= pxQueue->uxItemSize;
01116         if( pxQueue->pcReadFrom < pxQueue->pcHead )
01117         {
01118             pxQueue->pcReadFrom = ( pxQueue->pcTail - pxQueue->uxItemSize );
01119         }
01120     }
01121
01122     ++( pxQueue->uxMessagesWaiting );
01123 }
01124 /*-----*/
01125
01126 static void prvCopyDataFromQueue( xQUEUE * const pxQueue, const void *pvBuffer )
01127 {
01128     if( pxQueue->uxQueueType != queueQUEUE_IS_MUTEX )
01129     {
01130         pxQueue->pcReadFrom += pxQueue->uxItemSize;
01131         if( pxQueue->pcReadFrom >= pxQueue->pcTail )
01132         {
01133             pxQueue->pcReadFrom = pxQueue->pcHead;
01134         }
01135         memcpy( ( void * ) pvBuffer, ( void * ) pxQueue->pcReadFrom, ( unsigned ) pxQueue->uxItemSize
01136     );
01137 }
01138 /*-----*/
01139
01140 static void prvUnlockQueue( xQueueHandle pxQueue )
01141 {
01142     /* THIS FUNCTION MUST BE CALLED WITH THE SCHEDULER SUSPENDED. */
01143
01144     /* The lock counts contains the number of extra data items placed or
01145     removed from the queue while the queue was locked. When a queue is
01146     locked items can be added or removed, but the event lists cannot be
01147     updated. */
01148     taskENTER_CRITICAL();
01149     {
01150         /* See if data was added to the queue while it was locked. */
01151         while( pxQueue->xTxLock > queueLOCKED_UNMODIFIED )
01152         {
01153             /* Data was posted while the queue was locked. Are any tasks
01154             blocked waiting for data to become available? */
01155             if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToReceive ) ) == pdFALSE )
01156             {
01157                 /* Tasks that are removed from the event list will get added to
01158                 the pending ready list as the scheduler is still suspended. */
01159             }
01160         }
01161     }
01162 }
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
02109
02110
02111
02112
02113
02114
02115
02116
02117
02118
02119
02120
02121
02122
02123
02124
02125
02126
02127
02128
02129
02130
02131
02132
02133
02134
02135
02136
02137
02138
02139
02140
02141
02142
02143
02144
02145
02146
02147
02148
02149
02150
02151
02152
02153
02154
02155
02156
02157
02158
02159
02160
02161
02162
02163
02164
02165
02166
02167
02168
02169
02170
02171
02172
02173
02174
02175
02176
02177
02178
02179
02180
02181
02182
02183
02184
02185
02186
02187
02188
02189
02190
02191
02192
02193
02194
02195
02196
02197
02198
02199
02200
02201
02202
02203
02204
02205
02206
02207
02208
02209
02210
02211
02212
02213
02214
02215
02216
02217
02218
02219
02220
02221
02222
02223
02224
02225
02226
02227
02228
02229
02230
02231
02232
02233
02234
02235
02236
02237
02238
02239
02240
02241
02242
02243
02244
02245
02246
02247
02248
02249
02250
02251
02252
02253
02254
02255
02256
02257
02258
02259
02260
02261
02262
02263
02264
02265
02266
02267
02268
02269
02270
02271
02272
02273
02274
02275
02276
02277
02278
02279
02280
02281
02282
02283
02284
02285
02286
02287
02288
02289
02290
02291
02292
02293
02294
02295
02296
02297
02298
02299
02300
02301
02302
02303
02304
02305
02306
02307
02308
02309
02310
02311
02312
02313
02314
02315
02316
02317
02318
02319
02320
02321
02322
02323
02324
02325
02326
02327
02328
02329
02330
02331
02332
02333
02334
02335
02336
02337
02338
02339
02340
02341
02342
02343
02344
02345
02346
02347
02348
02349
02350
02351
02352
02353
02354
02355
02356
02357
02358
02359
02360
02361
02362
02363
02364
02365
02366
02367
02368
02369
02370
02371
02372
02373
02374
02375
02376
02377
02378
02379
02380
02381
02382
02383
02384
02385
02386
02387
02388
02389
02390
02391
02392
02393
02394
02395
02396
02397
02398
02399
02400
02401
02402
02403
02404
02405
02406
02407
02408
02409
02410
02411
02412
02413
02414
02415
02416
02417
02418
02419
02420
02421
02422
02423
02424
02425
02426
02427
02428
02429
02430
02431
02432
02433
02434
02435
02436
02437
02438
02439
02440
02441
02442
02443
02444
02445
02446
02447
02448
02449
02450
02451
02452
02453
02454
02455
02456
02457
02458
02459
02460
02461
02462
02463
02464
02465
02466
02467
02468
02469
02470
02471
02472
02473
02474
02475
02476
02477
02478
02479
02480
02481
02482
02483
02484
02485
02486
02487
02488
02489
02490
02491
02492
02493
02494
02495
02496
02497
02498
02499
02500
02501
02502
02503
02504
02505
02506
02507
02508
02509
02510
02511
02512
02513
02514
02515
02516
02517
02518
02519
02520
02521
02522
02523
02524
02525
02526
02527
02528
02529
02530
02531
02532
02533
02534
02535
02536
02537
02538
02539
02540
02541
02542
02543
02544
02545
02546
02547
02548
02549
02550
02551
02552
02553
02554
02555
02556
02557
02558
02559
02560
02561
02562
02563
02564
02565
02566
02567
02568
02569
02570
02571
02572
02573
02574
02575
02576
02577
02578
02579
02580
02581
02582
02583
02584
02585
02586
02587
02588
02589
02590
02591
02592
02593
02594
02595
02596
02597
02598
02599
02600
02601
02602
02603
02604
02605
02606
02607
02608
02609
02610
02611
02612
02613
02614
02615
02616
02617
02618
02619
02620
02621
02622
02623
02624
02625
02626
02627
02628
02629
02630
02631
02632
02633
02634
02635
02636
02637
02638
02639
02640
02641
02642
02643
02644
02645
02646
02647
02648
02649
02650
02651
02652
02653
02654
02655
02656
02657
02658
02659
02660
02661
02662
02663
02664
02665
02666
02667
02668
02669
02670
02671
02672
02673
02674
02675
02676
02677
02678
02679
02680
02681
02682
02683
02684
02685
02686
02687
02688
02689
02690
02691
02692
02693
02694
02695
02696
02697
02698
02699
02700
02701
02702
02703
02704
02705
02706
02707
02708
02709
02710
02711
02712
02713
02714
02715
02716
02717
02718
02719
02720
02721
02722
02723
02724
02725
02726
02727
02728
02729
02730
02731
02732
02733
02734
02735
02736
02737
02738
02739
02740
02741
02742
02743
02744
02745
02746
02747
02748
02749
02750
02751
02752
02753
02754
02755
02756
02757
02758
02759
02760
02761
02762
02763
02764
02765
02766
02767
02768
02769
02770
02771
02772
02773
02774
02775
02776
02777
02778
02779
02780
02781
02782
02783
02784
02785
02786
02787
02788
02789
02790
02791
02792
02793
02794
02795
02796
02797
02798
02799
02800
02801
02802
02803
02804
0
```

```

01159         if( xTaskRemoveFromEventList( &( pxQueue->xTasksWaitingToReceive ) ) != pdFALSE )
01160     {
01161         /* The task waiting has a higher priority so record that a
01162         context switch is required. */
01163         vTaskMissedYield();
01164     }
01165
01166     --( pxQueue->xTxLock );
01167 }
01168 else
01169 {
01170     break;
01171 }
01173
01174     pxQueue->xTxLock = queueUNLOCKED;
01175 }
01176 taskEXIT_CRITICAL();
01177
01178 /* Do the same for the Rx lock. */
01179 taskENTER_CRITICAL();
01180 {
01181     while( pxQueue->xRxLock > queueLOCKED_UNMODIFIED )
01182     {
01183         if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToSend ) ) == pdFALSE )
01184         {
01185             if( xTaskRemoveFromEventList( &( pxQueue->xTasksWaitingToSend ) ) != pdFALSE )
01186             {
01187                 vTaskMissedYield();
01188             }
01189
01190             --( pxQueue->xRxLock );
01191         }
01192         else
01193         {
01194             break;
01195         }
01196     }
01197
01198     pxQueue->xRxLock = queueUNLOCKED;
01199 }
01200 taskEXIT_CRITICAL();
01201 }
01202 /*****  

01203
01204 static signed portBASE_TYPE prvIsQueueEmpty( const xQueueHandle pxQueue )
01205 {
01206     signed portBASE_TYPE xReturn;
01207
01208     taskENTER_CRITICAL();
01209     xReturn = ( pxQueue->uxMessagesWaiting == ( unsigned portBASE_TYPE ) 0 );
01210     taskEXIT_CRITICAL();
01211
01212     return xReturn;
01213 }
01214 /*****  

01215
01216 signed portBASE_TYPE xQueueIsQueueEmptyFromISR( const xQueueHandle pxQueue )
01217 {
01218     signed portBASE_TYPE xReturn;
01219
01220     configASSERT( pxQueue );
01221     xReturn = ( pxQueue->uxMessagesWaiting == ( unsigned portBASE_TYPE ) 0 );
01222
01223     return xReturn;
01224 }
01225 /*****  

01226
01227 static signed portBASE_TYPE prvIsQueueFull( const xQueueHandle pxQueue )
01228 {
01229     signed portBASE_TYPE xReturn;
01230
01231     taskENTER_CRITICAL();
01232     xReturn = ( pxQueue->uxMessagesWaiting == pxQueue->uxLength );
01233     taskEXIT_CRITICAL();
01234
01235     return xReturn;
01236 }
01237 /*****  

01238
01239 signed portBASE_TYPE xQueueIsQueueFullFromISR( const xQueueHandle pxQueue )
01240 {
01241     signed portBASE_TYPE xReturn;
01242
01243     configASSERT( pxQueue );
01244     xReturn = ( pxQueue->uxMessagesWaiting == pxQueue->uxLength );
01245

```

```

01246     return xReturn;
01247 }
01248 /*-----*/
01249
01250 #if configUSE_CO_ROUTINES == 1
01251 signed portBASE_TYPE xQueueCRSend( xQueueHandle pxQueue, const void *pvItemToQueue, portTickType
01252 xTicksToWait )
01253 {
01254     signed portBASE_TYPE xReturn;
01255
01256     /* If the queue is already full we may have to block. A critical section
01257     is required to prevent an interrupt removing something from the queue
01258     between the check to see if the queue is full and blocking on the queue. */
01259     portDISABLE_INTERRUPTS();
01260     {
01261         if( prvIsQueueFull( pxQueue ) != pdFALSE )
01262         {
01263             /* The queue is full - do we want to block or just leave without
01264             posting? */
01265             if( xTicksToWait > ( portTickType ) 0 )
01266             {
01267                 /* As this is called from a coroutine we cannot block directly, but
01268                 return indicating that we need to block. */
01269                 vCoRoutineAddToDelayedList( xTicksToWait, &( pxQueue->xTasksWaitingToSend ) );
01270                 portENABLE_INTERRUPTS();
01271                 return errQUEUE_BLOCKED;
01272             }
01273         }
01274         portENABLE_INTERRUPTS();
01275         return errQUEUE_FULL;
01276     }
01277 }
01278 portENABLE_INTERRUPTS();
01279
01280 portNOP();
01281
01282 portDISABLE_INTERRUPTS();
01283 {
01284     if( pxQueue->uxMessagesWaiting < pxQueue->uxLength )
01285     {
01286         /* There is room in the queue, copy the data into the queue. */
01287         prvCopyDataToQueue( pxQueue, pvItemToQueue, queueSEND_TO_BACK );
01288         xReturn = pdPASS;
01289
01290         /* Were any co-routines waiting for data to become available? */
01291         if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToReceive ) ) == pdFALSE )
01292         {
01293             /* In this instance the co-routine could be placed directly
01294             into the ready list as we are within a critical section.
01295             Instead the same pending ready list mechanism is used as if
01296             the event were caused from within an interrupt. */
01297             if( xCoRoutineRemoveFromEventList( &( pxQueue->xTasksWaitingToReceive ) ) != pdFALSE )
01298             {
01299                 /* The co-routine waiting has a higher priority so record
01300                 that a yield might be appropriate. */
01301                 xReturn = errQUEUE_YIELD;
01302             }
01303         }
01304     }
01305 }
01306 else
01307 {
01308     xReturn = errQUEUE_FULL;
01309 }
01310 }
01311 portENABLE_INTERRUPTS();
01312
01313 return xReturn;
01314 }
01315 #endif
01316 /*-----*/
01317
01318 #if configUSE_CO_ROUTINES == 1
01319 signed portBASE_TYPE xQueueCRReceive( xQueueHandle pxQueue, void *pvBuffer, portTickType xTicksToWait
01320 )
01321 {
01322     signed portBASE_TYPE xReturn;
01323
01324     /* If the queue is already empty we may have to block. A critical section
01325     is required to prevent an interrupt adding something to the queue
01326     between the check to see if the queue is empty and blocking on the queue. */
01327     portDISABLE_INTERRUPTS();
01328     {
01329         if( pxQueue->uxMessagesWaiting == ( unsigned portBASE_TYPE ) 0 )
01330         {
01331             /* There are no messages in the queue, do we want to block or just

```

```

01331     leave with nothing? */
01332     if( xTicksToWait > ( portTickType ) 0 )
01333     {
01334         /* As this is a co-routine we cannot block directly, but return
01335            indicating that we need to block. */
01336         vCoRoutineAddToDelayedList( xTicksToWait, &( pxQueue->xTasksWaitingToReceive ) );
01337         portENABLE_INTERRUPTS();
01338         return errQUEUE_BLOCKED;
01339     }
01340     else
01341     {
01342         portENABLE_INTERRUPTS();
01343         return errQUEUE_FULL;
01344     }
01345 }
01346 portENABLE_INTERRUPTS();
01347 portNOP();
01348 portDISABLE_INTERRUPTS();
01349 {
01350     if( pxQueue->uxMessagesWaiting > ( unsigned portBASE_TYPE ) 0 )
01351     {
01352         /* Data is available from the queue. */
01353         pxQueue->pcReadFrom += pxQueue->uxItemSize;
01354         if( pxQueue->pcReadFrom >= pxQueue->pcTail )
01355         {
01356             pxQueue->pcReadFrom = pxQueue->pcHead;
01357         }
01358         --( pxQueue->uxMessagesWaiting );
01359         memcpy( ( void * ) pvBuffer, ( void * ) pxQueue->pcReadFrom, ( unsigned )
01360 pxQueue->uxItemSize );
01361
01362         xReturn = pdPASS;
01363
01364         /* Were any co-routines waiting for space to become available? */
01365         if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToSend ) ) == pdFALSE )
01366         {
01367             /* In this instance the co-routine could be placed directly
01368                into the ready list as we are within a critical section.
01369                Instead the same pending ready list mechanism is used as if
01370                the event were caused from within an interrupt. */
01371             if( xCoRoutineRemoveFromEventList( &( pxQueue->xTasksWaitingToSend ) ) != pdFALSE )
01372             {
01373                 xReturn = errQUEUE_YIELD;
01374             }
01375         }
01376     }
01377     else
01378     {
01379         xReturn = pdFAIL;
01380     }
01381 }
01382 portENABLE_INTERRUPTS();
01383
01384 return xReturn;
01385 }
01386 #endif
01387 /*-----*/
01388
01389
01390
01391
01392
01393 #if configUSE_CO_ROUTINES == 1
01394 signed portBASE_TYPE xQueueCRSendFromISR( xQueueHandle pxQueue, const void *pvItemToQueue, signed
01395 portBASE_TYPE xCoRoutinePreviouslyWoken )
01396 {
01397     /* Cannot block within an ISR so if there is no space on the queue then
01398        exit without doing anything. */
01399     if( pxQueue->uxMessagesWaiting < pxQueue->uxLength )
01400     {
01401         prvCopyDataToQueue( pxQueue, pvItemToQueue, queueSEND_TO_BACK );
01402
01403         /* We only want to wake one co-routine per ISR, so check that a
01404            co-routine has not already been woken. */
01405         if( xCoRoutinePreviouslyWoken == pdFALSE )
01406         {
01407             if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToReceive ) ) == pdFALSE )
01408             {
01409                 if( xCoRoutineRemoveFromEventList( &( pxQueue->xTasksWaitingToReceive ) ) != pdFALSE )
01410                 {
01411                     return pdTRUE;
01412                 }
01413             }
01414         }
01415 }

```

```

01416     return xCoRoutinePreviouslyWoken;
01417 }
01418 #endif
01419 /*-----*/
01420
01421 #if configUSE_CO_ROUTINES == 1
01422 signed portBASE_TYPE xQueueCRReceiveFromISR( xQueueHandle pxQueue, void *pvBuffer, signed
01423   portBASE_TYPE *pxCoRoutineWoken )
01424 {
01425   signed portBASE_TYPE xReturn;
01426
01427   /* We cannot block from an ISR, so check there is data available. If
01428   not then just leave without doing anything. */
01429   if( pxQueue->uxMessagesWaiting > ( unsigned portBASE_TYPE ) 0 )
01430   {
01431     /* Copy the data from the queue. */
01432     pxQueue->pcReadFrom += pxQueue->uxItemSize;
01433     if( pxQueue->pcReadFrom >= pxQueue->pcTail )
01434     {
01435       pxQueue->pcReadFrom = pxQueue->pcHead;
01436     }
01437     --( pxQueue->uxMessagesWaiting );
01438     memcpy( ( void * ) pvBuffer, ( void * ) pxQueue->pcReadFrom, ( unsigned ) pxQueue->uxItemSize
01439   );
01440
01441     if( ( *pxCoRoutineWoken ) == pdFALSE )
01442     {
01443       if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToSend ) ) == pdFALSE )
01444       {
01445         if( xCoRoutineRemoveFromEventList( &( pxQueue->xTasksWaitingToSend ) ) != pdFALSE )
01446         {
01447           *pxCoRoutineWoken = pdTRUE;
01448         }
01449       }
01450     xReturn = pdPASS;
01451   }
01452 else
01453 {
01454   xReturn = pdFAIL;
01455 }
01456
01457 return xReturn;
01458 }
01459#endif
01460 /*-----*/
01461
01462 #if configQUEUE_REGISTRY_SIZE > 0
01463
01464 void vQueueAddToRegistry( xQueueHandle xQueue, signed char *pcQueueName )
01465 {
01466   unsigned portBASE_TYPE ux;
01467
01468   /* See if there is an empty space in the registry. A NULL name denotes
01469   a free slot. */
01470   for( ux = ( unsigned portBASE_TYPE ) 0U; ux < ( unsigned portBASE_TYPE )
configQUEUE_REGISTRY_SIZE; ux++ )
01471   {
01472     if( xQueueRegistry[ ux ].pcQueueName == NULL )
01473     {
01474       /* Store the information on this queue. */
01475       xQueueRegistry[ ux ].pcQueueName = pcQueueName;
01476       xQueueRegistry[ ux ].xHandle = xQueue;
01477       break;
01478     }
01479   }
01480 }
01481
01482#endif
01483 /*-----*/
01484
01485 #if configQUEUE_REGISTRY_SIZE > 0
01486
01487 static void vQueueUnregisterQueue( xQueueHandle xQueue )
01488 {
01489   unsigned portBASE_TYPE ux;
01490
01491   /* See if the handle of the queue being unregistered is actually in the
01492   registry. */
01493   for( ux = ( unsigned portBASE_TYPE ) 0U; ux < ( unsigned portBASE_TYPE )
configQUEUE_REGISTRY_SIZE; ux++ )
01494   {
01495     if( xQueueRegistry[ ux ].xHandle == xQueue )
01496     {
01497       /* Set the name to NULL to show that this slot is free again. */
01498       xQueueRegistry[ ux ].pcQueueName = NULL;
01499     }
01500   }
01501 }
01502
01503#endif

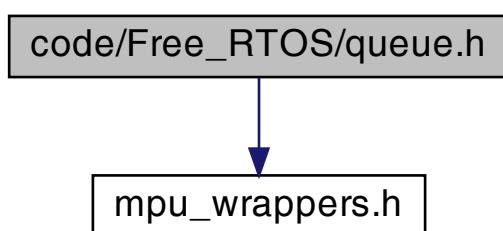
```

```
01499             break;
01500         }
01501     }
01502 }
01503 }
01504
01505 #endif
01506 /*-----*/
01507
01508 #if configUSE_TIMERS == 1
01509
01510     void vQueueWaitForMessageRestricted( xQueueHandle pxQueue, portTickType xTicksToWait )
01511     {
01512         /* This function should not be called by application code hence the
01513         'Restricted' in its name. It is not part of the public API. It is
01514         designed for use by kernel code, and has special calling requirements.
01515         It can result in vListInsert() being called on a list that can only
01516         possibly ever have one item in it, so the list will be fast, but even
01517         so it should be called with the scheduler locked and not from a critical
01518         section. */
01519
01520         /* Only do anything if there are no messages in the queue. This function
01521         will not actually cause the task to block, just place it on a blocked
01522         list. It will not block until the scheduler is unlocked - at which
01523         time a yield will be performed. If an item is added to the queue while
01524         the queue is locked, and the calling task blocks on the queue, then the
01525         calling task will be immediately unblocked when the queue is unlocked. */
01526         prvLockQueue( pxQueue );
01527         if( pxQueue->uxMessagesWaiting == ( unsigned portBASE_TYPE ) 0U )
01528         {
01529             /* There is nothing in the queue, block for the specified period. */
01530             vTaskPlaceOnEventListRestricted( &( pxQueue->xTasksWaitingToReceive ), xTicksToWait );
01531         }
01532         prvUnlockQueue( pxQueue );
01533     }
01534
01535 #endif
01536
```

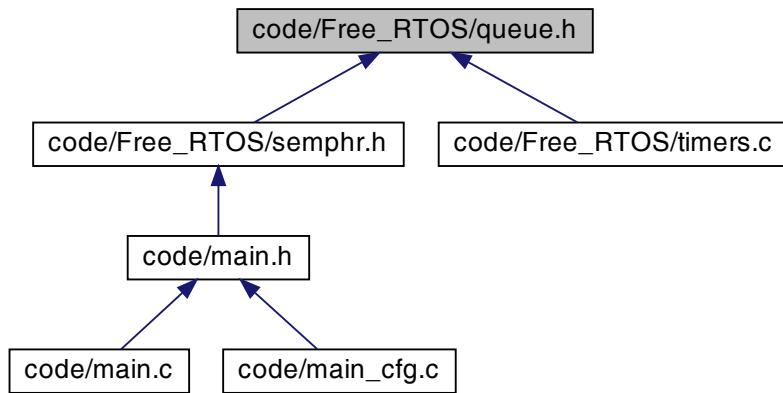
22.29 code/FreeRTOS/queue.h File Reference

#include "mpu_wrappers.h"

Include dependency graph for queue.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define queueSEND_TO_BACK (0)
- #define queueSEND_TO_FRONT (1)
- #define xQueueSendToFront(xQueue, pvItemToQueue, xTicksToWait) xQueueGenericSend((xQueue), (pvItemToQueue), (xTicksToWait), queueSEND_TO_FRONT)
- #define xQueueSendToBack(xQueue, pvItemToQueue, xTicksToWait) xQueueGenericSend((xQueue), (pvItemToQueue), (xTicksToWait), queueSEND_TO_BACK)
- #define xQueueSend(xQueue, pvItemToQueue, xTicksToWait) xQueueGenericSend((xQueue), (pvItemToQueue), (xTicksToWait), queueSEND_TO_BACK)
- #define xQueuePeek(xQueue, pvBuffer, xTicksToWait) xQueueGenericReceive((xQueue), (pvBuffer), (xTicksToWait), pdTRUE)
- #define xQueueReceive(xQueue, pvBuffer, xTicksToWait) xQueueGenericReceive((xQueue), (pvBuffer), (xTicksToWait), pdFALSE)
- #define xQueueSendToFrontFromISR(pxQueue, pvItemToQueue, pxHigherPriorityTaskWoken) xQueueGenericSendFromISR((pxQueue), (pvItemToQueue), (pxHigherPriorityTaskWoken), queueSEND_TO_FRONT)
- #define xQueueSendToBackFromISR(pxQueue, pvItemToQueue, pxHigherPriorityTaskWoken) xQueueGenericSendFromISR((pxQueue), (pvItemToQueue), (pxHigherPriorityTaskWoken), queueSEND_TO_BACK)
- #define xQueueSendFromISR(pxQueue, pvItemToQueue, pxHigherPriorityTaskWoken) xQueueGenericSendFromISR((pxQueue), (pvItemToQueue), (pxHigherPriorityTaskWoken), queueSEND_TO_BACK)
- #define xQueueAltSendToFront(xQueue, pvItemToQueue, xTicksToWait) xQueueAltGenericSend((xQueue), (pvItemToQueue), (xTicksToWait), queueSEND_TO_FRONT)
- #define xQueueAltSendToBack(xQueue, pvItemToQueue, xTicksToWait) xQueueAltGenericSend((xQueue), (pvItemToQueue), (xTicksToWait), queueSEND_TO_BACK)
- #define xQueueAltReceive(xQueue, pvBuffer, xTicksToWait) xQueueAltGenericReceive((xQueue), (pvBuffer), (xTicksToWait), pdFALSE)
- #define xQueueAltPeek(xQueue, pvBuffer, xTicksToWait) xQueueAltGenericReceive((xQueue), (pvBuffer), (xTicksToWait), pdTRUE)

Typedefs

- typedef void * xQueueHandle

Functions

- `xQueueHandle xQueueCreate (unsigned portBASE_TYPE uxQueueLength, unsigned portBASE_TYPE uxItemSize)`
- `signed portBASE_TYPE xQueueGenericSend (xQueueHandle pxQueue, const void *const pvItemToQueue, portTickType xTicksToWait, portBASE_TYPE xCopyPosition)`
- `signed portBASE_TYPE xQueueGenericReceive (xQueueHandle xQueue, void *const pvBuffer, portTickType xTicksToWait, portBASE_TYPE xJustPeek)`
- `unsigned portBASE_TYPE uxQueueMessagesWaiting (const xQueueHandle xQueue)`
- `void vQueueDelete (xQueueHandle pxQueue)`
- `signed portBASE_TYPE xQueueGenericSendFromISR (xQueueHandle pxQueue, const void *const pvItemToQueue, signed portBASE_TYPE *pxHigherPriorityTaskWoken, portBASE_TYPE xCopyPosition)`
- `signed portBASE_TYPE xQueueReceiveFromISR (xQueueHandle pxQueue, void *const pvBuffer, signed portBASE_TYPE *pxTaskWoken)`
- `signed portBASE_TYPE xQueueIsQueueEmptyFromISR (const xQueueHandle pxQueue)`
- `signed portBASE_TYPE xQueueIsQueueFullFromISR (const xQueueHandle pxQueue)`
- `unsigned portBASE_TYPE uxQueueMessagesWaitingFromISR (const xQueueHandle pxQueue)`
- `signed portBASE_TYPE xQueueAltGenericSend (xQueueHandle pxQueue, const void *const pvItemToQueue, portTickType xTicksToWait, portBASE_TYPE xCopyPosition)`
- `signed portBASE_TYPE xQueueAltGenericReceive (xQueueHandle pxQueue, void *const pvBuffer, portTickType xTicksToWait, portBASE_TYPE xJustPeeking)`
- `signed portBASE_TYPE xQueueCRSendFromISR (xQueueHandle pxQueue, const void *pvItemToQueue, signed portBASE_TYPE xCoRoutinePreviouslyWoken)`
- `signed portBASE_TYPE xQueueCRReceiveFromISR (xQueueHandle pxQueue, void *pvBuffer, signed portBASE_TYPE *pxTaskWoken)`
- `signed portBASE_TYPE xQueueCRSend (xQueueHandle pxQueue, const void *pvItemToQueue, portTickType xTicksToWait)`
- `signed portBASE_TYPE xQueueCRReceive (xQueueHandle pxQueue, void *pvBuffer, portTickType xTicksToWait)`
- `xQueueHandle xQueueCreateMutex (void)`
- `xQueueHandle xQueueCreateCountingSemaphore (unsigned portBASE_TYPE uxCountValue, unsigned portBASE_TYPE uxInitialCount)`
- `portBASE_TYPE xQueueTakeMutexRecursive (xQueueHandle pxMutex, portTickType xBlockTime)`
- `portBASE_TYPE xQueueGiveMutexRecursive (xQueueHandle pxMutex)`
- `void vQueueWaitForMessageRestricted (xQueueHandle pxQueue, portTickType xTicksToWait)`

22.29.1 Macro Definition Documentation

22.29.1.1 queueSEND_TO_BACK `#define queueSEND_TO_BACK (0)`

Definition at line 78 of file `queue.h`.

22.29.1.2 queueSEND_TO_FRONT `#define queueSEND_TO_FRONT (1)`

Definition at line 79 of file `queue.h`.

22.29.1.3 xQueueAltPeek #define xQueueAltPeek(xQueue,
pvBuffer,
xTicksToWait) xQueueAltGenericReceive((xQueue), (pvBuffer), (xTicksToWait
, pdTRUE)

Definition at line 1201 of file [queue.h](#).

22.29.1.4 xQueueAltReceive #define xQueueAltReceive(xQueue,
pvBuffer,
xTicksToWait) xQueueAltGenericReceive((xQueue), (pvBuffer), (xTicksToWait
, pdFALSE)

Definition at line 1200 of file [queue.h](#).

22.29.1.5 xQueueAltSendToBack #define xQueueAltSendToBack(xQueue,
pvItemToQueue,
xTicksToWait) xQueueAltGenericSend((xQueue), (pvItemToQueue), (xTicksToWait
, queueSEND_TO_BACK)

Definition at line 1199 of file [queue.h](#).

22.29.1.6 xQueueAltSendToFront #define xQueueAltSendToFront(xQueue,
pvItemToQueue,
xTicksToWait) xQueueAltGenericSend((xQueue), (pvItemToQueue), (xTicksToWait
, queueSEND_TO_FRONT)

Definition at line 1198 of file [queue.h](#).

22.29.1.7 xQueuePeek #define xQueuePeek(xQueue,
pvBuffer,
xTicksToWait) xQueueGenericReceive((xQueue), (pvBuffer), (xTicksToWait),
pdTRUE)

Definition at line 568 of file [queue.h](#).

```
22.29.1.8 xQueueReceive #define xQueueReceive(  
    xQueue,  
    pvBuffer,  
    xTicksToWait) xQueueGenericReceive( (xQueue), (pvBuffer), (xTicksToWait),  
    pdFALSE )
```

Definition at line 661 of file [queue.h](#).

```
22.29.1.9 xQueueSend #define xQueueSend(  
    xQueue,  
    pvItemToQueue,  
    xTicksToWait) xQueueGenericSend( (xQueue), (pvItemToQueue), (xTicksToWait)  
, queueSEND_TO_BACK )
```

Definition at line 386 of file [queue.h](#).

```
22.29.1.10 xQueueSendFromISR #define xQueueSendFromISR(  
    pxQueue,  
    pvItemToQueue,  
    pxHigherPriorityTaskWoken) xQueueGenericSendFromISR( (pxQueue), (pvItemToQueue), (pxHigherPriorityTaskWoken), queueSEND_TO_BACK )
```

Definition at line 1004 of file [queue.h](#).

```
22.29.1.11 xQueueSendToBack #define xQueueSendToBack(  
    xQueue,  
    pvItemToQueue,  
    xTicksToWait) xQueueGenericSend( (xQueue), (pvItemToQueue), (xTicksToWait)  
, queueSEND_TO_BACK )
```

Definition at line 302 of file [queue.h](#).

```
22.29.1.12 xQueueSendToBackFromISR #define xQueueSendToBackFromISR(  
    pxQueue,  
    pvItemToQueue,  
    pxHigherPriorityTaskWoken) xQueueGenericSendFromISR( (pxQueue), (pvItemToQueue), (pxHigherPriorityTaskWoken), queueSEND_TO_BACK )
```

Definition at line 930 of file [queue.h](#).

```
22.29.1.13 xQueueSendToFront #define xQueueSendToFront(  
    xQueue,  
    pvItemToQueue,  
    xTicksToWait ) xQueueGenericSend( ( xQueue ), ( pvItemToQueue ), ( xTicksToWait  
, queueSEND_TO_FRONT )
```

Definition at line 220 of file [queue.h](#).

```
22.29.1.14 xQueueSendToFrontFromISR #define xQueueSendToFrontFromISR(  
    pxQueue,  
    pvItemToQueue,  
    pxHigherPriorityTaskWoken ) xQueueGenericSendFromISR( ( pxQueue ), ( pvItemToQueue ),  
    ( pxHigherPriorityTaskWoken ), queueSEND_TO_FRONT )
```

Definition at line 859 of file [queue.h](#).

22.29.2 Typedef Documentation

```
22.29.2.1 xQueueHandle typedef void* xQueueHandle
```

Type by which queues are referenced. For example, a call to xQueueCreate returns (via a pointer parameter) an xQueueHandle variable that can then be used as a parameter to [xQueueSend\(\)](#), [xQueueReceive\(\)](#), etc.

Definition at line 74 of file [queue.h](#).

22.29.3 Function Documentation

```
22.29.3.1 uxQueueMessagesWaiting() unsigned portBASE_TYPE uxQueueMessagesWaiting (   
    const xQueueHandle xQueue )
```

Definition at line 1051 of file [queue.c](#).

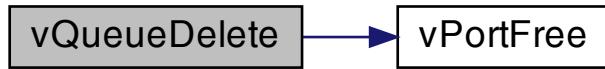
```
22.29.3.2 uxQueueMessagesWaitingFromISR() unsigned portBASE_TYPE uxQueueMessagesWaitingFromISR (   
    const xQueueHandle pxQueue )
```

Definition at line 1065 of file [queue.c](#).

```
22.29.3.3 vQueueDelete() void vQueueDelete (
    xQueueHandle pxQueue )
```

Definition at line 1077 of file [queue.c](#).

Here is the call graph for this function:



```
22.29.3.4 vQueueWaitForMessageRestricted() void vQueueWaitForMessageRestricted (
    xQueueHandle pxQueue,
    portTickType xTicksToWait )
```

```
22.29.3.5 xQueueAltGenericReceive() signed portBASE_TYPE xQueueAltGenericReceive (
    xQueueHandle pxQueue,
    void *const pvBuffer,
    portTickType xTicksToWait,
    portBASE_TYPE xJustPeeking )
```

```
22.29.3.6 xQueueAltGenericSend() signed portBASE_TYPE xQueueAltGenericSend (
    xQueueHandle pxQueue,
    const void *const pvItemToQueue,
    portTickType xTicksToWait,
    portBASE_TYPE xCopyPosition )
```

```
22.29.3.7 xQueueCreate() xQueueHandle xQueueCreate (
    unsigned portBASE_TYPE uxQueueLength,
    unsigned portBASE_TYPE uxItemSize )
```

22.29.3.8 xQueueCreateCountingSemaphore() `xQueueHandle xQueueCreateCountingSemaphore (`

```
    unsigned portBASE_TYPE uxCountValue,  
    unsigned portBASE_TYPE uxInitialCount )
```

Definition at line 440 of file [queue.c](#).

Here is the call graph for this function:

**22.29.3.9 xQueueCreateMutex()** `xQueueHandle xQueueCreateMutex (`

```
    void )
```

Definition at line 141 of file [queue.c](#).

22.29.3.10 xQueueCRReceive() `signed portBASE_TYPE xQueueCRReceive (`

```
    xQueueHandle pxQueue,  
    void * pvBuffer,  
    portTickType xTicksToWait )
```

22.29.3.11 xQueueCRReceiveFromISR() `signed portBASE_TYPE xQueueCRReceiveFromISR (`

```
    xQueueHandle pxQueue,  
    void * pvBuffer,  
    signed portBASE_TYPE * pxTaskWoken )
```

22.29.3.12 xQueueCRSend() `signed portBASE_TYPE xQueueCRSend (`

```
    xQueueHandle pxQueue,  
    const void * pvItemToQueue,  
    portTickType xTicksToWait )
```

22.29.3.13 xQueueCRSendFromISR() `signed portBASE_TYPE xQueueCRSendFromISR (`

```
    xQueueHandle pxQueue,  
    const void * pvItemToQueue,  
    signed portBASE_TYPE xCoRoutinePreviouslyWoken )
```

```
22.29.3.14 xQueueGenericReceive() signed portBASE_TYPE xQueueGenericReceive (
    xQueueHandle xQueue,
    void *const pvBuffer,
    portTickType xTicksToWait,
    portBASE_TYPE xJustPeek )
```

Definition at line 846 of file [queue.c](#).

```
22.29.3.15 xQueueGenericSend() signed portBASE_TYPE xQueueGenericSend (
    xQueueHandle pxQueue,
    const void *const pvItemToQueue,
    portTickType xTicksToWait,
    portBASE_TYPE xCopyPosition )
```

Definition at line 464 of file [queue.c](#).

```
22.29.3.16 xQueueGenericSendFromISR() signed portBASE_TYPE xQueueGenericSendFromISR (
    xQueueHandle pxQueue,
    const void *const pvItemToQueue,
    signed portBASE_TYPE * pxHigherPriorityTaskWoken,
    portBASE_TYPE xCopyPosition )
```

Definition at line 789 of file [queue.c](#).

```
22.29.3.17 xQueueGiveMutexRecursive() portBASE_TYPE xQueueGiveMutexRecursive (
    xQueueHandle pxMutex )
```

```
22.29.3.18 xQueueIsQueueEmptyFromISR() signed portBASE_TYPE xQueueIsQueueEmptyFromISR (
    const xQueueHandle pxQueue )
```

Definition at line 1216 of file [queue.c](#).

```
22.29.3.19 xQueueIsQueueFullFromISR() signed portBASE_TYPE xQueueIsQueueFullFromISR (
    const xQueueHandle pxQueue )
```

Definition at line 1239 of file [queue.c](#).

22.29.3.20 xQueueReceiveFromISR() signed `portBASE_TYPE xQueueReceiveFromISR (`
 `xQueueHandle pxQueue,`
 `void *const pvBuffer,`
 `signed portBASE_TYPE * pxTaskWoken)`

Definition at line 996 of file `queue.c`.

22.29.3.21 xQueueTakeMutexRecursive() `portBASE_TYPE xQueueTakeMutexRecursive (`
 `xQueueHandle pxMutex,`
 `portTickType xBlockTime)`

22.30 queue.h

```
00001 /*  
00002     FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.  
00003  
00004  
00005 ****  
00006 *  
00007 *     FreeRTOS tutorial books are available in pdf and paperback.  
00008 *     Complete, revised, and edited pdf reference manuals are also  
00009 *     available.  
00010 *  
00011 *     Purchasing FreeRTOS documentation will not only help you, by  
00012 *     ensuring you get running as quickly as possible and with an  
00013 *     in-depth knowledge of how to use FreeRTOS, it will also help  
00014 *     the FreeRTOS project to continue with its mission of providing  
00015 *     professional grade, cross platform, de facto standard solutions  
00016 *     for microcontrollers - completely free of charge!  
00017 *  
00018 *     >> See http://www.FreeRTOS.org/Documentation for details. <<  
00019 *  
00020 *     Thank you for using FreeRTOS, and thank you for your support!  
00021 *  
00022 ****  
00023  
00024  
00025 This file is part of the FreeRTOS distribution.  
00026  
00027 FreeRTOS is free software; you can redistribute it and/or modify it under  
00028 the terms of the GNU General Public License (version 2) as published by the  
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.  
00030 >>NOTE<< The modification to the GPL is included to allow you to  
00031 distribute a combined work that includes FreeRTOS without being obliged to  
00032 provide the source code for proprietary components outside of the FreeRTOS  
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but  
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY  
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for  
00036 more details. You should have received a copy of the GNU General Public  
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it  
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained  
00039 by writing to Richard Barry, contact details for whom are available on the  
00040 FreeRTOS WEB site.  
00041  
00042 1 tab == 4 spaces!  
00043  
00044 http://www.FreeRTOS.org - Documentation, latest information, license and  
00045 contact details.  
00046  
00047 http://www.SafeRTOS.com - A version that is certified for use in safety  
00048 critical systems.  
00049  
00050 http://www.OpenRTOS.com - Commercial support, development, porting,  
00051 licensing and training services.  
00052 */  
00053  
00054  
00055 #ifndef QUEUE_H  
00056 #define QUEUE_H  
00057  
00058 #ifndef INC_FREERTOS_H  
00059     #error "#include FreeRTOS.h" must appear in source files before "#include queue.h"  
00060 #endif  
00061  
00062 #ifdef __cplusplus
```

```

00063 extern "C" {
00064 #endif
00065
00066
00067 #include "mpu_wrappers.h"
00068
00074 typedef void * xQueueHandle;
00075
00076
00077 /* For internal use only. */
00078 #define queueSEND_TO_BACK ( 0 )
00079 #define queueSEND_TO_FRONT ( 1 )
00080
00081
00138 xQueueHandle xQueueCreate( unsigned portBASE_TYPE uxQueueLength, unsigned portBASE_TYPE uxItemSize );
00139
00220 #define xQueueSendToFront( xQueue, pvItemToQueue, xTicksToWait ) xQueueGenericSend( ( xQueue ), ( pvItemToQueue ), ( xTicksToWait ), queueSEND_TO_FRONT )
00221
00302 #define xQueueSendToBack( xQueue, pvItemToQueue, xTicksToWait ) xQueueGenericSend( ( xQueue ), ( pvItemToQueue ), ( xTicksToWait ), queueSEND_TO_BACK )
00303
00386 #define xQueueSend( xQueue, pvItemToQueue, xTicksToWait ) xQueueGenericSend( ( xQueue ), ( pvItemToQueue ), ( xTicksToWait ), queueSEND_TO_BACK )
00387
00388
00474 signed portBASE_TYPE xQueueGenericSend( xQueueHandle pxQueue, const void * const pvItemToQueue,
00475           portTickType xTicksToWait, portBASE_TYPE xCopyPosition );
00568 #define xQueuePeek( xQueue, pvBuffer, xTicksToWait ) xQueueGenericReceive( ( xQueue ), ( pvBuffer ), ( xTicksToWait ), pdTRUE )
00569
00661 #define xQueueReceive( xQueue, pvBuffer, xTicksToWait ) xQueueGenericReceive( ( xQueue ), ( pvBuffer ),
00662           ( xTicksToWait ), pdFALSE )
00663
00760 signed portBASE_TYPE xQueueGenericReceive( xQueueHandle xQueue, void * const pvBuffer, portTickType
00761           xTicksToWait, portBASE_TYPE xJustPeek );
00775 unsigned portBASE_TYPE uxQueueMessagesWaiting( const xQueueHandle xQueue );
00776
00789 void vQueueDelete( xQueueHandle pxQueue );
00790
00859 #define xQueueSendToFrontFromISR( pxQueue, pvItemToQueue, pxHigherPriorityTaskWoken )
00860           xQueueGenericSendFromISR( ( pxQueue ), ( pvItemToQueue ), ( pxHigherPriorityTaskWoken ),
00861           queueSEND_TO_FRONT )
00862
00930 #define xQueueSendToBackFromISR( pxQueue, pvItemToQueue, pxHigherPriorityTaskWoken )
00931           xQueueGenericSendFromISR( ( pxQueue ), ( pvItemToQueue ), ( pxHigherPriorityTaskWoken ),
00932           queueSEND_TO_BACK )
01004 #define xQueueSendFromISR( pxQueue, pvItemToQueue, pxHigherPriorityTaskWoken )
01005           xQueueGenericSendFromISR( ( pxQueue ), ( pvItemToQueue ), ( pxHigherPriorityTaskWoken ),
01006           queueSEND_TO_BACK )
01082 signed portBASE_TYPE xQueueGenericSendFromISR( xQueueHandle pxQueue, const void * const pvItemToQueue,
01083           signed portBASE_TYPE *pxHigherPriorityTaskWoken, portBASE_TYPE xCopyPosition );
01083
01171 signed portBASE_TYPE xQueueReceiveFromISR( xQueueHandle pxQueue, void * const pvBuffer, signed
01172           portBASE_TYPE *pxTaskWoken );
01173 /*
01174  * Utilities to query queue that are safe to use from an ISR. These utilities
01175  * should be used only from within an ISR, or within a critical section.
01176 */
01177 signed portBASE_TYPE xQueueIsQueueEmptyFromISR( const xQueueHandle pxQueue );
01178 signed portBASE_TYPE xQueueIsQueueFullFromISR( const xQueueHandle pxQueue );
01179 unsigned portBASE_TYPE uxQueueMessagesWaitingFromISR( const xQueueHandle pxQueue );
01180
01181
01182 /*
01183  * xQueueAltGenericSend() is an alternative version of xQueueGenericSend().
01184  * Likewise xQueueAltGenericReceive() is an alternative version of
01185  * xQueueGenericReceive().
01186 *
01187  * The source code that implements the alternative (Alt) API is much
01188  * simpler because it executes everything from within a critical section.
01189  * This is the approach taken by many other RTOSes, but FreeRTOS.org has the
01190  * preferred fully featured API too. The fully featured API has more
01191  * complex code that takes longer to execute, but makes much less use of
01192  * critical sections. Therefore the alternative API sacrifices interrupt
01193  * responsiveness to gain execution speed, whereas the fully featured API
01194  * sacrifices execution speed to ensure better interrupt responsiveness.
01195 */
01196 signed portBASE_TYPE xQueueAltGenericSend( xQueueHandle pxQueue, const void * const pvItemToQueue,
01197           portTickType xTicksToWait, portBASE_TYPE xCopyPosition );

```

```

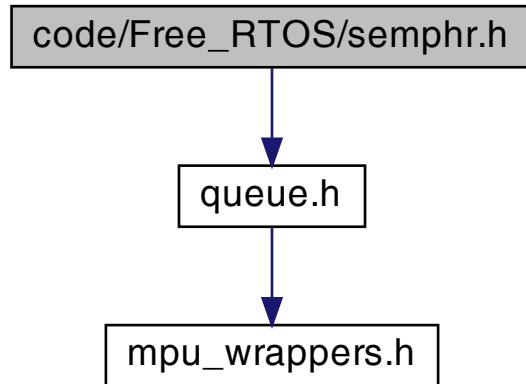
01197 signed portBASE_TYPE xQueueAltGenericReceive( xQueueHandle pxQueue, void * const pvBuffer,
01198     portTickType xTicksToWait, portBASE_TYPE xJustPeeking );
01199 #define xQueueAltSendToFront( xQueue, pvItemToQueue, xTicksToWait ) xQueueAltGenericSend( ( xQueue ),
01200     ( pvItemToQueue ), ( xTicksToWait ), queueSEND_TO_FRONT )
01201 #define xQueueAltSendToBack( xQueue, pvItemToQueue, xTicksToWait ) xQueueAltGenericSend( ( xQueue ), ( 
01202     pvItemToQueue ), ( xTicksToWait ), queueSEND_TO_BACK )
01203 #define xQueueAltReceive( xQueue, pvBuffer, xTicksToWait ) xQueueAltGenericReceive( ( xQueue ), ( 
01204     pvBuffer ), ( xTicksToWait ), pdFALSE )
01205 #define xQueueAltPeek( xQueue, pvBuffer, xTicksToWait ) xQueueAltGenericReceive( ( xQueue ), ( 
01206     pvBuffer ), ( xTicksToWait ), pdTRUE )
01207 /*
01208 * The functions defined above are for passing data to and from tasks. The
01209 * functions below are the equivalents for passing data to and from
01210 * co-routines.
01211 */
01212 signed portBASE_TYPE xQueueCRSendFromISR( xQueueHandle pxQueue, const void *pvItemToQueue, signed
01213     portBASE_TYPE xCoRoutinePreviouslyWoken );
01214 signed portBASE_TYPE xQueueCRReceiveFromISR( xQueueHandle pxQueue, void *pvBuffer, signed
01215     portBASE_TYPE *pxTaskWoken );
01216 signed portBASE_TYPE xQueueCRSend( xQueueHandle pxQueue, const void *pvItemToQueue, portTickType
01217     xTicksToWait );
01218 signed portBASE_TYPE xQueueCRReceive( xQueueHandle pxQueue, void *pvBuffer, portTickType xTicksToWait
01219 );
01220 /*
01221 * For internal use only. Use xSemaphoreCreateMutex() or
01222 * xSemaphoreCreateCounting() instead of calling these functions directly.
01223 */
01224 xQueueHandle xQueueCreateMutex( void );
01225 xQueueHandle xQueueCreateCountingSemaphore( unsigned portBASE_TYPE uxCountValue, unsigned
01226     portBASE_TYPE uxInitialCount );
01227 /*
01228 * For internal use only. Use xSemaphoreTakeMutexRecursive() or
01229 * xSemaphoreGiveMutexRecursive() instead of calling these functions directly.
01230 */
01231 /*
01232 * The registry is provided as a means for kernel aware debuggers to
01233 * locate queues, semaphores and mutexes. Call vQueueAddToRegistry() add
01234 * a queue, semaphore or mutex handle to the registry if you want the handle
01235 * to be available to a kernel aware debugger. If you are not using a kernel
01236 * aware debugger then this function can be ignored.
01237 *
01238 * configQUEUE_REGISTRY_SIZE defines the maximum number of handles the
01239 * registry can hold. configQUEUE_REGISTRY_SIZE must be greater than 0
01240 * within FreeRTOSConfig.h for the registry to be available. Its value
01241 * does not effect the number of queues, semaphores and mutexes that can be
01242 * created - just the number that the registry can hold.
01243 *
01244 * @param xQueue The handle of the queue being added to the registry. This
01245 * is the handle returned by a call to xQueueCreate(). Semaphore and mutex
01246 * handles can also be passed in here.
01247 *
01248 * @param pcName The name to be associated with the handle. This is the
01249 * name that the kernel aware debugger will display.
01250 */
01251 #if configQUEUE_REGISTRY_SIZE > 0U
01252     void vQueueAddToRegistry( xQueueHandle xQueue, signed char *pcName );
01253 #endif
01254
01255 /* Not a public API function, hence the 'Restricted' in the name. */
01256 void vQueueWaitForMessageRestricted( xQueueHandle pxQueue, portTickType xTicksToWait );
01257
01258
01259 #ifdef __cplusplus
01260 }
01261 #endif
01262
01263 #endif /* QUEUE_H */
01264

```

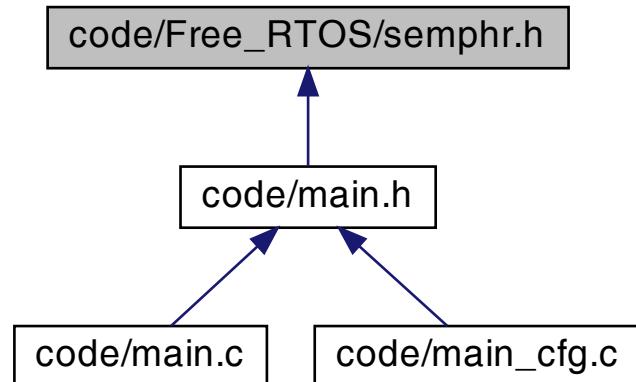
22.31 code/FreeRTOS/semphr.h File Reference

```
#include "queue.h"
```

Include dependency graph for semphr.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define semBINARY_SEMAPHORE_QUEUE_LENGTH ((unsigned char) 1U)
- #define semSEMAPHORE_QUEUE_ITEM_LENGTH ((unsigned char) 0U)
- #define semGIVE_BLOCK_TIME ((portTickType) 0U)
- #define vSemaphoreCreateBinary(xSemaphore)
- #define xSemaphoreTake(xSemaphore, xBlockTime) xQueueGenericReceive((xQueueHandle) (xSemaphore), NULL, (xBlockTime), pdFALSE)
- #define xSemaphoreTakeRecursive(xMutex, xBlockTime) xQueueTakeMutexRecursive((xMutex), (xBlockTime))

- #define xSemaphoreAltTake(xSemaphore, xBlockTime) xQueueAltGenericReceive((xQueueHandle) (xSemaphore), NULL, (xBlockTime), pdFALSE)
- #define xSemaphoreGive(xSemaphore) xQueueGenericSend((xQueueHandle) (xSemaphore), NULL, semGIVE_BLOCK_TIME, queueSEND_TO_BACK)
- #define xSemaphoreGiveRecursive(xMutex) xQueueGiveMutexRecursive((xMutex))
- #define xSemaphoreAltGive(xSemaphore) xQueueAltGenericSend((xQueueHandle) (xSemaphore), NULL, semGIVE_BLOCK_TIME, queueSEND_TO_BACK)
- #define xSemaphoreGiveFromISR(xSemaphore, pxHigherPriorityTaskWoken) xQueueGenericSendFromISR((xQueueHandle) (xSemaphore), NULL, (pxHigherPriorityTaskWoken), queueSEND_TO_BACK)
- #define xSemaphoreCreateMutex() xQueueCreateMutex()
- #define xSemaphoreCreateRecursiveMutex() xQueueCreateMutex()
- #define xSemaphoreCreateCounting(uxMaxCount, uxInitialCount) xQueueCreateCountingSemaphore((uxMaxCount), (uxInitialCount))
- #define vSemaphoreDelete(xSemaphore) vQueueDelete((xQueueHandle) xSemaphore)

Typedefs

- typedef xQueueHandle xSemaphoreHandle

22.31.1 Macro Definition Documentation

22.31.1.1 semBINARY_SEMAPHORE_QUEUE_LENGTH #define semBINARY_SEMAPHORE_QUEUE_LENGTH ((unsigned char) 1U)

Definition at line 65 of file [semphr.h](#).

22.31.1.2 semGIVE_BLOCK_TIME #define semGIVE_BLOCK_TIME ((portTickType) 0U)

Definition at line 67 of file [semphr.h](#).

22.31.1.3 semSEMAPHORE_QUEUE_ITEM_LENGTH #define semSEMAPHORE_QUEUE_ITEM_LENGTH ((unsigned char) 0U)

Definition at line 66 of file [semphr.h](#).

22.31.1.4 vSemaphoreCreateBinary #define vSemaphoreCreateBinary(
 xSemaphore)

Value:

```
\n    {\n        ( xSemaphore ) = xQueueCreate( ( unsigned\n        portBASE_TYPE ) 1, semSEMAPHORE_QUEUE_ITEM_LENGTH );\n        if( ( xSemaphore ) != NULL )\n            {\n                xSemaphoreGive( ( xSemaphore ) );\n            }\n    }\n}
```

Definition at line 108 of file [semphr.h](#).

22.31.1.5 vSemaphoreDelete #define vSemaphoreDelete(
 xSemaphore) vQueueDelete((xQueueHandle) xSemaphore)

Definition at line 720 of file [semphr.h](#).

22.31.1.6 xSemaphoreAltGive #define xSemaphoreAltGive(
 xSemaphore) xQueueAltGenericSend((xQueueHandle) (xSemaphore), NULL, semGIVE_BLOCK_TIME,
queueSEND_TO_BACK)

Definition at line 450 of file [semphr.h](#).

22.31.1.7 xSemaphoreAltTake #define xSemaphoreAltTake(
 xSemaphore,
 xBlockTime) xQueueAltGenericReceive((xQueueHandle) (xSemaphore), NULL, (xBlockTime), pdFALSE)

Definition at line 289 of file [semphr.h](#).

22.31.1.8 xSemaphoreCreateCounting #define xSemaphoreCreateCounting(
 uxMaxCount,
 uxInitialCount) xQueueCreateCountingSemaphore((uxMaxCount), (uxInitialCount
))

Definition at line 706 of file [semphr.h](#).

22.31.1.9 xSemaphoreCreateMutex `#define xSemaphoreCreateMutex() xQueueCreateMutex()`

Definition at line [588](#) of file [semphr.h](#).

22.31.1.10 xSemaphoreCreateRecursiveMutex `#define xSemaphoreCreateRecursiveMutex() xQueueCreateMutex()`

Definition at line [643](#) of file [semphr.h](#).

22.31.1.11 xSemaphoreGive `#define xSemaphoreGive(xSemaphore) xQueueGenericSend((xQueueHandle) (xSemaphore), NULL, semGIVE_BLOCK_TIME, queueSEND_TO_BACK)`

Definition at line [352](#) of file [semphr.h](#).

22.31.1.12 xSemaphoreGiveFromISR `#define xSemaphoreGiveFromISR(xSemaphore, pxHigherPriorityTaskWoken) xQueueGenericSendFromISR((xQueueHandle) (xSemaphore), NULL, (pxHigherPriorityTaskWoken), queueSEND_TO_BACK)`

Definition at line [541](#) of file [semphr.h](#).

22.31.1.13 xSemaphoreGiveRecursive `#define xSemaphoreGiveRecursive(xMutex) xQueueGiveMutexRecursive((xMutex))`

Definition at line [436](#) of file [semphr.h](#).

22.31.1.14 xSemaphoreTake `#define xSemaphoreTake(xSemaphore, xBlockTime) xQueueGenericReceive((xQueueHandle) (xSemaphore), NULL, (xBlockTime), pdFALSE)`

Definition at line [181](#) of file [semphr.h](#).

22.31.1.15 xSemaphoreTakeRecursive `#define xSemaphoreTakeRecursive(xMutex, xBlockTime) xQueueTakeMutexRecursive((xMutex), (xBlockTime))`

Definition at line [274](#) of file [semphr.h](#).

22.31.2 Typedef Documentation

22.31.2.1 xSemaphoreHandle `typedef xQueueHandle xSemaphoreHandle`

Definition at line 63 of file [semphr.h](#).

22.32 semphr.h

```

00001 /*
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005 ****
00006 *
00007 *   FreeRTOS tutorial books are available in pdf and paperback.
00008 *   Complete, revised, and edited pdf reference manuals are also
00009 *   available.
00010 *
00011 *   Purchasing FreeRTOS documentation will not only help you, by
00012 *   ensuring you get running as quickly as possible and with an
00013 *   in-depth knowledge of how to use FreeRTOS, it will also help
00014 *   the FreeRTOS project to continue with its mission of providing
00015 *   professional grade, cross platform, de facto standard solutions
00016 *   for microcontrollers - completely free of charge!
00017 *
00018 *   >> See http://www.FreeRTOS.org/Documentation for details. <<
00019 *
00020 *   Thank you for using FreeRTOS, and thank you for your support!
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 >>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039 by writing to Richard Barry, contact details for whom are available on the
00040 FreeRTOS WEB site.
00041
00042 1 tab == 4 spaces!
00043
00044 http://www.FreeRTOS.org - Documentation, latest information, license and
00045 contact details.
00046
00047 http://www.SafeRTOS.com - A version that is certified for use in safety
00048 critical systems.
00049
00050 http://www.OpenRTOS.com - Commercial support, development, porting,
00051 licensing and training services.
00052 */
00053
00054 #ifndef SEMAPHORE_H
00055 #define SEMAPHORE_H
00056
00057 #ifndef INC_FREERTOS_H
00058     #error "#include FreeRTOS.h" must appear in source files before "#include semphr.h"
00059 #endif
00060
00061 #include "queue.h"
00062
00063 typedef xQueueHandle xSemaphoreHandle;
00064
00065 #define semBINARY_SEMAPHORE_QUEUE_LENGTH      ( ( unsigned char ) 1U )
00066 #define semSEMAPHORE_QUEUE_ITEM_LENGTH        ( ( unsigned char ) 0U )
00067 #define semGIVE_BLOCK_TIME                   ( ( portTickType ) 0U )
00068
00069

```

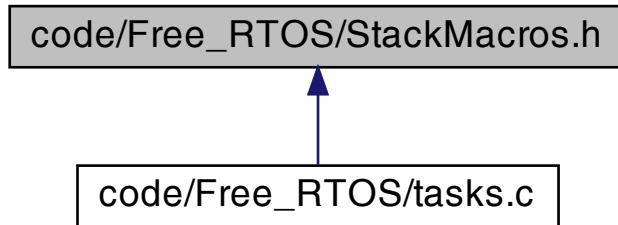
```

00108 #define vSemaphoreCreateBinary( xSemaphore )           \
00109     portBASE_TYPE ) 1, semSEMAPHORE_QUEUE_ITEM_LENGTH ); \
00110                                         \
00111                                         \
00112                                         \
00113                                         \
00114                                         \
00115                                         \
00116 #define xSemaphoreTake( xSemaphore, xBlockTime )        xQueueGenericReceive( ( xQueueHandle ) ( \
00117     xSemaphore ), NULL, ( xBlockTime ), pdFALSE ) \
00118                                         \
00119 #define xSemaphoreTakeRecursive( xMutex, xBlockTime )   xQueueTakeMutexRecursive( ( xMutex ), ( \
00120     xBlockTime ) ) \
00121                                         \
00122                                         \
00123                                         \
00124                                         \
00125                                         \
00126                                         \
00127 /* \
00128 * xSemaphoreAltTake() is an alternative version of xSemaphoreTake(). \
00129 * \
00130 * The source code that implements the alternative (Alt) API is much \
00131 * simpler because it executes everything from within a critical section. \
00132 * This is the approach taken by many other RTOSes, but FreeRTOS.org has the \
00133 * preferred fully featured API too. The fully featured API has more \
00134 * complex code that takes longer to execute, but makes much less use of \
00135 * critical sections. Therefore the alternative API sacrifices interrupt \
00136 * responsiveness to gain execution speed, whereas the fully featured API \
00137 * sacrifices execution speed to ensure better interrupt responsiveness. \
00138 */ \
00139 #define xSemaphoreAltTake( xSemaphore, xBlockTime )       xQueueAltGenericReceive( ( xQueueHandle ) ( \
00140     xSemaphore ), NULL, ( xBlockTime ), pdFALSE ) \
00141                                         \
00142 #define xSemaphoreGive( xSemaphore )                   xQueueGenericSend( ( xQueueHandle ) ( xSemaphore ), NULL, \
00143     semGIVE_BLOCK_TIME, queueSEND_TO_BACK ) \
00144                                         \
00145 #define xSemaphoreGiveRecursive( xMutex )             xQueueGiveMutexRecursive( ( xMutex ) ) \
00146                                         \
00147 /* \
00148 * xSemaphoreAltGive() is an alternative version of xSemaphoreGive(). \
00149 * \
00150 * The source code that implements the alternative (Alt) API is much \
00151 * simpler because it executes everything from within a critical section. \
00152 * This is the approach taken by many other RTOSes, but FreeRTOS.org has the \
00153 * preferred fully featured API too. The fully featured API has more \
00154 * complex code that takes longer to execute, but makes much less use of \
00155 * critical sections. Therefore the alternative API sacrifices interrupt \
00156 * responsiveness to gain execution speed, whereas the fully featured API \
00157 * sacrifices execution speed to ensure better interrupt responsiveness. \
00158 */ \
00159 #define xSemaphoreAltGive( xSemaphore )      xQueueAltGenericSend( ( xQueueHandle ) ( xSemaphore ), \
00160     NULL, semGIVE_BLOCK_TIME, queueSEND_TO_BACK ) \
00161                                         \
00162 #define xSemaphoreGiveFromISR( xSemaphore, pxHigherPriorityTaskWoken ) \
00163     xQueueGenericSendFromISR( ( xQueueHandle ) ( xSemaphore ), NULL, ( pxHigherPriorityTaskWoken ), \
00164     queueSEND_TO_BACK ) \
00165                                         \
00166 #define xSemaphoreCreateMutex()               xQueueCreateMutex() \
00167                                         \
00168 #define xSemaphoreCreateRecursiveMutex()    xQueueCreateMutex() \
00169                                         \
00170 #define xSemaphoreCreateCounting( uxMaxCount, uxInitialCount ) xQueueCreateCountingSemaphore( ( \
00171     uxMaxCount ), ( uxInitialCount ) ) \
00172                                         \
00173 #define vSemaphoreDelete( xSemaphore )        vQueueDelete( ( xQueueHandle ) xSemaphore ) \
00174                                         \
00175 #endif /* SEMAPHORE_H */

```

22.33 code/FreeRTOS/StackMacros.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define taskFIRST_CHECK_FOR_STACK_OVERFLOW()
- #define taskSECOND_CHECK_FOR_STACK_OVERFLOW()

22.33.1 Macro Definition Documentation

22.33.1.1 taskFIRST_CHECK_FOR_STACK_OVERFLOW `#define taskFIRST_CHECK_FOR_STACK_OVERFLOW()`

Definition at line 76 of file [StackMacros.h](#).

22.33.1.2 taskSECOND_CHECK_FOR_STACK_OVERFLOW `#define taskSECOND_CHECK_FOR_STACK_OVERFLOW()`

Definition at line 77 of file [StackMacros.h](#).

22.34 StackMacros.h

```

00001 /*
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005 ****
00006 *
00007 *   FreeRTOS tutorial books are available in pdf and paperback.
00008 *   Complete, revised, and edited pdf reference manuals are also
00009 *   available.
00010 *
00011 *   Purchasing FreeRTOS documentation will not only help you, by
00012 *   ensuring you get running as quickly as possible and with an
00013 *   in-depth knowledge of how to use FreeRTOS, it will also help
00014 *   the FreeRTOS project to continue with its mission of providing
00015 *   professional grade, cross platform, de facto standard solutions
00016 *   for microcontrollers - completely free of charge!
00017 *
00018 *   >>> See http://www.FreeRTOS.org/Documentation for details. <<
00019 *
00020 *   Thank you for using FreeRTOS, and thank you for your support!
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 >>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039 by writing to Richard Barry, contact details for whom are available on the
00040 FreeRTOS WEB site.
00041
00042 1 tab == 4 spaces!
00043
00044 http://www.FreeRTOS.org - Documentation, latest information, license and
00045 contact details.
00046
00047 http://www.SafeRTOS.com - A version that is certified for use in safety
00048 critical systems.
00049
00050 http://www.OpenRTOS.com - Commercial support, development, porting,
00051 licensing and training services.
00052 */
00053
00054 #ifndef STACK_MACROS_H
00055 #define STACK_MACROS_H
00056
00057 /*
00058 * Call the stack overflow hook function if the stack of the task being swapped
00059 * out is currently overflowed, or looks like it might have overflowed in the
00060 * past.
00061 *
00062 * Setting configCHECK_FOR_STACK_OVERFLOW to 1 will cause the macro to check
00063 * the current stack state only - comparing the current top of stack value to
00064 * the stack limit. Setting configCHECK_FOR_STACK_OVERFLOW to greater than 1
00065 * will also cause the last few stack bytes to be checked to ensure the value
00066 * to which the bytes were set when the task was created have not been
00067 * overwritten. Note this second test does not guarantee that an overflowed
00068 * stack will always be recognised.
00069 */
00070
00071 /*****
00072
00073 #if( configCHECK_FOR_STACK_OVERFLOW == 0 )
00074
00075     /* FreeRTOSConfig.h is not set to check for stack overflows. */
00076     #define taskFIRST_CHECK_FOR_STACK_OVERFLOW()
00077     #define taskSECOND_CHECK_FOR_STACK_OVERFLOW()
00078
00079 #endif /* configCHECK_FOR_STACK_OVERFLOW == 0 */
00080 /*****
00081
00082 #if( configCHECK_FOR_STACK_OVERFLOW == 1 )
00083
00084     /* FreeRTOSConfig.h is only set to use the first method of
00085     * overflow checking. */
00086

```

```
00086     #define taskSECOND_CHECK_FOR_STACK_OVERFLOW()
00087
00088 #endif
00089 /*-----*/
00090
00091 #if( ( configCHECK_FOR_STACK_OVERFLOW > 0 ) && ( portSTACK_GROWTH < 0 ) )
00092
00093     /* Only the current stack state is to be checked. */
00094     #define taskFIRST_CHECK_FOR_STACK_OVERFLOW()
00095
00096     /*
00097         /* Is the currently saved stack pointer within the stack limit? */
00098
00099         if( pxCurrentTCB->pTopOfStack <= pxCurrentTCB->pStack )
00100
00101     {
00102
00103     vApplicationStackOverflowHook( ( xTaskHandle ) pxCurrentTCB, pxCurrentTCB->pcTaskName );
00104
00105     }
00106
00107 #endif /* configCHECK_FOR_STACK_OVERFLOW > 0 */
00108 /*-----*/
00109
00110 #if( ( configCHECK_FOR_STACK_OVERFLOW > 0 ) && ( portSTACK_GROWTH > 0 ) )
00111
00112     /* Only the current stack state is to be checked. */
00113     #define taskFIRST_CHECK_FOR_STACK_OVERFLOW()
00114
00115     /*
00116         /* Is the currently saved stack pointer within the stack limit? */
00117
00118         if( pxCurrentTCB->pTopOfStack >= pxCurrentTCB->pEndOfStack )
00119
00120     {
00121
00122     vApplicationStackOverflowHook( ( xTaskHandle ) pxCurrentTCB, pxCurrentTCB->pcTaskName );
00123
00124     }
00125
00126 #endif /* configCHECK_FOR_STACK_OVERFLOW == 1 */
00127 /*-----*/
00128
00129 #if( ( configCHECK_FOR_STACK_OVERFLOW > 1 ) && ( portSTACK_GROWTH < 0 ) )
00130
00131     #define taskSECOND_CHECK_FOR_STACK_OVERFLOW()
00132
00133     /*
00134         /* Has the extremity of the task stack ever been written over? */
00135
00136         if( memcmp( ( void * ) pxCurrentTCB->pStack, ( void * ) ucExpectedStackBytes, sizeof(
00137             ucExpectedStackBytes ) ) != 0 )
00138
00139     {
00140
00141     vApplicationStackOverflowHook( ( xTaskHandle ) pxCurrentTCB, pxCurrentTCB->pcTaskName );
00142
00143     }
00144
00145 #endif /* #if( configCHECK_FOR_STACK_OVERFLOW > 1 ) */
00146 /*-----*/
00147
00148 #if( ( configCHECK_FOR_STACK_OVERFLOW > 1 ) && ( portSTACK_GROWTH > 0 ) )
```

```

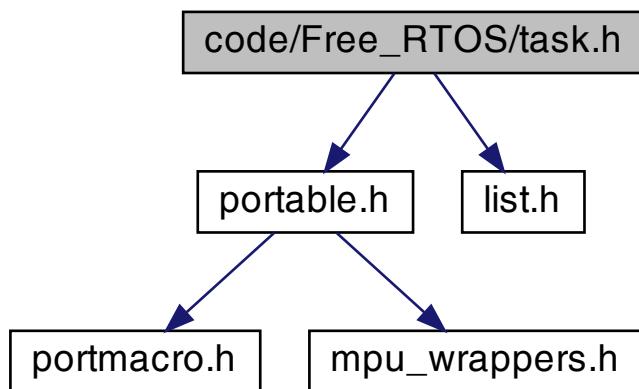
00144 #define taskSECOND_CHECK_FOR_STACK_OVERFLOW()
00145     \
00146     {
00147         \
00148         char *pcEndOfStack = ( char * ) pxCurrentTCB->pxEndOfStack;
00149         \
00150         static const unsigned char ucExpectedStackBytes[] = { tskSTACK_FILL_BYTE, tskSTACK_FILL_BYTE,
00151             tskSTACK_FILL_BYTE, tskSTACK_FILL_BYTE, \
00152             tskSTACK_FILL_BYTE, tskSTACK_FILL_BYTE, \
00153             tskSTACK_FILL_BYTE, tskSTACK_FILL_BYTE, \
00154             tskSTACK_FILL_BYTE, tskSTACK_FILL_BYTE } ; \
00155         \
00156         \
00157         /* Has the extremity of the task stack ever been written over? */
00158         if( memcmp( ( void * ) pcEndOfStack, ( void * ) ucExpectedStackBytes, sizeof(
00159             ucExpectedStackBytes ) ) != 0 ) \
00160             vApplicationStackOverflowHook( \
00161                 ( xTaskHandle ) pxCurrentTCB, pxCurrentTCB->pcTaskName );
00162     \
00163 }
00164 #endif /* #if( configCHECK_FOR_STACK_OVERFLOW > 1 ) */
00165 /*-----*/
00166
00167 #endif /* STACK_MACROS_H */
00168

```

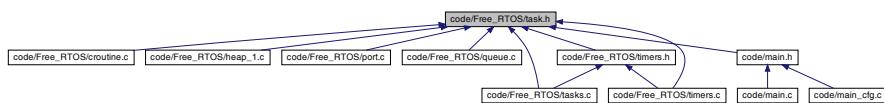
22.35 code/FreeRTOS/task.h File Reference

```
#include "portable.h"
#include "list.h"

Include dependency graph for task.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `xTIME_OUT`
- struct `xMEMORY_REGION`
- struct `xTASK_PARAMTERS`

Macros

- `#define tskKERNEL_VERSION_NUMBER "V7.0.1"`
- `#define tskIDLE_PRIORITY ((unsigned portBASE_TYPE) 0U)`
- `#define taskYIELD() portYIELD()`
- `#define taskENTER_CRITICAL() portENTER_CRITICAL()`
- `#define taskEXIT_CRITICAL() portEXIT_CRITICAL()`
- `#define taskDISABLE_INTERRUPTS() portDISABLE_INTERRUPTS()`
- `#define taskENABLE_INTERRUPTS() portENABLE_INTERRUPTS()`
- `#define taskSCHEDULER_NOT_STARTED 0`
- `#define taskSCHEDULER_RUNNING 1`
- `#define taskSCHEDULER_SUSPENDED 2`
- `#define xTaskCreate(pvTaskCode, pcName, usStackDepth, pvParameters, uxPriority, pxCreatedTask) xTaskGenericCreate((pvTaskCode), (pcName), (usStackDepth), (pvParameters), (uxPriority), (pxCreatedTask), (NULL), (NULL))`
- `#define xTaskCreateRestricted(x, pxCreatedTask) xTaskGenericCreate(((x)->pvTaskCode), ((x)->pcName), ((x)->usStackDepth), ((x)->pvParameters), ((x)->uxPriority), (pxCreatedTask), ((x)->pxuStackBuffer), ((x)->xRegions))`

Typedefs

- `typedef void * xTaskHandle`
- `typedef struct xTIME_OUT xTimeOutType`
- `typedef struct xMEMORY_REGION xMemoryRegion`
- `typedef struct xTASK_PARAMTERS xTaskParameters`

Functions

- `void vTaskAllocateMPURegions (xTaskHandle xTask, const xMemoryRegion *const pxRegions) PRIVILEGED_FUNCTION`
- `void vTaskDelete (xTaskHandle pxTaskToDelete) PRIVILEGED_FUNCTION`
- `void vTaskDelay (portTickType xTicksToDelay) PRIVILEGED_FUNCTION`
- `void vTaskDelayUntil (portTickType *const pxPreviousWakeTime, portTickType xTimeIncrement) PRIVILEGED_FUNCTION`
- `unsigned portBASE_TYPE uxTaskPriorityGet (xTaskHandle pxTask) PRIVILEGED_FUNCTION`
- `void vTaskPrioritySet (xTaskHandle pxTask, unsigned portBASE_TYPE uxNewPriority) PRIVILEGED_FUNCTION`
- `void vTaskSuspend (xTaskHandle pxTaskToSuspend) PRIVILEGED_FUNCTION`

- void `vTaskResume` (`xTaskHandle pxTaskToResume`) `PRIVILEGED_FUNCTION`
- `portBASE_TYPE xTaskResumeFromISR` (`xTaskHandle pxTaskToResume`) `PRIVILEGED_FUNCTION`
- void `vTaskStartScheduler` (`void`) `PRIVILEGED_FUNCTION`
- void `vTaskEndScheduler` (`void`) `PRIVILEGED_FUNCTION`
- void `vTaskSuspendAll` (`void`) `PRIVILEGED_FUNCTION`
- signed `portBASE_TYPE xTaskResumeAll` (`void`) `PRIVILEGED_FUNCTION`
- signed `portBASE_TYPE xTaskIsTaskSuspended` (`xTaskHandle xTask`) `PRIVILEGED_FUNCTION`
- `portTickType xTaskGetTickCount` (`void`) `PRIVILEGED_FUNCTION`
- `portTickType xTaskGetTickCountFromISR` (`void`) `PRIVILEGED_FUNCTION`
- unsigned `portBASE_TYPE uxTaskGetNumberOfTasks` (`void`) `PRIVILEGED_FUNCTION`
- signed char * `pcTaskGetTaskName` (`xTaskHandle xTaskToQuery`)
- void `vTaskList` (signed char *`pcWriteBuffer`) `PRIVILEGED_FUNCTION`
- void `vTaskGetRunTimeStats` (signed char *`pcWriteBuffer`) `PRIVILEGED_FUNCTION`
- void `vTaskStartTrace` (signed char *`pcBuffer`, unsigned long `ulBufferSize`) `PRIVILEGED_FUNCTION`
- unsigned long `ulTaskEndTrace` (`void`) `PRIVILEGED_FUNCTION`
- unsigned `portBASE_TYPE uxTaskGetStackHighWaterMark` (`xTaskHandle xTask`) `PRIVILEGED_FUNCTION`
- `portBASE_TYPE xTaskCallApplicationTaskHook` (`xTaskHandle xTask`, `void *pvParameter`) `PRIVILEGED_FUNCTION`
- `xTaskHandle xTaskGetIdleTaskHandle` (`void`)
- void `vTaskIncrementTick` (`void`) `PRIVILEGED_FUNCTION`
- void `vTaskPlaceOnEventList` (const `xList *const pxEventList`, `portTickType xTicksToWait`) `PRIVILEGED_FUNCTION`
- void `vTaskPlaceOnEventListRestricted` (const `xList *const pxEventList`, `portTickType xTicksToWait`) `PRIVILEGED_FUNCTION`
- signed `portBASE_TYPE xTaskRemoveFromEventList` (const `xList *const pxEventList`) `PRIVILEGED_FUNCTION`
- void `vTaskSwitchContext` (`void`) `PRIVILEGED_FUNCTION`
- `xTaskHandle xTaskGetCurrentTaskHandle` (`void`) `PRIVILEGED_FUNCTION`
- void `vTaskSetTimeOutState` (`xTimeOutType *const pxTimeOut`) `PRIVILEGED_FUNCTION`
- `portBASE_TYPE xTaskCheckForTimeOut` (`xTimeOutType *const pxTimeOut`, `portTickType *const pxTicksToWait`) `PRIVILEGED_FUNCTION`
- void `vTaskMissedYield` (`void`) `PRIVILEGED_FUNCTION`
- `portBASE_TYPE xTaskGetSchedulerState` (`void`) `PRIVILEGED_FUNCTION`
- void `vTaskPriorityInherit` (`xTaskHandle *const pxMutexHolder`) `PRIVILEGED_FUNCTION`
- void `vTaskPriorityDisinherit` (`xTaskHandle *const pxMutexHolder`) `PRIVILEGED_FUNCTION`
- signed `portBASE_TYPE xTaskGenericCreate` (`pdTASK_CODE pxTaskCode`, const signed char *`const pcName`, unsigned short `usStackDepth`, `void *pvParameters`, unsigned `portBASE_TYPE uxPriority`, `xTaskHandle *pxCreatedTask`, `portSTACK_TYPE *puxBStackBuffer`, const `xMemoryRegion *const xRegions`) `PRIVILEGED_FUNCTION`

22.35.1 Macro Definition Documentation

22.35.1.1 `taskDISABLE_INTERRUPTS` `#define taskDISABLE_INTERRUPTS() portDISABLE_INTERRUPTS()`

Definition at line 173 of file `task.h`.

22.35.1.2 `taskENABLE_INTERRUPTS` `#define taskENABLE_INTERRUPTS() portENABLE_INTERRUPTS()`

Definition at line 183 of file `task.h`.

22.35.1.3 taskENTER_CRITICAL #define taskENTER_CRITICAL() portENTER_CRITICAL()

Definition at line 149 of file [task.h](#).

22.35.1.4 taskEXIT_CRITICAL #define taskEXIT_CRITICAL() portEXIT_CRITICAL()

Definition at line 163 of file [task.h](#).

22.35.1.5 taskSCHEDULER_NOT_STARTED #define taskSCHEDULER_NOT_STARTED 0

Definition at line 186 of file [task.h](#).

22.35.1.6 taskSCHEDULER_RUNNING #define taskSCHEDULER_RUNNING 1

Definition at line 187 of file [task.h](#).

22.35.1.7 taskSCHEDULER_SUSPENDED #define taskSCHEDULER_SUSPENDED 2

Definition at line 188 of file [task.h](#).

22.35.1.8 taskYIELD #define taskYIELD() portYIELD()

Definition at line 135 of file [task.h](#).

22.35.1.9 tskIDLE_PRIORITY #define tskIDLE_PRIORITY ((unsigned portBASE_TYPE) 0U)

Definition at line 125 of file [task.h](#).

22.35.1.10 tskKERNEL_VERSION_NUMBER #define tskKERNEL_VERSION_NUMBER "V7.0.1"

Definition at line 73 of file [task.h](#).

```
22.35.1.11 xTaskCreate #define xTaskCreate(  
    pvTaskCode,  
    pcName,  
    usStackDepth,  
    pvParameters,  
    uxPriority,  
    pxCreatedTask ) xTaskGenericCreate( ( pvTaskCode ), ( pcName ), ( usStackDepth  
, ( pvParameters ), ( uxPriority ), ( pxCreatedTask ), ( NULL ), ( NULL ) )
```

Definition at line 270 of file [task.h](#).

```
22.35.1.12 xTaskCreateRestricted #define xTaskCreateRestricted(  
    x,  
    pxCreatedTask ) xTaskGenericCreate( ((x)->pvTaskCode), ((x)->pcName), ((x)->us←  
StackDepth), ((x)->pvParameters), ((x)->uxPriority), (pxCreatedTask), ((x)->puxStackBuffer),  
((x)->xRegions) )
```

Definition at line 339 of file [task.h](#).

22.35.2 Typedef Documentation

```
22.35.2.1 xMemoryRegion typedef struct xMEMORY_REGION xMemoryRegion
```

```
22.35.2.2 xTaskHandle typedef void* xTaskHandle
```

Definition at line 85 of file [task.h](#).

```
22.35.2.3 xTaskParameters typedef struct xTASK_PARAMTERS xTaskParameters
```

```
22.35.2.4 xTimeOutType typedef struct xTIME_OUT xTimeOutType
```

22.35.3 Function Documentation

```
22.35.3.1 pcTaskGetTaskName() signed char* pcTaskGetTaskName (   
    xTaskHandle xTaskToQuery )
```

22.35.3.2 ulTaskEndTrace() `unsigned long ulTaskEndTrace (void)`

22.35.3.3 uxTaskGetNumberOfTasks() `unsigned portBASE_TYPE uxTaskGetNumberOfTasks (void)`

Definition at line 1277 of file [tasks.c](#).

22.35.3.4 uxTaskGetStackHighWaterMark() `unsigned portBASE_TYPE uxTaskGetStackHighWaterMark (xTaskHandle xTask)`

[task.h](#)

`unsigned portBASE_TYPE uxTaskGetStackHighWaterMark(xTaskHandle xTask);`

INCLUDE_uxTaskGetStackHighWaterMark must be set to 1 in [FreeRTOSConfig.h](#) for this function to be available.

Returns the high water mark of the stack associated with xTask. That is, the minimum free stack space there has been (in words, so on a 32 bit machine a value of 1 means 4 bytes) since the task started. The smaller the returned number the closer the task has come to overflowing its stack.

Parameters

<code>xTask</code>	Handle of the task associated with the stack to be checked. Set xTask to NULL to check the stack of the calling task.
--------------------	---

Returns

The smallest amount of free stack space there has been (in bytes) since the task referenced by xTask was created.

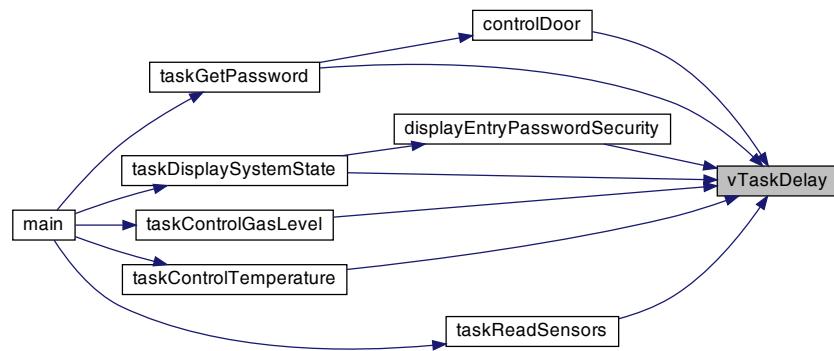
22.35.3.5 uxTaskPriorityGet() `unsigned portBASE_TYPE uxTaskPriorityGet (xTaskHandle pxTask)`

22.35.3.6 vTaskAllocateMPURegions() `void vTaskAllocateMPURegions (xTaskHandle xTask, const xMemoryRegion *const pxRegions)`

```
22.35.3.7 vTaskDelay() void vTaskDelay (
    portTickType xTicksToDelay )
```

Definition at line 718 of file [tasks.c](#).

Here is the caller graph for this function:



```
22.35.3.8 vTaskDelayUntil() void vTaskDelayUntil (
    portTickType *const pxPreviousWakeTime,
    portTickType xTimeIncrement )
```

Definition at line 653 of file [tasks.c](#).

Here is the call graph for this function:



```
22.35.3.9 vTaskDelete() void vTaskDelete (
    xTaskHandle pxTaskToDelete )
```

Definition at line 587 of file [tasks.c](#).

```
22.35.3.10 vTaskEndScheduler() void vTaskEndScheduler (
    void )
```

Definition at line 1143 of file [tasks.c](#).

```
22.35.3.11 vTaskGetRunTimeStats() void vTaskGetRunTimeStats (
    signed char * pcWriteBuffer )
```

```
22.35.3.12 vTaskIncrementTick() void vTaskIncrementTick (
    void )
```

Definition at line 1493 of file [tasks.c](#).

```
22.35.3.13 vTaskList() void vTaskList (
    signed char * pcWriteBuffer )
```

```
22.35.3.14 vTaskMissedYield() void vTaskMissedYield (
    void )
```

Definition at line 1894 of file [tasks.c](#).

```
22.35.3.15 vTaskPlaceOnEventList() void vTaskPlaceOnEventList (
    const xList *const pxEventList,
    portTickType xTicksToWait )
```

Definition at line 1707 of file [tasks.c](#).

```
22.35.3.16 vTaskPlaceOnEventListRestricted() void vTaskPlaceOnEventListRestricted (
    const xList *const pxEventList,
    portTickType xTicksToWait )
```

```
22.35.3.17 vTaskPriorityDisinherit() void vTaskPriorityDisinherit (
    xTaskHandle *const pxMutexHolder )
```

22.35.3.18 vTaskPriorityInherit() void vTaskPriorityInherit (
 xTaskHandle *const pxMutexHolder)

22.35.3.19 vTaskPrioritySet() void vTaskPrioritySet (
 xTaskHandle pxTask,
 unsigned portBASE_TYPE uxNewPriority)

22.35.3.20 vTaskResume() void vTaskResume (
 xTaskHandle pxTaskToResume)

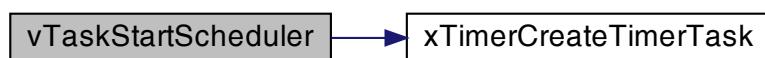
22.35.3.21 vTaskSetTimeOutState() void vTaskSetTimeOutState (
 xTimeOutType *const pxTimeOut)

Definition at line 1840 of file [tasks.c](#).

22.35.3.22 vTaskStartScheduler() void vTaskStartScheduler (
 void)

Definition at line 1078 of file [tasks.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



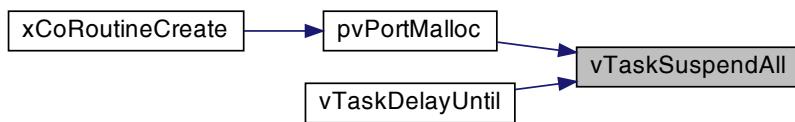
```
22.35.3.23 vTaskStartTrace() void vTaskStartTrace (
    signed char * pcBuffer,
    unsigned long ulBufferSize )
```

```
22.35.3.24 vTaskSuspend() void vTaskSuspend (
    xTaskHandle pxTaskToSuspend )
```

```
22.35.3.25 vTaskSuspendAll() void vTaskSuspendAll (
    void )
```

Definition at line 1154 of file [tasks.c](#).

Here is the caller graph for this function:



```
22.35.3.26 vTaskSwitchContext() void vTaskSwitchContext (
    void )
```

Definition at line 1655 of file [tasks.c](#).

```
22.35.3.27 xTaskCallApplicationTaskHook() portBASE_TYPE xTaskCallApplicationTaskHook (
    xTaskHandle xTask,
    void * pvParameter )
```

[task.h](#)

```
portBASE_TYPE xTaskCallApplicationTaskHook( xTaskHandle xTask, pdTASK_HOOK_CODE pxHookFunction );
```

Calls the hook function associated with xTask. Passing xTask as NULL has the effect of calling the Running tasks (the calling task) hook function.

pvParameter is passed to the hook function for the task to interpret as it wants.

```
22.35.3.28 xTaskCheckForTimeOut() portBASE_TYPE xTaskCheckForTimeOut (
    xTimeOutType *const pxTimeOut,
    portTickType *const pxTicksToWait )
```

Definition at line 1848 of file [tasks.c](#).

```
22.35.3.29 xTaskGenericCreate() signed portBASE_TYPE xTaskGenericCreate (
    pdTASK_CODE pxTaskCode,
    const signed char *const pcName,
    unsigned short usStackDepth,
    void * pvParameters,
    unsigned portBASE_TYPE uxPriority,
    xTaskHandle * pxCreatedTask,
    portSTACK_TYPE * puxStackBuffer,
    const xMemoryRegion *const xRegions )
```

```
22.35.3.30 xTaskGetCurrentTaskHandle() xTaskHandle xTaskGetCurrentTaskHandle (
    void )
```

```
22.35.3.31 xTaskGetIdleTaskHandle() xTaskHandle xTaskGetIdleTaskHandle (
    void )
```

`xTaskGetIdleTaskHandle()` is only available if `INCLUDE_xTaskGetIdleTaskHandle` is set to 1 in [FreeRTOSConfig.h](#).

Simply returns the handle of the idle task. It is not valid to call `xTaskGetIdleTaskHandle()` before the scheduler has been started.

```
22.35.3.32 xTaskGetSchedulerState() portBASE_TYPE xTaskGetSchedulerState (
    void )
```

```
22.35.3.33 xTaskGetTickCount() portTickType xTaskGetTickCount (
    void )
```

Definition at line 1249 of file [tasks.c](#).

```
22.35.3.34 xTaskGetTickCountFromISR() portTickType xTaskGetTickCountFromISR (
    void )
```

Definition at line 1264 of file [tasks.c](#).

22.35.3.35 xTaskIsTaskSuspended() signed `portBASE_TYPE xTaskIsTaskSuspended(xTaskHandle xTask)`

task.h

```
signed portBASE_TYPE xTaskIsTaskSuspended( xTaskHandle xTask );
```

Utility task that simply returns pdTRUE if the task referenced by xTask is currently in the Suspended state, or pdFALSE if the task referenced by xTask is in any other state.

22.35.3.36 xTaskRemoveFromEventList() signed `portBASE_TYPE xTaskRemoveFromEventList(const xList *const pxEventList)`

Definition at line 1789 of file [tasks.c](#).

22.35.3.37 xTaskResumeAll() signed `portBASE_TYPE xTaskResumeAll(void)`

Definition at line 1162 of file [tasks.c](#).

22.35.3.38 xTaskResumeFromISR() `portBASE_TYPE xTaskResumeFromISR(xTaskHandle pxTaskToResume)`

22.36 task.h

```
00001 /*
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005 ****
00006 *
00007 *   FreeRTOS tutorial books are available in pdf and paperback.
00008 *   Complete, revised, and edited pdf reference manuals are also
00009 *   available.
00010 *
00011 *   Purchasing FreeRTOS documentation will not only help you, by
00012 *   ensuring you get running as quickly as possible and with an
00013 *   in-depth knowledge of how to use FreeRTOS, it will also help
00014 *   the FreeRTOS project to continue with its mission of providing
00015 *   professional grade, cross platform, de facto standard solutions
00016 *   for microcontrollers - completely free of charge!
00017 *
00018 *   >> See http://www.FreeRTOS.org/Documentation for details. <<
00019 *
00020 *   Thank you for using FreeRTOS, and thank you for your support!
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 >>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
```

```

00037     License and the FreeRTOS license exception along with FreeRTOS; if not it
00038     can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039     by writing to Richard Barry, contact details for whom are available on the
00040     FreeRTOS WEB site.
00041
00042     1 tab == 4 spaces!
00043
00044     http://www.FreeRTOS.org - Documentation, latest information, license and
00045     contact details.
00046
00047     http://www.SafeRTOS.com - A version that is certified for use in safety
00048     critical systems.
00049
00050     http://www.OpenRTOS.com - Commercial support, development, porting,
00051     licensing and training services.
00052 */
00053
00054
00055 #ifndef TASK_H
00056 #define TASK_H
00057
00058 #ifndef INC_FREERTOS_H
00059     #error "include FreeRTOS.h must appear in source files before include task.h"
00060 #endif
00061
00062 #include "portable.h"
00063 #include "list.h"
00064
00065 #ifdef __cplusplus
00066 extern "C" {
00067 #endif
00068
00069 /-----
00070 * MACROS AND DEFINITIONS
00071 -----*/
00072
00073 #define tskKERNEL_VERSION_NUMBER "V7.0.1"
00074
00075 typedef void * xTaskHandle;
00076
00077 /*
00078 * Used internally only.
00079 */
00080 typedef struct xTIME_OUT
00081 {
00082     portBASE_TYPE xOverflowCount;
00083     portTickType xTimeOnEntering;
00084 } xTimeOutType;
00085
00086 /*
00087 * Defines the memory ranges allocated to the task when an MPU is used.
00088 */
00089 typedef struct xMEMORY_REGION
00090 {
00091     void *pvBaseAddress;
00092     unsigned long ulLengthInBytes;
00093     unsigned long ulParameters;
00094 } xMemoryRegion;
00095
00096 /*
00097 * Parameters required to create an MPU protected task.
00098 */
00099 typedef struct xTASK_PARAMETERS
00100 {
00101     pdTASK_CODE pvTaskCode;
00102     const signed char * const pcName;
00103     unsigned short usStackDepth;
00104     void *pvParameters;
00105     unsigned portBASE_TYPE uxPriority;
00106     portSTACK_TYPE *pxuStackBuffer;
00107     xMemoryRegion xRegions[ portNUM_CONFIGURABLE_REGIONS ];
00108 } xTaskParameters;
00109
00110 /*
00111 * Defines the priority used by the idle task. This must not be modified.
00112 */
00113 /* \ingroup TaskUtils
00114 */
00115 #define tskIDLE_PRIORITY      ( ( unsigned portBASE_TYPE ) 0U )
00116
00117 #define taskYIELD()           portYIELD()
00118
00119 #define taskENTER_CRITICAL()  portENTER_CRITICAL()
00120
00121 #define taskEXIT_CRITICAL()   portEXIT_CRITICAL()
00122
00123 #define taskDISABLE_INTERRUPTS() portDISABLE_INTERRUPTS()

```

```
00174
00183 #define taskENABLE_INTERRUPTS()      portENABLE_INTERRUPTS()
00184
00185 /* Definitions returned by xTaskGetSchedulerState(). */
00186 #define taskSCHEDULER_NOT_STARTED    0
00187 #define taskSCHEDULER_RUNNING       1
00188 #define taskSCHEDULER_SUSPENDED     2
00189
00190 /-----
00191 * TASK CREATION API
00192 -----*/
00193
00270 #define xTaskCreate( pvTaskCode, pcName, usStackDepth, pvParameters, uxPriority, pxCreatedTask )
    xTaskGenericCreate( (pvTaskCode), (pcName), (usStackDepth), (pvParameters), (uxPriority), (pxCreatedTask),
                        (NULL), (NULL) )
00271
00339 #define xTaskCreateRestricted( x, pxCreatedTask ) xTaskGenericCreate( ((x)->pvTaskCode),
    ((x)->pcName), ((x)->usStackDepth), ((x)->pvParameters), ((x)->uxPriority), (pxCreatedTask),
    ((x)->puxStackBuffer), ((x)->xRegions) )
00340
00387 void vTaskAllocateMPURegions( xTaskHandle xTask, const xMemoryRegion * const pxRegions )
    PRIVILEGED_FUNCTION;
00388
00428 void vTaskDelete( xTaskHandle pxTaskToDelete ) PRIVILEGED_FUNCTION;
00429
00430 /-----
00431 * TASK CONTROL API
00432 -----*/
00433
00482 void vTaskDelay( portTickType xTicksToDelay ) PRIVILEGED_FUNCTION;
00483
00541 void vTaskDelayUntil( portTickType * const pxPreviousWakeTime, portTickType xTimeIncrement )
    PRIVILEGED_FUNCTION;
00542
00588 unsigned portBASE_TYPE uxTaskPriorityGet( xTaskHandle pxTask ) PRIVILEGED_FUNCTION;
00589
00630 void vTaskPrioritySet( xTaskHandle pxTask, unsigned portBASE_TYPE uxNewPriority ) PRIVILEGED_FUNCTION;
00631
00681 void vTaskSuspend( xTaskHandle pxTaskToSuspend ) PRIVILEGED_FUNCTION;
00682
00730 void vTaskResume( xTaskHandle pxTaskToResume ) PRIVILEGED_FUNCTION;
00731
00750 portBASE_TYPE xTaskResumeFromISR( xTaskHandle pxTaskToResume ) PRIVILEGED_FUNCTION;
00751
00752 /-----
00753 * SCHEDULER CONTROL
00754 -----*/
00755
00788 void vTaskStartScheduler( void ) PRIVILEGED_FUNCTION;
00789
00841 void vTaskEndScheduler( void ) PRIVILEGED_FUNCTION;
00842
00892 void vTaskSuspendAll( void ) PRIVILEGED_FUNCTION;
00893
00944 signed portBASE_TYPE xTaskResumeAll( void ) PRIVILEGED_FUNCTION;
00945
00955 signed portBASE_TYPE xTaskIsTaskSuspended( xTaskHandle xTask ) PRIVILEGED_FUNCTION;
00956
00957 /-----
00958 * TASK UTILITIES
00959 -----*/
00960
00970 portTickType xTaskGetTickCount( void ) PRIVILEGED_FUNCTION;
00971
00986 portTickType xTaskGetTickCountFromISR( void ) PRIVILEGED_FUNCTION;
00987
01000 unsigned portBASE_TYPE uxTaskGetNumberOfTasks( void ) PRIVILEGED_FUNCTION;
01001
01014 signed char *pcTaskGetTaskName( xTaskHandle xTaskToQuery );
01015
01040 void vTaskList( signed char *pcWriteBuffer ) PRIVILEGED_FUNCTION;
01041
01072 void vTaskGetRunTimeStats( signed char *pcWriteBuffer ) PRIVILEGED_FUNCTION;
01073
01093 void vTaskStartTrace( signed char * pcBuffer, unsigned long ulBufferSize ) PRIVILEGED_FUNCTION;
01094
01106 unsigned long ulTaskEndTrace( void ) PRIVILEGED_FUNCTION;
01107
01126 unsigned portBASE_TYPE uxTaskGetStackHighWaterMark( xTaskHandle xTask ) PRIVILEGED_FUNCTION;
01127
01128 /* When using trace macros it is sometimes necessary to include tasks.h before
01129 FreeRTOS.h. When this is done pdTASK_HOOK_CODE will not yet have been defined,
01130 so the following two prototypes will cause a compilation error. This can be
01131 fixed by simply guarding against the inclusion of these two prototypes unless
01132 they are explicitly required by the configUSE_APPLICATION_TASK_TAG configuration
01133 constant. */
01134 #ifdef configUSE_APPLICATION_TASK_TAG
```

```

01135     #if configUSE_APPLICATION_TASK_TAG == 1
01144         void vTaskSetApplicationTaskTag( xTaskHandle xTask, pdTASK_HOOK_CODE pxHookFunction )
01145             PRIVILEGED_FUNCTION;
01146         pdTASK_HOOK_CODE xTaskGetApplicationTaskTag( xTaskHandle xTask ) PRIVILEGED_FUNCTION;
01147     #endif /* configUSE_APPLICATION_TASK_TAG ==1 */
01148 #endif /* ifdef configUSE_APPLICATION_TASK_TAG */
01149
01150 portBASE_TYPE xTaskCallApplicationTaskHook( xTaskHandle xTask, void *pvParameter )
01151     PRIVILEGED_FUNCTION;
01152
01153 xTaskHandle xTaskGetIdleTaskHandle( void );
01154
01155 /*-----
01156  * SCHEDULER INTERNALS AVAILABLE FOR PORTING PURPOSES
01157  *-----*/
01158
01159 /*
01160  * THIS FUNCTION MUST NOT BE USED FROM APPLICATION CODE. IT IS ONLY
01161  * INTENDED FOR USE WHEN IMPLEMENTING A PORT OF THE SCHEDULER AND IS
01162  * AN INTERFACE WHICH IS FOR THE EXCLUSIVE USE OF THE SCHEDULER.
01163  *
01164  * Called from the real time kernel tick (either preemptive or cooperative),
01165  * this increments the tick count and checks if any tasks that are blocked
01166  * for a finite period required removing from a blocked list and placing on
01167  * a ready list.
01168  */
01169 void vTaskIncrementTick( void ) PRIVILEGED_FUNCTION;
01170
01171 /*
01172  * THIS FUNCTION MUST NOT BE USED FROM APPLICATION CODE. IT IS AN
01173  * INTERFACE WHICH IS FOR THE EXCLUSIVE USE OF THE SCHEDULER.
01174  *
01175  * THIS FUNCTION MUST BE CALLED WITH INTERRUPTS DISABLED.
01176  *
01177  * Removes the calling task from the ready list and places it both
01178  * on the list of tasks waiting for a particular event, and the
01179  * list of delayed tasks. The task will be removed from both lists
01180  * and replaced on the ready list should either the event occur (and
01181  * there be no higher priority tasks waiting on the same event) or
01182  * the delay period expires.
01183  *
01184  * @param pxEventList The list containing tasks that are blocked waiting
01185  * for the event to occur.
01186  *
01187  * @param xTicksToWait The maximum amount of time that the task should wait
01188  * for the event to occur. This is specified in kernel ticks, the constant
01189  * portTICK_RATE_MS can be used to convert kernel ticks into a real time
01190  * period.
01191  */
01192 void vTaskPlaceOnEventList( const xList * const pxEventList, portTickType xTicksToWait )
01193     PRIVILEGED_FUNCTION;
01194
01195 /*
01196  * THIS FUNCTION MUST NOT BE USED FROM APPLICATION CODE. IT IS AN
01197  * INTERFACE WHICH IS FOR THE EXCLUSIVE USE OF THE SCHEDULER.
01198  *
01199  * THIS FUNCTION MUST BE CALLED WITH INTERRUPTS DISABLED.
01200  *
01201  * This function performs nearly the same function as vTaskPlaceOnEventList().
01202  * The difference being that this function does not permit tasks to block
01203  * indefinitely, whereas vTaskPlaceOnEventList() does.
01204  *
01205  * @return pdTRUE if the task being removed has a higher priority than the task
01206  * making the call, otherwise pdFALSE.
01207  */
01208 void vTaskPlaceOnEventListRestricted( const xList * const pxEventList, portTickType xTicksToWait )
01209     PRIVILEGED_FUNCTION;
01210
01211 /*
01212  * THIS FUNCTION MUST NOT BE USED FROM APPLICATION CODE. IT IS AN
01213  * INTERFACE WHICH IS FOR THE EXCLUSIVE USE OF THE SCHEDULER.
01214  *
01215  * THIS FUNCTION MUST BE CALLED WITH INTERRUPTS DISABLED.
01216  *
01217  * Removes a task from both the specified event list and the list of blocked
01218  * tasks, and places it on a ready queue.
01219  *
01220  * xTaskRemoveFromEventList () will be called if either an event occurs to
01221  * unblock a task, or the block timeout period expires.
01222  *
01223  * @return pdTRUE if the task being removed has a higher priority than the task
01224  * making the call, otherwise pdFALSE.
01225  */
01226 signed portBASE_TYPE xTaskRemoveFromEventList( const xList * const pxEventList ) PRIVILEGED_FUNCTION;
01227
01228 /*

```

```

01249 * THIS FUNCTION MUST NOT BE USED FROM APPLICATION CODE. IT IS ONLY
01250 * INTENDED FOR USE WHEN IMPLEMENTING A PORT OF THE SCHEDULER AND IS
01251 * AN INTERFACE WHICH IS FOR THE EXCLUSIVE USE OF THE SCHEDULER.
01252 *
01253 * Sets the pointer to the current TCB to the TCB of the highest priority task
01254 * that is ready to run.
01255 */
01256 void vTaskSwitchContext( void ) PRIVILEGED_FUNCTION;
01257 /*
01258 * Return the handle of the calling task.
01259 */
01260 xTaskHandle xTaskGetCurrentTaskHandle( void ) PRIVILEGED_FUNCTION;
01261 /*
01262 * Capture the current time status for future reference.
01263 */
01264 void vTaskSetTimeOutState( xTimeOutType * const pxTimeOut ) PRIVILEGED_FUNCTION;
01265 /*
01266 * Compare the time status now with that previously captured to see if the
01267 * timeout has expired.
01268 */
01269 portBASE_TYPE xTaskCheckForTimeOut( xTimeOutType * const pxTimeOut, portTickType * const pxTicksToWait
01270 ) PRIVILEGED_FUNCTION;
01271 /*
01272 * Shortcut used by the queue implementation to prevent unnecessary call to
01273 * taskYIELD();
01274 */
01275 void vTaskMissedYield( void ) PRIVILEGED_FUNCTION;
01276 /*
01277 * Returns the scheduler state as taskSCHEDULER_RUNNING,
01278 * taskSCHEDULER_NOT_STARTED or taskSCHEDULER_SUSPENDED.
01279 */
01280 portBASE_TYPE xTaskGetSchedulerState( void ) PRIVILEGED_FUNCTION;
01281 /*
01282 * Raises the priority of the mutex holder to that of the calling task should
01283 * the mutex holder have a priority less than the calling task.
01284 */
01285 void vTaskPriorityInherit( xTaskHandle * const pxMutexHolder ) PRIVILEGED_FUNCTION;
01286 /*
01287 * Set the priority of a task back to its proper priority in the case that it
01288 * inherited a higher priority while it was holding a semaphore.
01289 */
01290 void vTaskPriorityDisinherit( xTaskHandle * const pxMutexHolder ) PRIVILEGED_FUNCTION;
01291 /*
01292 * Generic version of the task creation function which is in turn called by the
01293 * xTaskCreate() and xTaskCreateRestricted() macros.
01294 */
01295 signed portBASE_TYPE xTaskGenericCreate( pdTASK_CODE pxTaskCode, const signed char * const pcName,
01296 unsigned short usStackDepth, void *pvParameters, unsigned portBASE_TYPE uxPriority, xTaskHandle
01297 *pxCreatedTask, portSTACK_TYPE *pxuStackBuffer, const xMemoryRegion * const xRegions )
01298 PRIVILEGED_FUNCTION;
01299 #ifdef __cplusplus
01300 }
01301#endif
01302 #endif /* TASK_H */
01303
01304 #endif
01305
01306 #endif
01307 #endif /* TASK_H */
01308
01309
01310

```

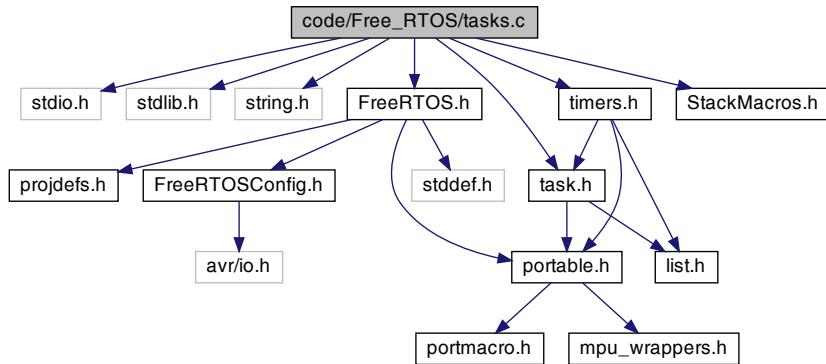
22.37 code/FreeRTOS/tasks.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "FreeRTOS.h"
#include "task.h"
#include "timers.h"
#include "StackMacros.h"

```

Include dependency graph for tasks.c:



Data Structures

- struct `tskTaskControlBlock`

Macros

- `#define MPU_WRAPPERS_INCLUDED_FROM_API_FILE`
- `#define tskIDLE_STACK_SIZE configMINIMAL_STACK_SIZE`
- `#define tskSTACK_FILL_BYTE (0xa5U)`
- `#define tskBLOCKED_CHAR ((signed char) 'B')`
- `#define tskREADY_CHAR ((signed char) 'R')`
- `#define tskDELETED_CHAR ((signed char) 'D')`
- `#define tskSUSPENDED_CHAR ((signed char) 'S')`
- `#define vWriteTraceToBuffer()`
- `#define prvAddTaskToReadyQueue(pxTCB)`
- `#define prvCheckDelayedTasks()`
- `#define prvGetTCBFromHandle(pxHandle) ((pxHandle) == NULL) ? (tskTCB *) pxCurrentTCB : (tskTCB *) (pxHandle)`

Typedefs

- typedef struct `tskTaskControlBlock` `tskTCB`

Functions

- `void vApplicationStackOverflowHook (xTaskHandle *pxTask, signed char *pcTaskName)`
- `void vApplicationTickHook (void)`
- `void vTaskDelete (xTaskHandle pxTaskToDelete)`
- `void vTaskDelayUntil (portTickType *const pxPreviousWakeTime, portTickType xTimeIncrement)`
- `void vTaskDelay (portTickType xTicksToDelay)`
- `void vTaskStartScheduler (void)`
- `void vTaskEndScheduler (void)`
- `void vTaskSuspendAll (void)`
- `signed portBASE_TYPE xTaskResumeAll (void)`

- `portTickType xTaskGetTickCount (void)`
- `portTickType xTaskGetTickCountFromISR (void)`
- `unsigned portBASE_TYPE uxTaskGetNumberOfTasks (void)`
- `void vTaskIncrementTick (void)`
- `void vTaskSwitchContext (void)`
- `void vTaskPlaceOnEventList (const xList *const pxEventList, portTickType xTicksToWait)`
- `signed portBASE_TYPE xTaskRemoveFromEventList (const xList *const pxEventList)`
- `void vTaskSetTimeOutState (xTimeOutType *const pxTimeOut)`
- `portBASE_TYPE xTaskCheckForTimeOut (xTimeOutType *const pxTimeOut, portTickType *const pxTicks←ToWait)`
- `void vTaskMissedYield (void)`

Variables

- `PRIVILEGED_DATA tskTCB *volatile pxCurrentTCB = NULL`

22.37.1 Macro Definition Documentation

22.37.1.1 MPU_WRAPPERS_INCLUDED_FROM_API_FILE `#define MPU_WRAPPERS_INCLUDED_FROM_API_←FILE`

Definition at line 62 of file [tasks.c](#).

22.37.1.2 prvAddTaskToReadyQueue `#define prvAddTaskToReadyQueue(`
`pxTCB)`

Value:

```
if( ( pxTCB )->uxPriority > uxTopReadyPriority )
{
    uxTopReadyPriority = ( pxTCB )->uxPriority;
}
vListInsertEnd( ( xList * ) &( pxReadyTasksLists[ ( pxTCB )->uxPriority ] ), &( ( pxTCB
)->xGenericListItem ) )
```

Definition at line 258 of file [tasks.c](#).

22.37.1.3 prvCheckDelayedTasks `#define prvCheckDelayedTasks()`

Definition at line 274 of file [tasks.c](#).

22.37.1.4 prvGetTCBFromHandle #define prvGetTCBFromHandle(pxHandle) (((pxHandle) == NULL) ? (tskTCB *) pxCurrentTCB : (tskTCB *) (pxHandle))

Definition at line 323 of file [tasks.c](#).

22.37.1.5 tskBLOCKED_CHAR #define tskBLOCKED_CHAR ((signed char) 'B')

Definition at line 192 of file [tasks.c](#).

22.37.1.6 tskDELETED_CHAR #define tskDELETED_CHAR ((signed char) 'D')

Definition at line 194 of file [tasks.c](#).

22.37.1.7 tskIDLE_STACK_SIZE #define tskIDLE_STACK_SIZE configMINIMAL_STACK_SIZE

Definition at line 74 of file [tasks.c](#).

22.37.1.8 tskREADY_CHAR #define tskREADY_CHAR ((signed char) 'R')

Definition at line 193 of file [tasks.c](#).

22.37.1.9 tskSTACK_FILL_BYTE #define tskSTACK_FILL_BYTE (0xa5U)

Definition at line 187 of file [tasks.c](#).

22.37.1.10 tskSUSPENDED_CHAR #define tskSUSPENDED_CHAR ((signed char) 'S')

Definition at line 195 of file [tasks.c](#).

22.37.1.11 vWriteTraceToBuffer #define vWriteTraceToBuffer()

Definition at line 246 of file [tasks.c](#).

22.37.2 Typedef Documentation

22.37.2.1 `tskTCB` `typedef struct tskTaskControlBlock tskTCB`

22.37.3 Function Documentation

22.37.3.1 `uxTaskGetNumberOfTasks()` `unsigned portBASE_TYPE uxTaskGetNumberOfTasks (void)`

Definition at line 1277 of file [tasks.c](#).

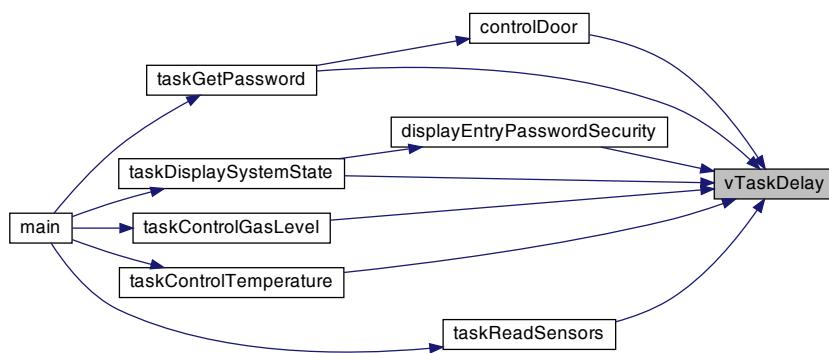
22.37.3.2 `vApplicationStackOverflowHook()` `void vApplicationStackOverflowHook (xTaskHandle * pxTask, signed char * pcTaskName)`

22.37.3.3 `vApplicationTickHook()` `void vApplicationTickHook (void)`

22.37.3.4 `vTaskDelay()` `void vTaskDelay (portTickType xTicksToDelay)`

Definition at line 718 of file [tasks.c](#).

Here is the caller graph for this function:



```
22.37.3.5 vTaskDelayUntil() void vTaskDelayUntil (
    portTickType *const pxPreviousWakeTime,
    portTickType xTimeIncrement )
```

Definition at line 653 of file [tasks.c](#).

Here is the call graph for this function:



```
22.37.3.6 vTaskDelete() void vTaskDelete (
    xTaskHandle pxTaskToDelete )
```

Definition at line 587 of file [tasks.c](#).

```
22.37.3.7 vTaskEndScheduler() void vTaskEndScheduler (
    void )
```

Definition at line 1143 of file [tasks.c](#).

```
22.37.3.8 vTaskIncrementTick() void vTaskIncrementTick (
    void )
```

Definition at line 1493 of file [tasks.c](#).

```
22.37.3.9 vTaskMissedYield() void vTaskMissedYield (
    void )
```

Definition at line 1894 of file [tasks.c](#).

```
22.37.3.10 vTaskPlaceOnEventList() void vTaskPlaceOnEventList ( const xList *const pxEventList, portTickType xTicksToWait )
```

Definition at line 1707 of file [tasks.c](#).

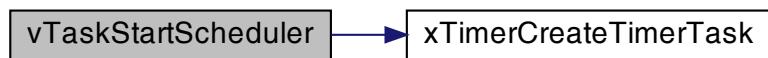
```
22.37.3.11 vTaskSetTimeOutState() void vTaskSetTimeOutState ( xTimeOutType *const pxTimeOut )
```

Definition at line 1840 of file [tasks.c](#).

```
22.37.3.12 vTaskStartScheduler() void vTaskStartScheduler ( void )
```

Definition at line 1078 of file [tasks.c](#).

Here is the call graph for this function:



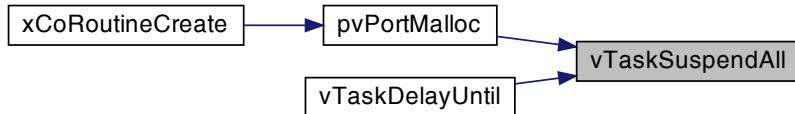
Here is the caller graph for this function:



```
22.37.3.13 vTaskSuspendAll() void vTaskSuspendAll (
    void )
```

Definition at line 1154 of file [tasks.c](#).

Here is the caller graph for this function:



```
22.37.3.14 vTaskSwitchContext() void vTaskSwitchContext (
    void )
```

Definition at line 1655 of file [tasks.c](#).

```
22.37.3.15 xTaskCheckForTimeOut() portBASE_TYPE xTaskCheckForTimeOut (
    xTimeOutType *const pxTimeOut,
    portTickType *const pxTicksToWait )
```

Definition at line 1848 of file [tasks.c](#).

```
22.37.3.16 xTaskGetTickCount() portTickType xTaskGetTickCount (
    void )
```

Definition at line 1249 of file [tasks.c](#).

```
22.37.3.17 xTaskGetTickCountFromISR() portTickType xTaskGetTickCountFromISR (
    void )
```

Definition at line 1264 of file [tasks.c](#).

```
22.37.3.18 xTaskRemoveFromEventList() signed portBASE_TYPE xTaskRemoveFromEventList (
    const xList *const pxEventList )
```

Definition at line 1789 of file [tasks.c](#).

```
22.37.3.19 xTaskResumeAll() signed portBASE_TYPE xTaskResumeAll (
    void )
```

Definition at line 1162 of file [tasks.c](#).

22.37.4 Variable Documentation

22.37.4.1 pxCurrentTCB `PRIVILEGED_DATA tskTCB* volatile pxCurrentTCB = NULL`

Definition at line 130 of file [tasks.c](#).

22.38 tasks.c

```
00001 /*
00002     FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005 ****
00006 *
00007 *     FreeRTOS tutorial books are available in pdf and paperback.
00008 *     Complete, revised, and edited pdf reference manuals are also
00009 *     available.
00010 *
00011 *     Purchasing FreeRTOS documentation will not only help you, by
00012 *     ensuring you get running as quickly as possible and with an
00013 *     in-depth knowledge of how to use FreeRTOS, it will also help
00014 *     the FreeRTOS project to continue with its mission of providing
00015 *     professional grade, cross platform, de facto standard solutions
00016 *     for microcontrollers - completely free of charge!
00017 *
00018 *     »> See http://www.FreeRTOS.org/Documentation for details. «<
00019 *
00020 *     Thank you for using FreeRTOS, and thank you for your support!
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 »>NOTE«< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039 by writing to Richard Barry, contact details for whom are available on the
00040 FreeRTOS WEB site.
00041
00042 1 tab == 4 spaces!
00043
00044 http://www.FreeRTOS.org - Documentation, latest information, license and
00045 contact details.
00046
00047 http://www.SafeRTOS.com - A version that is certified for use in safety
00048 critical systems.
00049
00050 http://www.OpenRTOS.com - Commercial support, development, porting,
00051 licensing and training services.
00052 */
00053
00054
00055 #include <stdio.h>
00056 #include <stdlib.h>
00057 #include <string.h>
00058
00059 /* Defining MPU_WRAPPERS_INCLUDED_FROM_API_FILE prevents task.h from redefining
```

```

00060 all the API functions to use the MPU wrappers. That should only be done when
00061 task.h is included from an application file. */
00062 #define MPU_WRAPPERS_INCLUDED_FROM_API_FILE
00063
00064 #include "FreeRTOS.h"
00065 #include "task.h"
00066 #include "timers.h"
00067 #include "StackMacros.h"
00068
00069 #undef MPU_WRAPPERS_INCLUDED_FROM_API_FILE
00070
00071 /*
00072  * Macro to define the amount of stack available to the idle task.
00073  */
00074 #define tskIDLE_STACK_SIZE configMINIMAL_STACK_SIZE
00075
00076 /*
00077  * Task control block. A task control block (TCB) is allocated to each task,
00078  * and stores the context of the task.
00079  */
00080 typedef struct tskTaskControlBlock
00081 {
00082     volatile portSTACK_TYPE *pxTopOfStack;      /*< Points to the location of the last item placed
00083     on the tasks stack. THIS MUST BE THE FIRST MEMBER OF THE STRUCT. */
00084
00085     #if ( portUSING MPU_WRAPPERS == 1 )
00086         xMPU_SETTINGS xMPUSettings;           /*< The MPU settings are defined as part of the port
00087         layer. THIS MUST BE THE SECOND MEMBER OF THE STRUCT. */
00088     #endif
00089
00090     xListItem           xGenericListItem;    /*< List item used to place the TCB in ready and
00091     blocked queues. */
00092     xListItem           xEventListItem;       /*< List item used to place the TCB in event lists.
00093     */
00094     unsigned portBASE_TYPE uxPriority;        /*< The priority of the task where 0 is the lowest
00095     priority. */
00096     portSTACK_TYPE      *pxStack;             /*< Points to the start of the stack. */
00097     signed char          pcTaskName[ configMAX_TASK_NAME_LEN ];/*< Descriptive name given to the
00098     task when created. Facilitates debugging only. */
00099
00100     #if ( portSTACK_GROWTH > 0 )
00101         portSTACK_TYPE *pxEndOfStack;        /*< Used for stack overflow checking on architectures
00102         where the stack grows up from low memory. */
00103     #endif
00104
00105     #if ( portCRITICAL_NESTING_IN_TCB == 1 )
00106         unsigned portBASE_TYPE uxCriticalNesting;
00107     #endif
00108
00109     #if ( configUSE_TRACE_FACILITY == 1 )
00110         unsigned portBASE_TYPE uxCBNumber;    /*< This is used for tracing the scheduler and making
00111         debugging easier only. */
00112     #endif
00113
00114     #if ( configUSE_MUTEXES == 1 )
00115         unsigned portBASE_TYPE uxBasePriority; /*< The priority last assigned to the task - used by
00116         the priority inheritance mechanism. */
00117     #endif
00118 } tskTCB;
00119
00120
00121 /*
00122  * Some kernel aware debuggers require data to be viewed to be global, rather
00123  * than file scope.
00124 */
00125 #ifdef portREMOVE_STATIC_QUALIFIER
00126     #define static
00127 #endif
00128
00129 /*lint -e956 */
00130 PRIVILEGED_DATA tskTCB * volatile pxCurrentTCB = NULL;
00131
00132 /* Lists for ready and blocked tasks. -----*/
00133
00134 PRIVILEGED_DATA static xList pxReadyTasksLists[ configMAX_PRIORITIES ]; /*< Prioritised ready tasks.
00135 */
00136 PRIVILEGED_DATA static xList xDelayedTaskList1;                           /*< Delayed tasks. */

```

```

00136 PRIVILEGED_DATA static xList xDelayedTaskList2;                                /*< Delayed tasks (two lists
00137     are used - one for delays that have overflowed the current tick count. */
00138 PRIVILEGED_DATA static xList * volatile pxDelayedTaskList;                      /*< Points to the delayed task
00139     list currently being used. */
00140 PRIVILEGED_DATA static xList * volatile pxOverflowDelayedTaskList;             /*< Points to the delayed task
00141     list currently being used to hold tasks that have overflowed the current tick count. */
00142 PRIVILEGED_DATA static xList xPendingReadyList;                                 /*< Tasks that have been
00143     readied while the scheduler was suspended. They will be moved to the ready queue when the scheduler
00144     is resumed. */
00145
00146 #if ( INCLUDE_vTaskDelete == 1 )
00147
00148     PRIVILEGED_DATA static xList xTasksWaitingTermination;                     /*< Tasks that have been
00149     deleted - but the their memory not yet freed. */
00150     PRIVILEGED_DATA static volatile unsigned portBASE_TYPE uxTasksDeleted = ( unsigned portBASE_TYPE )
00151     0U;
00152 #endif
00153
00154 #if ( INCLUDE_xTaskGetIdleTaskHandle == 1 )
00155
00156     PRIVILEGED_DATA static xTaskHandle xIdleTaskHandle = NULL;
00157
00158 #endif
00159
00160 /* File private variables. -----*/
00161 PRIVILEGED_DATA static volatile unsigned portBASE_TYPE uxCurrentNumberOfTasks = ( unsigned
00162     portBASE_TYPE ) 0U;
00163 PRIVILEGED_DATA static volatile portTickType xTickCount = ( portTickType ) 0U;
00164 PRIVILEGED_DATA static unsigned portBASE_TYPE uxTopUsedPriority = tskIDLE_PRIORITY;
00165 PRIVILEGED_DATA static volatile unsigned portBASE_TYPE uxTopReadyPriority = tskIDLE_PRIORITY;
00166 PRIVILEGED_DATA static volatile signed portBASE_TYPE xSchedulerRunning = pdFALSE;
00167 PRIVILEGED_DATA static volatile unsigned portBASE_TYPE uxSchedulerSuspended = ( unsigned
00168     portBASE_TYPE ) pdFALSE;
00169 PRIVILEGED_DATA static volatile unsigned portBASE_TYPE uxMissedTicks = ( unsigned
00170     portBASE_TYPE ) 0U;
00171 PRIVILEGED_DATA static volatile portBASE_TYPE xMissedYield = ( portBASE_TYPE )
00172     pdFALSE;
00173 PRIVILEGED_DATA static volatile portBASE_TYPE xNumOfOverflows = ( portBASE_TYPE ) 0;
00174 PRIVILEGED_DATA static unsigned portBASE_TYPE uxTaskNumber = ( unsigned
00175     portBASE_TYPE ) 0U;
00176 PRIVILEGED_DATA static portTickType xNextTaskUnblockTime = ( portTickType )
00177     portMAX_DELAY;
00178
00179 #if ( configGENERATE_RUN_TIME_STATS == 1 )
00180
00181     PRIVILEGED_DATA static char pcStatsString[ 50 ];
00182     PRIVILEGED_DATA static unsigned long ulTaskSwitchedInTime = 0UL; /*< Holds the value of a
00183         timer/counter the last time a task was switched in. */
00184     static void prvGenerateRunTimeStatsForTasksInList( const signed char *pcWriteBuffer, xList
00185         *pxList, unsigned long ulTotalRunTime ) PRIVILEGED_FUNCTION;
00186
00187 #endif
00188
00189 /* Debugging and trace facilities private variables and macros. -----*/
00190
00191 /* The value used to fill the stack of a task when the task is created. This
00192 * is used purely for checking the high water mark for tasks.
00193 */
00194 #define tskSTACK_FILL_BYTE ( 0xa5U )
00195
00196 /*
00197 * Macros used by vListTask to indicate which state a task is in.
00198 */
00199 #define tskBLOCKED_CHAR      ( ( signed char ) 'B' )
00200 #define tskREADY_CHAR        ( ( signed char ) 'R' )
00201 #define tskDELETED_CHAR      ( ( signed char ) 'D' )
00202 #define tskSUSPENDED_CHAR    ( ( signed char ) 'S' )
00203
00204 /*
00205 * Macros and private variables used by the trace facility.
00206 */
00207 #if ( configUSE_TRACE_FACILITY == 1 )
00208
00209     #define tskSIZE_OF_EACH_TRACE_LINE      ( ( unsigned long ) ( sizeof( unsigned long ) +
00210         sizeof( unsigned long ) ) )
00211     PRIVILEGED_DATA static volatile signed char * volatile pcTraceBuffer;
00212     PRIVILEGED_DATA static signed char *pcTraceBufferStart;
00213     PRIVILEGED_DATA static signed char *pcTraceBufferEnd;

```

```

00206     PRIVILEGED_DATA static signed portBASE_TYPE xTracing = pdFALSE;
00207     static unsigned portBASE_TYPE uxPreviousTask = 255U;
00208     PRIVILEGED_DATA static char pcStatusString[ 50 ];
00209
00210 #endif
00211
00212 /*-----*/
00213
00214 /*
00215  * Macro that writes a trace of scheduler activity to a buffer. This trace
00216  * shows which task is running when and is very useful as a debugging tool.
00217  * As this macro is called each context switch it is a good idea to undefine
00218  * it if not using the facility.
00219 */
00220 #if ( configUSE_TRACE_FACILITY == 1 )
00221
00222     #define vWriteTraceToBuffer()
00223     {
00224         if( xTracing != pdFALSE )
00225         {
00226             if( uxPreviousTask != pxCurrentTCB->uxTCBNumber )
00227             {
00228                 if( ( pcTraceBuffer + tskSIZE_OF_EACH_TRACE_LINE ) < pcTraceBufferEnd )
00229                 {
00230                     uxPreviousTask = pxCurrentTCB->uxTCBNumber;
00231                     *( unsigned long * ) pcTraceBuffer = ( unsigned long ) xTickCount;
00232                     pcTraceBuffer += sizeof( unsigned long );
00233                     *( unsigned long * ) pcTraceBuffer = ( unsigned long ) uxPreviousTask;
00234                     pcTraceBuffer += sizeof( unsigned long );
00235                 }
00236                 else
00237                 {
00238                     xTracing = pdFALSE;
00239                 }
00240             }
00241         }
00242     }
00243
00244 #else
00245
00246     #define vWriteTraceToBuffer()
00247
00248 #endif
00249 /*-----*/
00250
00251 /*
00252  * Place the task represented by pxTCB into the appropriate ready queue for
00253  * the task. It is inserted at the end of the list. One quirk of this is
00254  * that if the task being inserted is at the same priority as the currently
00255  * executing task, then it will only be rescheduled after the currently
00256  * executing task has been rescheduled.
00257 */
00258 #define prvAddTaskToReadyQueue( pxTCB )
00259     \
00260     if( ( pxTCB )->uxPriority > uxTopReadyPriority )
00261     \
00262     {
00263         uxTopReadyPriority = ( pxTCB )->uxPriority;
00264     \
00265         \
00266         vListInsertEnd( ( xList * ) &( pxReadyTasksLists[ ( pxTCB )->uxPriority ] ), &( ( pxTCB
00267 )->xGenericListItem ) )
00268 /*-----*/
00269
00270 /*
00271  * Macro that looks at the list of tasks that are currently delayed to see if
00272  * any require waking.
00273 */
00274 #define prvCheckDelayedTasks()
00275 {
00276     portTickType xItemValue;
00277
00278     /* Is the tick count greater than or equal to the wake time of the first
00279      task referenced from the delayed tasks list? */
00280     if( xTickCount >= xNextTaskUnblockTime )
00281     {
00282         for( ; ; )
00283         {
00284             if( listLIST_IS_EMPTY( pxDelayedTaskList ) != pdFALSE )
00285             {
00286                 /* The delayed list is empty. Set xNextTaskUnblockTime to the

```

```

00287             maximum possible value so it is extremely unlikely that the
00288             if( xTickCount >= xNextTaskUnblockTime ) test will pass next
00289             time through. */
00290             xNextTaskUnblockTime = portMAX_DELAY;
00291             break;
00292         }
00293     else
00294     {
00295         /* The delayed list is not empty, get the value of the item at
00296         the head of the delayed list. This is the time at which the
00297         task at the head of the delayed list should be removed from
00298         the Blocked state. */
00299         pxTCB = ( tskTCB * ) listGET_OWNER_OF_HEAD_ENTRY( pxDelayedTaskList );
00300         xItemValue = listGET_LIST_ITEM_VALUE( &( pxTCB->xGenericListItem ) );
00301
00302         if( xTickCount < xItemValue )
00303         {
00304             /* It is not time to unblock this item yet, but the item
00305             value is the time at which the task at the head of the
00306             blocked list should be removed from the Blocked state -
00307             so record the item value in xNextTaskUnblockTime. */
00308             xNextTaskUnblockTime = xItemValue;
00309             break;
00310         }
00311
00312         /* It is time to remove the item from the Blocked state. */
00313         vListRemove( &( pxTCB->xGenericListItem ) );
00314
00315         /* Is the task waiting on an event also? */
00316         if( pxTCB->xEventListItem.pvContainer != NULL )
00317         {
00318             vListRemove( &( pxTCB->xEventListItem ) );
00319         }
00320         prvAddTaskToReadyQueue( pxTCB );
00321     }
00322 }
00323 }
00324 }
00325 /***** */
00326
00327 /*
00328  * Several functions take an xTaskHandle parameter that can optionally be NULL,
00329  * where NULL is used to indicate that the handle of the currently executing
00330  * task should be used in place of the parameter. This macro simply checks to
00331  * see if the parameter is NULL and returns a pointer to the appropriate TCB.
00332 */
00333 #define prvGetTCBFromHandle( pxHandle ) ( ( pxHandle ) == NULL ) ? ( tskTCB * ) pxCurrentTCB : (
00334     tskTCB * ) ( pxHandle )
00335 /* Callback function prototypes. -----*/
00336 extern void vApplicationStackOverflowHook( xTaskHandle *pxTask, signed char *pcTaskName );
00337 extern void vApplicationTickHook( void );
00338
00339 /* File private functions. -----*/
00340
00341 /*
00342  * Utility to ready a TCB for a given task. Mainly just copies the parameters
00343  * into the TCB structure.
00344 */
00345 static void prvInitialiseTCBVariables( tskTCB *pxTCB, const signed char * const pcName, unsigned
00346                                         portBASE_TYPE uxPriority, const xMemoryRegion * const xRegions, unsigned short usStackDepth )
00347                                         PRIVILEGED_FUNCTION;
00348
00349 /*
00350  * Utility to ready all the lists used by the scheduler. This is called
00351  * automatically upon the creation of the first task.
00352 */
00353 static void prvInitialiseTaskLists( void ) PRIVILEGED_FUNCTION;
00354
00355 /*
00356  * The idle task, which as all tasks is implemented as a never ending loop.
00357  * The idle task is automatically created and added to the ready lists upon
00358  * creation of the first user task.
00359 */
00360
00361 /*
00362  * The portTASK_FUNCTION_PROTO() macro is used to allow port/compiler specific
00363  * language extensions. The equivalent prototype for this function is:
00364
00365  * void prvIdleTask( void *pvParameters );
00366 */
00367 static portTASK_FUNCTION_PROTO( prvIdleTask, pvParameters );
00368
00369 /*
00370  * Utility to free all memory allocated by the scheduler to hold a TCB,
00371  * including the stack pointed to by the TCB.
00372 */
00373
00374 /*
00375  * This does not free memory allocated by the task itself (i.e. memory

```

```

00371 * allocated by calls to pvPortMalloc from within the tasks application code).
00372 */
00373 #if ( INCLUDE_vTaskDelete == 1 )
00374
00375     static void prvDeleteTCB( tskTCB *pxTCB ) PRIVILEGED_FUNCTION;
00376
00377 #endif
00378
00379 /*
00380 * Used only by the idle task. This checks to see if anything has been placed
00381 * in the list of tasks waiting to be deleted. If so the task is cleaned up
00382 * and its TCB deleted.
00383 */
00384 static void prvCheckTasksWaitingTermination( void ) PRIVILEGED_FUNCTION;
00385
00386 /*
00387 * The currently executing task is entering the Blocked state. Add the task to
00388 * either the current or the overflow delayed task list.
00389 */
00390 static void prvAddCurrentTaskToDelayedList( portTickType xTimeToWake ) PRIVILEGED_FUNCTION;
00391
00392 /*
00393 * Allocates memory from the heap for a TCB and associated stack. Checks the
00394 * allocation was successful.
00395 */
00396 static tskTCB *prvAllocateTCBAndStack( unsigned short usStackDepth, portSTACK_TYPE *pxStackBuffer ) PRIVILEGED_FUNCTION;
00397
00398 /*
00399 * Called from vTaskList. vListTasks details all the tasks currently under
00400 * control of the scheduler. The tasks may be in one of a number of lists.
00401 * prvListTaskWithinSingleList accepts a list and details the tasks from
00402 * within just that list.
00403 *
00404 * THIS FUNCTION IS INTENDED FOR DEBUGGING ONLY, AND SHOULD NOT BE CALLED FROM
00405 * NORMAL APPLICATION CODE.
00406 */
00407 #if ( configUSE_TRACE_FACILITY == 1 )
00408
00409     static void prvListTaskWithinSingleList( const signed char *pcWriteBuffer, xList *pxList, signed
00410         char cStatus ) PRIVILEGED_FUNCTION;
00411 #endif
00412
00413 /*
00414 * When a task is created, the stack of the task is filled with a known value.
00415 * This function determines the 'high water mark' of the task stack by
00416 * determining how much of the stack remains at the original preset value.
00417 */
00418 #if ( ( configUSE_TRACE_FACILITY == 1 ) || ( INCLUDE uxTaskGetStackHighWaterMark == 1 ) )
00419
00420     static unsigned short usTaskCheckFreeStackSpace( const unsigned char * pucStackByte ) PRIVILEGED_FUNCTION;
00421
00422 #endif
00423
00424
00425 /*lint +e956 */
00426
00427
00428
00429 *-----*
00430 * TASK CREATION API documented in task.h
00431 *-----*/
00432
00433 signed portBASE_TYPE xTaskGenericCreate( pdTASK_CODE pxTaskCode, const signed char * const pcName,
00434     unsigned short usStackDepth, void *pvParameters, unsigned portBASE_TYPE uxPriority, xTaskHandle
00435     *pxCreatedTask, portSTACK_TYPE *pxStackBuffer, const xMemoryRegion * const xRegions )
00436 {
00437     signed portBASE_TYPE xReturn;
00438     tskTCB * pxNewTCB;
00439
00440     configASSERT( pxTaskCode );
00441     configASSERT( ( uxPriority < configMAX_PRIORITIES ) );
00442
00443     /* Allocate the memory required by the TCB and stack for the new task,
00444      checking that the allocation was successful. */
00445     pxNewTCB = prvAllocateTCBAndStack( usStackDepth, pxStackBuffer );
00446
00447     if( pxNewTCB != NULL )
00448     {
00449         portSTACK_TYPE *pxTopOfStack;
00450
00451         #if( portUSING_MPU_WRAPPERS == 1 )
00452             /* Should the task be created in privileged mode? */
00453             portBASE_TYPE xRunPrivileged;
00454             if( ( uxPriority & portPRIVILEGE_BIT ) != 0U )

```

```

00453         {
00454             xRunPrivileged = pdTRUE;
00455         }
00456     else
00457     {
00458         xRunPrivileged = pdFALSE;
00459     }
00460     uxPriority &= ~portPRIVILEGE_BIT;
00461 #endif /* portUSING_MPU_WRAPPERS == 1 */
00462
00463     /* Calculate the top of stack address. This depends on whether the
00464     stack grows from high memory to low (as per the 80x86) or visa versa.
00465     portSTACK_GROWTH is used to make the result positive or negative as
00466     required by the port. */
00467 #if( portSTACK_GROWTH < 0 )
00468 {
00469     pxTopOfStack = pxNewTCB->pxStack + ( usStackDepth - ( unsigned short ) 1 );
00470     pxTopOfStack = ( portSTACK_TYPE * ) ( ( ( portPOINTER_SIZE_TYPE ) pxTopOfStack ) & ( (
00471         portPOINTER_SIZE_TYPE ) ~portBYTE_ALIGNMENT_MASK ) );
00472
00473     /* Check the alignment of the calculated top of stack is correct. */
00474     configASSERT( ( ( ( unsigned long ) pxTopOfStack & ( unsigned long )
00475         portBYTE_ALIGNMENT_MASK ) == 0UL ) );
00476
00477 #else
00478 {
00479     pxTopOfStack = pxNewTCB->pxStack;
00480
00481     /* Check the alignment of the stack buffer is correct. */
00482     configASSERT( ( ( ( unsigned long ) pxNewTCB->pxStack & ( unsigned long )
00483         portBYTE_ALIGNMENT_MASK ) == 0UL ) );
00484
00485     /* If we want to use stack checking on architectures that use
00486     a positive stack growth direction then we also need to store the
00487     other extreme of the stack space. */
00488     pxNewTCB->pxEndOfStack = pxNewTCB->pxStack + ( usStackDepth - 1 );
00489 }
00490#endif
00491
00492     /* Setup the newly allocated TCB with the initial state of the task. */
00493     prvInitialiseTCBVariables( pxNewTCB, pcName, uxPriority, xRegions, usStackDepth );
00494
00495     /* Initialize the TCB stack to look as if the task was already running,
00496     but had been interrupted by the scheduler. The return address is set
00497     to the start of the task function. Once the stack has been initialised
00498     the top of stack variable is updated. */
00499 #if( portUSING_MPU_WRAPPERS == 1 )
00500 {
00501     pxNewTCB->pxTopOfStack = pxPortInitialiseStack( pxTopOfStack, pxTaskCode, pvParameters,
00502         xRunPrivileged );
00503
00504 #else
00505 {
00506     pxNewTCB->pxTopOfStack = pxPortInitialiseStack( pxTopOfStack, pxTaskCode, pvParameters );
00507
00508     /* Check the alignment of the initialised stack. */
00509     configASSERT( ( ( ( unsigned long ) pxNewTCB->pxTopOfStack & ( unsigned long )
00510         portBYTE_ALIGNMENT_MASK ) == 0UL ) );
00511
00512     if( ( void * ) pxCreatedTask != NULL )
00513     {
00514         /* Pass the TCB out - in an anonymous way. The calling function/
00515         task can use this as a handle to delete the task later if
00516         required.*/
00517         *pxCreatedTask = ( xTaskHandle ) pxNewTCB;
00518     }
00519
00520     /* We are going to manipulate the task queues to add this task to a
00521     ready list, so must make sure no interrupts occur. */
00522     taskENTER_CRITICAL();
00523     {
00524         uxCurrentNumberOfTasks++;
00525         if( pxCurrentTCB == NULL )
00526         {
00527             /* There are no other tasks, or all the other tasks are in
00528             the suspended state - make this the current task. */
00529             pxCurrentTCB = pxNewTCB;
00530
00531             if( uxCurrentNumberOfTasks == ( unsigned portBASE_TYPE ) 1 )
00532             {
00533                 /* This is the first task to be created so do the preliminary
00534                 initialisation required. We will not recover if this call
00535                 fails, but we will report the failure. */
00536                 prvInitialiseTaskLists();
00537             }
00538         }
00539     }
00540 }
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
02109
02110
02111
02112
02113
02114
02115
02116
02117
02118
02119
02120
02121
02122
02123
02124
02125
02126
02127
02128
02129
02130
02131
02132
02133
02134
02135
02136
02137
02138
02139
02140
02141
02142
02143
02144
02145
02146
02147
02148
02149
02150
02151
02152
02153
02154
02155
02156
02157
02158
02159
02160
02161
02162
02163
02164
02165
02166
02167

```

```

00535         }
00536     else
00537     {
00538         /* If the scheduler is not already running, make this task the
00539         current task if it is the highest priority task to be created
00540         so far. */
00541         if( xSchedulerRunning == pdFALSE )
00542         {
00543             if( pxCurrentTCB->uxPriority <= uxPriority )
00544             {
00545                 pxCurrentTCB = pxNewTCB;
00546             }
00547         }
00548     }
00549
00550     /* Remember the top priority to make context switching faster. Use
00551     the priority in pxNewTCB as this has been capped to a valid value. */
00552     if( pxNewTCB->uxPriority > uxTopUsedPriority )
00553     {
00554         uxTopUsedPriority = pxNewTCB->uxPriority;
00555     }
00556
00557     #if ( configUSE_TRACE_FACILITY == 1 )
00558     {
00559         /* Add a counter into the TCB for tracing only. */
00560         pxNewTCB->uxTCBNumber = uxTaskNumber;
00561     }
00562 #endif
00563     uxTaskNumber++;
00564
00565     prvAddTaskToReadyQueue( pxNewTCB );
00566
00567     xReturn = pdPASS;
00568     traceTASK_CREATE( pxNewTCB );
00569 }
00570 taskEXIT_CRITICAL();
00571 }
00572 else
00573 {
00574     xReturn = errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY;
00575     traceTASK_CREATE_FAILED();
00576 }
00577
00578 if( xReturn == pdPASS )
00579 {
00580     if( xSchedulerRunning != pdFALSE )
00581     {
00582         /* If the created task is of a higher priority than the current task
00583         then it should run now. */
00584         if( pxCurrentTCB->uxPriority < uxPriority )
00585         {
00586             portYIELD_WITHIN_API();
00587         }
00588     }
00589 }
00590
00591 return xReturn;
00592 }
00593 /***** vTaskDelete *****/
00594
00595 #if ( INCLUDE_vTaskDelete == 1 )
00596
00597 void vTaskDelete( xTaskHandle pxTaskToDelete )
00598 {
00599     tskTCB *pxTCB;
00600
00601     taskENTER_CRITICAL();
00602     {
00603         /* Ensure a yield is performed if the current task is being
00604         deleted. */
00605         if( pxTaskToDelete == pxCurrentTCB )
00606         {
00607             pxTaskToDelete = NULL;
00608         }
00609
00610         /* If null is passed in here then we are deleting ourselves. */
00611         pxTCB = prvGetTCBFromHandle( pxTaskToDelete );
00612
00613         /* Remove task from the ready list and place in the termination list.
00614         This will stop the task from being scheduled. The idle task will check
00615         the termination list and free up any memory allocated by the
00616         scheduler for the TCB and stack. */
00617         vListRemove( &( pxTCB->xGenericListItem ) );
00618
00619         /* Is the task waiting on an event also? */
00620         if( pxTCB->xEventListItem.pvContainer != NULL )
00621         {

```

```

00622         vListRemove( &( pxTCB->xEventListItem ) );
00623     }
00624
00625     vListInsertEnd( ( xList * ) &xTasksWaitingTermination, &( pxTCB->xGenericListItem ) );
00626
00627     /* Increment the uxTasksDeleted variable so the idle task knows
00628     there is a task that has been deleted and that it should therefore
00629     check the xTasksWaitingTermination list. */
00630     ++uxTasksDeleted;
00631
00632     /* Increment the uxTaskNumberVariable also so kernel aware debuggers
00633     can detect that the task lists need re-generating. */
00634     uxTaskNumber++;
00635
00636     traceTASK_DELETE( pxTCB );
00637 }
00638 taskEXIT_CRITICAL();
00639
00640     /* Force a reschedule if we have just deleted the current task. */
00641 if( xSchedulerRunning != pdFALSE )
00642 {
00643     if( ( void * ) pxTaskToDelete == NULL )
00644     {
00645         portYIELD_WITHIN_API();
00646     }
00647 }
00648 }
00649
00650 #endif
00651
00652
00653
00654
00655
00656
00657 /*-----*
00658 * TASK CONTROL API documented in task.h
00659 *-----*/
00660
00661 #if ( INCLUDE_vTaskDelayUntil == 1 )
00662
00663     void vTaskDelayUntil( portTickType * const pxPreviousWakeTime, portTickType xTimeIncrement )
00664     {
00665         portTickType xTimeToWake;
00666         portBASE_TYPE xAlreadyYielded, xShouldDelay = pdFALSE;
00667
00668         configASSERT( pxPreviousWakeTime );
00669         configASSERT( ( xTimeIncrement > 0U ) );
00670
00671         vTaskSuspendAll();
00672     {
00673         /* Generate the tick time at which the task wants to wake. */
00674         xTimeToWake = *pxPreviousWakeTime + xTimeIncrement;
00675
00676         if( xTickCount < *pxPreviousWakeTime )
00677     {
00678             /* The tick count has overflowed since this function was
00679             lasted called. In this case the only time we should ever
00680             actually delay is if the wake time has also overflowed,
00681             and the wake time is greater than the tick time. When this
00682             is the case it is as if neither time had overflowed. */
00683             if( ( xTimeToWake < *pxPreviousWakeTime ) && ( xTimeToWake > xTickCount ) )
00684             {
00685                 xShouldDelay = pdTRUE;
00686             }
00687         }
00688         else
00689     {
00690             /* The tick time has not overflowed. In this case we will
00691             delay if either the wake time has overflowed, and/or the
00692             tick time is less than the wake time. */
00693             if( ( xTimeToWake < *pxPreviousWakeTime ) || ( xTimeToWake > xTickCount ) )
00694             {
00695                 xShouldDelay = pdTRUE;
00696             }
00697         }
00698
00699         /* Update the wake time ready for the next call. */
00700         *pxPreviousWakeTime = xTimeToWake;
00701
00702         if( xShouldDelay != pdFALSE )
00703     {
00704             traceTASK_DELAY_UNTIL();
00705
00706             /* We must remove ourselves from the ready list before adding
00707             ourselves to the blocked list as the same list item is used for
00708             both lists. */
00709

```

```

00709         vListRemove( ( xListItem * ) & ( pxCurrentTCB->xGenericListItem ) );
00710         prvAddCurrentTaskToDelayedList( xTimeToWake );
00711     }
00712 }
00713 xAlreadyYielded = xTaskResumeAll();
00714
00715 /* Force a reschedule if xTaskResumeAll has not already done so, we may
00716 have put ourselves to sleep. */
00717 if( xAlreadyYielded == pdFALSE )
00718 {
00719     portYIELD_WITHIN_API();
00720 }
00721 }
00722
00723 #endif
00724 /*-----*/
00725
00726 #if ( INCLUDE_vTaskDelay == 1 )
00727
00728 void vTaskDelay( portTickType xTicksToDelay )
00729 {
00730     portTickType xTimeToWake;
00731     signed portBASE_TYPE xAlreadyYielded = pdFALSE;
00732
00733     /* A delay time of zero just forces a reschedule. */
00734     if( xTicksToDelay > ( portTickType ) 0U )
00735     {
00736         vTaskSuspendAll();
00737     {
00738         traceTASK_DELAY();
00739
00740             /* A task that is removed from the event list while the
00741 scheduler is suspended will not get placed in the ready
00742 list or removed from the blocked list until the scheduler
00743 is resumed.
00744
00745             This task cannot be in an event list as it is the currently
00746 executing task. */
00747
00748             /* Calculate the time to wake - this may overflow but this is
00749 not a problem. */
00750             xTimeToWake = xTickCount + xTicksToDelay;
00751
00752             /* We must remove ourselves from the ready list before adding
00753 ourselves to the blocked list as the same list item is used for
00754 both lists. */
00755             vListRemove( ( xListItem * ) & ( pxCurrentTCB->xGenericListItem ) );
00756             prvAddCurrentTaskToDelayedList( xTimeToWake );
00757     }
00758     xAlreadyYielded = xTaskResumeAll();
00759 }
00760
00761     /* Force a reschedule if xTaskResumeAll has not already done so, we may
00762 have put ourselves to sleep. */
00763     if( xAlreadyYielded == pdFALSE )
00764     {
00765         portYIELD_WITHIN_API();
00766     }
00767 }
00768
00769 #endif
00770 /*-----*/
00771
00772 #if ( INCLUDE_uxTaskPriorityGet == 1 )
00773
00774     unsigned portBASE_TYPE uxTaskPriorityGet( xTaskHandle pxTask )
00775     {
00776         tskTCB *pxTCB;
00777         unsigned portBASE_TYPE uxReturn;
00778
00779         taskENTER_CRITICAL();
00780     {
00781         /* If null is passed in here then we are changing the
00782 priority of the calling function. */
00783         pxTCB = prvGetTCBFromHandle( pxTask );
00784         uxReturn = pxTCB->uxPriority;
00785     }
00786     taskEXIT_CRITICAL();
00787
00788     return uxReturn;
00789 }
00790
00791 #endif
00792 /*-----*/
00793
00794 #if ( INCLUDE_vTaskPrioritySet == 1 )
00795

```

```

00796     void vTaskPrioritySet( xTaskHandle pxTask, unsigned portBASE_TYPE uxNewPriority )
00797     {
00798         tskTCB *pxTCB;
00799         unsigned portBASE_TYPE uxCurrentPriority;
00800         portBASE_TYPE xYieldRequired = pdFALSE;
00801
00802         configASSERT( ( uxNewPriority < configMAX_PRIORITIES ) );
00803
00804         /* Ensure the new priority is valid. */
00805         if( uxNewPriority >= configMAX_PRIORITIES )
00806         {
00807             uxNewPriority = configMAX_PRIORITIES - ( unsigned portBASE_TYPE ) 1U;
00808         }
00809
00810         taskENTER_CRITICAL();
00811     {
00812         if( pxTask == pxCurrentTCB )
00813         {
00814             pxTask = NULL;
00815         }
00816
00817         /* If null is passed in here then we are changing the
00818         priority of the calling function. */
00819         pxTCB = prvGetTCBFromHandle( pxTask );
00820
00821         traceTASK_PRIORITY_SET( pxTask, uxNewPriority );
00822
00823         #if ( configUSE_MUTEXES == 1 )
00824         {
00825             uxCurrentPriority = pxTCB->uxBasePriority;
00826         }
00827         #else
00828         {
00829             uxCurrentPriority = pxTCB->uxPriority;
00830         }
00831         #endif
00832
00833         if( uxCurrentPriority != uxNewPriority )
00834         {
00835             /* The priority change may have readied a task of higher
00836             priority than the calling task. */
00837             if( uxNewPriority > uxCurrentPriority )
00838             {
00839                 if( pxTask != NULL )
00840                 {
00841                     /* The priority of another task is being raised. If we
00842                     were raising the priority of the currently running task
00843                     there would be no need to switch as it must have already
00844                     been the highest priority task. */
00845                     xYieldRequired = pdTRUE;
00846                 }
00847             }
00848             else if( pxTask == NULL )
00849             {
00850                 /* Setting our own priority down means there may now be another
00851                 task of higher priority that is ready to execute. */
00852                 xYieldRequired = pdTRUE;
00853             }
00854
00855
00856         #if ( configUSE_MUTEXES == 1 )
00857         {
00858             /* Only change the priority being used if the task is not
00859             currently using an inherited priority. */
00860             if( pxTCB->uxBasePriority == pxTCB->uxPriority )
00861             {
00862                 pxTCB->uxPriority = uxNewPriority;
00863             }
00864
00865             /* The base priority gets set whatever. */
00866             pxTCB->uxBasePriority = uxNewPriority;
00867         }
00868         #else
00869         {
00870             pxTCB->uxPriority = uxNewPriority;
00871         }
00872         #endif
00873
00874         listSET_LIST_ITEM_VALUE( &( pxTCB->xEventListItem ), ( configMAX_PRIORITIES - (
00875             portTickType ) uxNewPriority ) );
00876
00877         /* If the task is in the blocked or suspended list we need do
00878         nothing more than change it's priority variable. However, if
00879         the task is in a ready list it needs to be removed and placed
00880         in the queue appropriate to its new priority. */
00881         if( listIS_CONTAINED_WITHIN( &( pxReadyTasksLists[ uxCurrentPriority ] ), &(
```

```

00882     pxTCB->xGenericListItem ) ) )
00883     /* The task is currently in its ready list - remove before adding
00884     it to its new ready list. As we are in a critical section we
00885     can do this even if the scheduler is suspended. */
00886     vListRemove( &( pxTCB->xGenericListItem ) );
00887     prvAddTaskToReadyQueue( pxTCB );
00888 }
00889
00890     if( xYieldRequired == pdTRUE )
00891 {
00892     portYIELD_WITHIN_API();
00893 }
00894 }
00895 }
00896 taskEXIT_CRITICAL();
00897 }
00898
00899 #endif
00900 /*-----*/
00901
00902 #if ( INCLUDE_vTaskSuspend == 1 )
00903
00904 void vTaskSuspend( xTaskHandle pxTaskToSuspend )
00905 {
00906     tskTCB *pxTCB;
00907
00908     taskENTER_CRITICAL();
00909 {
00910     /* Ensure a yield is performed if the current task is being
00911     suspended. */
00912     if( pxTaskToSuspend == pxCurrentTCB )
00913     {
00914         pxTaskToSuspend = NULL;
00915     }
00916
00917     /* If null is passed in here then we are suspending ourselves. */
00918     pxTCB = prvGetTCBFromHandle( pxTaskToSuspend );
00919
00920     traceTASK_SUSPEND( pxTCB );
00921
00922     /* Remove task from the ready/delayed list and place in the suspended list. */
00923     vListRemove( &( pxTCB->xGenericListItem ) );
00924
00925     /* Is the task waiting on an event also? */
00926     if( pxTCB->xEventListItem.pvContainer != NULL )
00927     {
00928         vListRemove( &( pxTCB->xEventListItem ) );
00929     }
00930
00931     vListInsertEnd( ( xList * ) &xSuspendedTaskList, &( pxTCB->xGenericListItem ) );
00932 }
00933 taskEXIT_CRITICAL();
00934
00935 if( ( void * ) pxTaskToSuspend == NULL )
00936 {
00937     if( xSchedulerRunning != pdFALSE )
00938     {
00939         /* We have just suspended the current task. */
00940         portYIELD_WITHIN_API();
00941     }
00942     else
00943     {
00944         /* The scheduler is not running, but the task that was pointed
00945         to by pxCurrentTCB has just been suspended and pxCurrentTCB
00946         must be adjusted to point to a different task. */
00947         if( listCURRENT_LIST_LENGTH( &xSuspendedTaskList ) == uxCurrentNumberOfTasks )
00948         {
00949             /* No other tasks are ready, so set pxCurrentTCB back to
00950             NULL so when the next task is created pxCurrentTCB will
00951             be set to point to it no matter what its relative priority
00952             is. */
00953             pxCurrentTCB = NULL;
00954         }
00955     else
00956     {
00957         vTaskSwitchContext();
00958     }
00959 }
00960 }
00961 }
00962
00963 #endif
00964 /*-----*/
00965
00966 #if ( INCLUDE_vTaskSuspend == 1 )
00967

```

```

00968     signed portBASE_TYPE xTaskIsTaskSuspended( xTaskHandle xTask )
00969     {
00970         portBASE_TYPE xReturn = pdFALSE;
00971         const tskTCB * const pxtCB = ( tskTCB * ) xTask;
00972
00973         /* It does not make sense to check if the calling task is suspended. */
00974         configASSERT( xTask );
00975
00976         /* Is the task we are attempting to resume actually in the
00977            suspended list? */
00978         if( listIS_CONTAINED_WITHIN( &xSuspendedTaskList, &( pxTCB->xGenericListItem ) ) != pdFALSE )
00979         {
00980             /* Has the task already been resumed from within an ISR? */
00981             if( listIS_CONTAINED_WITHIN( &xPendingReadyList, &( pxTCB->xEventListItem ) ) != pdTRUE )
00982             {
00983                 /* Is it in the suspended list because it is in the
00984                    Suspended state? It is possible to be in the suspended
00985                    list because it is blocked on a task with no timeout
00986                    specified. */
00987                 if( listIS_CONTAINED_WITHIN( NULL, &( pxTCB->xEventListItem ) ) == pdTRUE )
00988                 {
00989                     xReturn = pdTRUE;
00990                 }
00991             }
00992         }
00993
00994         return xReturn;
00995     }
00996
00997 #endif
00998 /*-----*/
00999
01000 #if ( INCLUDE_vTaskSuspend == 1 )
01001
01002     void vTaskResume( xTaskHandle pxTaskToResume )
01003     {
01004         tskTCB *pxTCB;
01005
01006         /* It does not make sense to resume the calling task. */
01007         configASSERT( pxTaskToResume );
01008
01009         /* Remove the task from whichever list it is currently in, and place
01010            it in the ready list. */
01011         pxTCB = ( tskTCB * ) pxTaskToResume;
01012
01013         /* The parameter cannot be NULL as it is impossible to resume the
01014            currently executing task. */
01015         if( ( pxTCB != NULL ) && ( pxTCB != pxCurrentTCB ) )
01016         {
01017             taskENTER_CRITICAL();
01018             {
01019                 if( xTaskIsTaskSuspended( pxTCB ) == pdTRUE )
01020                 {
01021                     traceTASK_RESUME( pxTCB );
01022
01023                     /* As we are in a critical section we can access the ready
01024                        lists even if the scheduler is suspended. */
01025                     vListRemove( &( pxTCB->xGenericListItem ) );
01026                     prvAddTaskToReadyQueue( pxTCB );
01027
01028                     /* We may have just resumed a higher priority task. */
01029                     if( pxTCB->uxPriority >= pxCurrentTCB->uxPriority )
01030                     {
01031                         /* This yield may not cause the task just resumed to run, but
01032                            will leave the lists in the correct state for the next yield. */
01033                         portYIELD_WITHIN_API();
01034                     }
01035                 }
01036             }
01037             taskEXIT_CRITICAL();
01038         }
01039     }
01040
01041 #endif
01042
01043 /*-----*/
01044
01045 #if ( ( INCLUDE_xTaskResumeFromISR == 1 ) && ( INCLUDE_vTaskSuspend == 1 ) )
01046
01047     portBASE_TYPE xTaskResumeFromISR( xTaskHandle pxTaskToResume )
01048     {
01049         portBASE_TYPE xYieldRequired = pdFALSE;
01050         tskTCB *pxTCB;
01051
01052         configASSERT( pxTaskToResume );
01053
01054         pxTCB = ( tskTCB * ) pxTaskToResume;

```

```

01055     if( xTaskIsTaskSuspended( pxTCB ) == pdTRUE )
01056     {
01057         traceTASK_RESUME_FROM_ISR( pxTCB );
01058
01059         if( uxSchedulerSuspended == ( unsigned portBASE_TYPE ) pdFALSE )
01060         {
01061             xYieldRequired = ( pxTCB->uxPriority >= pxCurrentTCB->uxPriority );
01062             vListRemove( &( pxTCB->xGenericListItem ) );
01063             prvAddTaskToReadyQueue( pxTCB );
01064         }
01065     }
01066     else
01067     {
01068         /* We cannot access the delayed or ready lists, so will hold this
01069         task pending until the scheduler is resumed, at which point a
01070         yield will be performed if necessary. */
01071         vListInsertEnd( ( xList * ) &( xPendingReadyList ), &( pxTCB->xEventListItem ) );
01072     }
01073 }
01074
01075     return xYieldRequired;
01076 }
01077
01078 #endif
01079
01080
01081
01082
01083 /*****
01084 * PUBLIC SCHEDULER CONTROL documented in task.h
01085 *****/
01086
01087
01088 void vTaskStartScheduler( void )
01089 {
01090     portBASE_TYPE xReturn;
01091
01092     /* Add the idle task at the lowest priority. */
01093     #if ( INCLUDE_xTaskGetIdleTaskHandle == 1 )
01094     {
01095         /* Create the idle task, storing its handle in xIdleTaskHandle so it can
01096         be returned by the xTaskGetIdleTaskHandle() function. */
01097         xReturn = xTaskCreate( prvIdleTask, ( signed char * ) "IDLE", tskIDLE_STACK_SIZE, ( void * )
01098             NULL, ( tskIDLE_PRIORITY | portPRIVILEGE_BIT ), &xIdleTaskHandle );
01099     }
01100     #else
01101     {
01102         /* Create the idle task without storing its handle. */
01103         xReturn = xTaskCreate( prvIdleTask, ( signed char * ) "IDLE", tskIDLE_STACK_SIZE, ( void * )
01104             NULL, ( tskIDLE_PRIORITY | portPRIVILEGE_BIT ), NULL );
01105     }
01106     #endif
01107
01108     #if ( configUSE_TIMERS == 1 )
01109     {
01110         if( xReturn == pdPASS )
01111         {
01112             xReturn = xTimerCreateTimerTask();
01113         }
01114     }
01115
01116     if( xReturn == pdPASS )
01117     {
01118         /* Interrupts are turned off here, to ensure a tick does not occur
01119         before or during the call to xPortStartScheduler(). The stacks of
01120         the created tasks contain a status word with interrupts switched on
01121         so interrupts will automatically get re-enabled when the first task
01122         starts to run.
01123
01124         STEPPING THROUGH HERE USING A DEBUGGER CAN CAUSE BIG PROBLEMS IF THE
01125         DEBUGGER ALLOWS INTERRUPTS TO BE PROCESSED. */
01126         portDISABLE_INTERRUPTS();
01127
01128         xSchedulerRunning = pdTRUE;
01129         xTickCount = ( portTickType ) 0U;
01130
01131         /* If configGENERATE_RUN_TIME_STATS is defined then the following
01132         macro must be defined to configure the timer/counter used to generate
01133         the run time counter time base. */
01134         portCONFIGURE_TIMER_FOR_RUN_TIME_STATS();
01135
01136         /* Setting up the timer tick is hardware specific and thus in the
01137         portable interface. */
01138         if( xPortStartScheduler() != pdFALSE )
01139         {
01140             /* Should not reach here as if the scheduler is running the

```

```

01140         function will not return. */
01141     }
01142     else
01143     {
01144         /* Should only reach here if a task calls xTaskEndScheduler(). */
01145     }
01146 }
01147
01148 /* This line will only be reached if the kernel could not be started. */
01149 configASSERT( xReturn );
01150 }
01151 /*-----*/
01152
01153 void vTaskEndScheduler( void )
01154 {
01155     /* Stop the scheduler interrupts and call the portable scheduler end
01156     routine so the original ISRs can be restored if necessary. The port
01157     layer must ensure interrupts enable bit is left in the correct state. */
01158     portDISABLE_INTERRUPTS();
01159     xSchedulerRunning = pdFALSE;
01160     vPortEndScheduler();
01161 }
01162 /*-----*/
01163
01164 void vTaskSuspendAll( void )
01165 {
01166     /* A critical section is not required as the variable is of type
01167     portBASE_TYPE. */
01168     ++uxSchedulerSuspended;
01169 }
01170 /*-----*/
01171
01172 signed portBASE_TYPE xTaskResumeAll( void )
01173 {
01174     register tskTCB *pxTCB;
01175     signed portBASE_TYPE xAlreadyYielded = pdFALSE;
01176
01177     /* If uxSchedulerSuspended is zero then this function does not match a
01178     previous call to vTaskSuspendAll(). */
01179     configASSERT( uxSchedulerSuspended );
01180
01181     /* It is possible that an ISR caused a task to be removed from an event
01182     list while the scheduler was suspended. If this was the case then the
01183     removed task will have been added to the xPendingReadyList. Once the
01184     scheduler has been resumed it is safe to move all the pending ready
01185     tasks from this list into their appropriate ready list. */
01186     taskENTER_CRITICAL();
01187     {
01188         --uxSchedulerSuspended;
01189
01190         if( uxSchedulerSuspended == ( unsigned portBASE_TYPE ) pdFALSE )
01191         {
01192             if( uxCurrentNumberOfTasks > ( unsigned portBASE_TYPE ) OU )
01193             {
01194                 portBASE_TYPE xYieldRequired = pdFALSE;
01195
01196                 /* Move any readied tasks from the pending list into the
01197                 appropriate ready list. */
01198                 while( listLIST_IS_EMPTY( ( xList * ) &xPendingReadyList ) == pdFALSE )
01199                 {
01200                     pxTCB = ( tskTCB * ) listGET_OWNER_OF_HEAD_ENTRY( ( ( xList * )
01201                     &xPendingReadyList ) );
01202                     vListRemove( &( pxTCB->xEventListItem ) );
01203                     vListRemove( &( pxTCB->xGenericListItem ) );
01204                     prvAddTaskToReadyQueue( pxTCB );
01205
01206                     /* If we have moved a task that has a priority higher than
01207                     the current task then we should yield. */
01208                     if( pxTCB->uxPriority >= pxCurrentTCB->uxPriority )
01209                     {
01210                         xYieldRequired = pdTRUE;
01211                     }
01212
01213                     /* If any ticks occurred while the scheduler was suspended then
01214                     they should be processed now. This ensures the tick count does not
01215                     slip, and that any delayed tasks are resumed at the correct time. */
01216                     if( uxMissedTicks > ( unsigned portBASE_TYPE ) OU )
01217                     {
01218                         while( uxMissedTicks > ( unsigned portBASE_TYPE ) OU )
01219                         {
01220                             vTaskIncrementTick();
01221                             --uxMissedTicks;
01222                         }
01223
01224                     /* As we have processed some ticks it is appropriate to yield
01225                     to ensure the highest priority task that is ready to run is

```

```

01226             the task actually running. */
01227             #if configUSE_PREEMPTION == 1
01228             {
01229                 xYieldRequired = pdTRUE;
01230             }
01231             #endif
01232         }
01233     }
01234     if( ( xYieldRequired == pdTRUE ) || ( xMissedYield == pdTRUE ) )
01235     {
01236         xAlreadyYielded = pdTRUE;
01237         xMissedYield = pdFALSE;
01238         portYIELD_WITHIN_API();
01239     }
01240 }
01241 }
01242 }
01243 taskEXIT_CRITICAL();
01244
01245     return xAlreadyYielded;
01246 }
01247
01248
01249 /*-----*
01250 * PUBLIC TASK UTILITIES documented in task.h
01251 *-----*/
01252
01253 /*-----*
01254 *-----*
01255 *-----*/
01256
01257
01258
01259 portTickType xTaskGetTickCount( void )
01260 {
01261     portTickType xTicks;
01262
01263     /* Critical section required if running on a 16 bit processor. */
01264     taskENTER_CRITICAL();
01265     {
01266         xTicks = xTickCount;
01267     }
01268     taskEXIT_CRITICAL();
01269
01270     return xTicks;
01271 }
01272 /*-----*/
01273
01274 portTickType xTaskGetTickCountFromISR( void )
01275 {
01276     portTickType xReturn;
01277     unsigned portBASE_TYPE uxSavedInterruptStatus;
01278
01279     uxSavedInterruptStatus = portSET_INTERRUPT_MASK_FROM_ISR();
01280     xReturn = xTickCount;
01281     portCLEAR_INTERRUPT_MASK_FROM_ISR( uxSavedInterruptStatus );
01282
01283     return xReturn;
01284 }
01285 /*-----*/
01286
01287 unsigned portBASE_TYPE uxTaskGetNumberOfTasks( void )
01288 {
01289     /* A critical section is not required because the variables are of type
01290     portBASE_TYPE. */
01291     return uxCurrentNumberOfTasks;
01292 }
01293 /*-----*/
01294
01295 #if ( INCLUDE_pcTaskGetTaskName == 1 )
01296
01297     signed char *pcTaskGetTaskName( xTaskHandle xTaskToQuery )
01298     {
01299         tskTCB *pxTCB;
01300
01301         /* If null is passed in here then the name of the calling task is being queried. */
01302         pxTCB = prvGetTCBFromHandle( xTaskToQuery );
01303         configASSERT( pxTCB );
01304         return &( pxTCB->pcTaskName[ 0 ] );
01305     }
01306
01307 #endif
01308 /*-----*/
01309
01310 #if ( configUSE_TRACE_FACILITY == 1 )
01311
01312     void vTaskList( signed char *pcWriteBuffer )

```

```

01313     {
01314     unsigned portBASE_TYPE uxQueue;
01315
01316     /* This is a VERY costly function that should be used for debug only.
01317     It leaves interrupts disabled for a LONG time. */
01318
01319     vTaskSuspendAll();
01320     {
01321         /* Run through all the lists that could potentially contain a TCB and
01322         report the task name, state and stack high water mark. */
01323
01324         *pcWriteBuffer = ( signed char ) 0x00;
01325         strcat( ( char * ) pcWriteBuffer, ( const char * ) "\r\n" );
01326
01327         uxQueue = uxTopUsedPriority + ( unsigned portBASE_TYPE ) 1U;
01328
01329         do
01330         {
01331             uxQueue--;
01332
01333             if( listLIST_IS_EMPTY( &( pxReadyTasksLists[ uxQueue ] ) ) == pdFALSE )
01334             {
01335                 prvListTaskWithinSingleList( pcWriteBuffer, ( xList * ) &( pxReadyTasksLists[
01336                     uxQueue ] ), tskREADY_CHAR );
01337             }
01338             }while( uxQueue > ( unsigned short ) tskIDLE_PRIORITY );
01339
01340             if( listLIST_IS_EMPTY( pxDelayedTaskList ) == pdFALSE )
01341             {
01342                 prvListTaskWithinSingleList( pcWriteBuffer, ( xList * ) pxDelayedTaskList,
01343                     tskBLOCKED_CHAR );
01344             }
01345
01346             if( listLIST_IS_EMPTY( pxOverflowDelayedTaskList ) == pdFALSE )
01347             {
01348                 prvListTaskWithinSingleList( pcWriteBuffer, ( xList * ) pxOverflowDelayedTaskList,
01349                     tskBLOCKED_CHAR );
01350             }
01351
01352             #if( INCLUDE_vTaskDelete == 1 )
01353             {
01354                 if( listLIST_IS_EMPTY( &xTasksWaitingTermination ) == pdFALSE )
01355                 {
01356                     prvListTaskWithinSingleList( pcWriteBuffer, &xTasksWaitingTermination,
01357                         tskDELETED_CHAR );
01358                 }
01359             }
01360             #endif
01361
01362             #if( INCLUDE_vTaskSuspend == 1 )
01363             {
01364                 if( listLIST_IS_EMPTY( &xSuspendedTaskList ) == pdFALSE )
01365                 {
01366                     prvListTaskWithinSingleList( pcWriteBuffer, &xSuspendedTaskList, tskSUSPENDED_CHAR
01367                 );
01368             }
01369         }
01370     #endif
01371     /*-----*/
01372
01373 #if ( configGENERATE_RUN_TIME_STATS == 1 )
01374
01375     void vTaskGetRunTimeStats( signed char *pcWriteBuffer )
01376     {
01377         unsigned portBASE_TYPE uxQueue;
01378         unsigned long ulTotalRunTime;
01379
01380         /* This is a VERY costly function that should be used for debug only.
01381         It leaves interrupts disabled for a LONG time. */
01382
01383         vTaskSuspendAll();
01384         {
01385             #ifdef portALT_GET_RUN_TIME_COUNTER_VALUE
01386                 portALT_GET_RUN_TIME_COUNTER_VALUE( ulTotalRunTime );
01387             #else
01388                 ulTotalRunTime = portGET_RUN_TIME_COUNTER_VALUE();
01389             #endif
01390
01391             /* Divide ulTotalRunTime by 100 to make the percentage caluclations
01392             simpler in the prvGenerateRunTimeStatsForTasksInList() function. */
01393             ulTotalRunTime /= 100UL;
01394

```

```

01395     /* Run through all the lists that could potentially contain a TCB,
01396      generating a table of run timer percentages in the provided
01397      buffer. */
01398
01399     *pcWriteBuffer = ( signed char ) 0x00;
01400     strcat( ( char * ) pcWriteBuffer, ( const char * ) "\r\n" );
01401
01402     uxQueue = uxTopUsedPriority + ( unsigned portBASE_TYPE ) 1U;
01403
01404     do
01405     {
01406         uxQueue--;
01407
01408         if( listLIST_IS_EMPTY( &( pxReadyTasksLists[ uxQueue ] ) ) == pdFALSE )
01409         {
01410             prvGenerateRunTimeStatsForTasksInList( pcWriteBuffer, ( xList * ) &( pxReadyTasksLists[ uxQueue ] ), ulTotalRunTime );
01411             }
01412             }while( uxQueue > ( unsigned short ) tskIDLE_PRIORITY );
01413
01414         if( listLIST_IS_EMPTY( pxDelayedTaskList ) == pdFALSE )
01415         {
01416             prvGenerateRunTimeStatsForTasksInList( pcWriteBuffer, ( xList * ) pxDelayedTaskList,
01417             ulTotalRunTime );
01418             }
01419             if( listLIST_IS_EMPTY( pxOverflowDelayedTaskList ) == pdFALSE )
01420             {
01421                 prvGenerateRunTimeStatsForTasksInList( pcWriteBuffer, ( xList * ) pxOverflowDelayedTaskList,
01422                 ulTotalRunTime );
01423             }
01424
01425         #if ( INCLUDE_vTaskDelete == 1 )
01426         {
01427             if( listLIST_IS_EMPTY( &xTasksWaitingTermination ) == pdFALSE )
01428             {
01429                 prvGenerateRunTimeStatsForTasksInList( pcWriteBuffer, &xTasksWaitingTermination,
01430                 ulTotalRunTime );
01431             }
01432             #endif
01433
01434         #if ( INCLUDE_vTaskSuspend == 1 )
01435         {
01436             if( listLIST_IS_EMPTY( &xSuspendedTaskList ) == pdFALSE )
01437             {
01438                 prvGenerateRunTimeStatsForTasksInList( pcWriteBuffer, &xSuspendedTaskList,
01439                 ulTotalRunTime );
01440             }
01441             #endif
01442             xTaskResumeAll();
01443         }
01444
01445 #endif
01446 /*-----*/
01447
01448 #if ( configUSE_TRACE_FACILITY == 1 )
01449
01450     void vTaskStartTrace( signed char * pcBuffer, unsigned long ulBufferSize )
01451     {
01452         configASSERT( pcBuffer );
01453         configASSERT( ulBufferSize );
01454
01455         taskENTER_CRITICAL();
01456         {
01457             pcTraceBuffer = ( signed char * )pcBuffer;
01458             pcTraceBufferStart = pcBuffer;
01459             pcTraceBufferEnd = pcBuffer + ( ulBufferSize - tskSIZE_OF_EACH_TRACE_LINE );
01460             xTracing = pdTRUE;
01461         }
01462         taskEXIT_CRITICAL();
01463     }
01464
01465 #endif
01466 /*-----*/
01467
01468 #if ( configUSE_TRACE_FACILITY == 1 )
01469
01470     unsigned long ulTaskEndTrace( void )
01471     {
01472         unsigned long ulBufferLength;
01473
01474         taskENTER_CRITICAL();
01475         xTracing = pdFALSE;
01476         taskEXIT_CRITICAL();
01477

```

```

01477     ulBufferLength = ( unsigned long ) ( pcTraceBuffer - pcTraceBufferStart );
01478
01479     return ulBufferLength;
01480 }
01481 }
01482
01483 #endif
01484 /*-----*/
01485
01486 #if ( INCLUDE_xTaskGetIdleTaskHandle == 1 )
01487
01488     xTaskHandle xTaskGetIdleTaskHandle( void )
01489 {
01490     /* If xTaskGetIdleTaskHandle() is called before the scheduler has been
01491     started, then xIdleTaskHandle will be NULL. */
01492     configASSERT( ( xIdleTaskHandle != NULL ) );
01493     return xIdleTaskHandle;
01494 }
01495
01496 #endif
01497
01498 /*-----*/
01499 * SCHEDULER INTERNALS AVAILABLE FOR PORTING PURPOSES
01500 * documented in task.h
01501 *-----*/
01502
01503 void vTaskIncrementTick( void )
01504 {
01505     tskTCB * pxTCB;
01506
01507     /* Called by the portable layer each time a tick interrupt occurs.
01508     Increments the tick then checks to see if the new tick value will cause any
01509     tasks to be unblocked. */
01510     if( uxSchedulerSuspended == ( unsigned portBASE_TYPE ) pdFALSE )
01511     {
01512         ++xTickCount;
01513         if( xTickCount == ( portTickType ) OU )
01514         {
01515             xList *pxTemp;
01516
01517             /* Tick count has overflowed so we need to swap the delay lists.
01518             If there are any items in pxDelayedTaskList here then there is
01519             an error! */
01520             configASSERT( ( listLIST_IS_EMPTY( pxDelayedTaskList ) ) );
01521
01522             pxTemp = pxDelayedTaskList;
01523             pxDelayedTaskList = pxOverflowDelayedTaskList;
01524             pxOverflowDelayedTaskList = pxTemp;
01525             xNumOfOverflows++;
01526
01527             if( listLIST_IS_EMPTY( pxDelayedTaskList ) != pdFALSE )
01528             {
01529                 /* The new current delayed list is empty. Set
01530                 xNextTaskUnblockTime to the maximum possible value so it is
01531                 extremely unlikely that the
01532                 if( xTickCount >= xNextTaskUnblockTime ) test will pass until
01533                 there is an item in the delayed list. */
01534                 xNextTaskUnblockTime = portMAX_DELAY;
01535             }
01536             else
01537             {
01538                 /* The new current delayed list is not empty, get the value of
01539                 the item at the head of the delayed list. This is the time at
01540                 which the task at the head of the delayed list should be removed
01541                 from the Blocked state. */
01542                 pxTCB = ( tskTCB * ) listGET_OWNER_OF_HEAD_ENTRY( pxDelayedTaskList );
01543                 xNextTaskUnblockTime = listGET_LIST_ITEM_VALUE( &( pxTCB->xGenericListItem ) );
01544             }
01545         }
01546
01547         /* See if this tick has made a timeout expire. */
01548         prvCheckDelayedTasks();
01549     }
01550     else
01551     {
01552         ++uxMissedTicks;
01553
01554         /* The tick hook gets called at regular intervals, even if the
01555         scheduler is locked. */
01556         #if ( configUSE_TICK_HOOK == 1 )
01557         {
01558             vApplicationTickHook();
01559         }
01560         #endif
01561     }
01562
01563 #if ( configUSE_TICK_HOOK == 1 )

```

```

01564     {
01565         /* Guard against the tick hook being called when the missed tick
01566         count is being unwound (when the scheduler is being unlocked. */
01567         if( uxMissedTicks == ( unsigned portBASE_TYPE ) OU )
01568         {
01569             vApplicationTickHook();
01570         }
01571     }
01572 #endif
01573
01574     traceTASK_INCREMENT_TICK( xTickCount );
01575 }
01576 /*-----*/
01577
01578 #if ( configUSE_APPLICATION_TASK_TAG == 1 )
01579
01580     void vTaskSetApplicationTaskTag( xTaskHandle xTask, pdTASK_HOOK_CODE pxHookFunction )
01581     {
01582         tskTCB *xTCB;
01583
01584         /* If xTask is NULL then we are setting our own task hook. */
01585         if( xTask == NULL )
01586         {
01587             xTCB = ( tskTCB * ) pxCurrentTCB;
01588         }
01589         else
01590         {
01591             xTCB = ( tskTCB * ) xTask;
01592         }
01593
01594         /* Save the hook function in the TCB. A critical section is required as
01595         the value can be accessed from an interrupt. */
01596         taskENTER_CRITICAL();
01597         xTCB->pxTaskTag = pxHookFunction;
01598         taskEXIT_CRITICAL();
01599     }
01600
01601 #endif
01602 /*-----*/
01603
01604 #if ( configUSE_APPLICATION_TASK_TAG == 1 )
01605
01606     pdTASK_HOOK_CODE xTaskGetApplicationTaskTag( xTaskHandle xTask )
01607     {
01608         tskTCB *xTCB;
01609         pdTASK_HOOK_CODE xReturn;
01610
01611         /* If xTask is NULL then we are setting our own task hook. */
01612         if( xTask == NULL )
01613         {
01614             xTCB = ( tskTCB * ) pxCurrentTCB;
01615         }
01616         else
01617         {
01618             xTCB = ( tskTCB * ) xTask;
01619         }
01620
01621         /* Save the hook function in the TCB. A critical section is required as
01622         the value can be accessed from an interrupt. */
01623         taskENTER_CRITICAL();
01624         xReturn = xTCB->pxTaskTag;
01625         taskEXIT_CRITICAL();
01626
01627         return xReturn;
01628     }
01629
01630 #endif
01631 /*-----*/
01632
01633 #if ( configUSE_APPLICATION_TASK_TAG == 1 )
01634
01635     portBASE_TYPE xTaskCallApplicationTaskHook( xTaskHandle xTask, void *pvParameter )
01636     {
01637         tskTCB *xTCB;
01638         portBASE_TYPE xReturn;
01639
01640         /* If xTask is NULL then we are calling our own task hook. */
01641         if( xTask == NULL )
01642         {
01643             xTCB = ( tskTCB * ) pxCurrentTCB;
01644         }
01645         else
01646         {
01647             xTCB = ( tskTCB * ) xTask;
01648         }
01649
01650         if( xTCB->pxTaskTag != NULL )

```

```

01651         {
01652             xReturn = xTCB->pxTaskTag( pvParameter );
01653         }
01654     else
01655     {
01656         xReturn = pdFAIL;
01657     }
01658
01659     return xReturn;
01660 }
01661
01662 #endif
01663 /*-----*/
01664
01665 void vTaskSwitchContext( void )
01666 {
01667     if( uxSchedulerSuspended != ( unsigned portBASE_TYPE ) pdFALSE )
01668     {
01669         /* The scheduler is currently suspended - do not allow a context
01670         switch. */
01671         xMissedYield = pdTRUE;
01672     }
01673 else
01674 {
01675     traceTASK_SWITCHED_OUT();
01676
01677     #if ( configGENERATE_RUN_TIME_STATS == 1 )
01678     {
01679         unsigned long ulTempCounter;
01680
01681         #ifdef portALT_GET_RUN_TIME_COUNTER_VALUE
01682             portALT_GET_RUN_TIME_COUNTER_VALUE( ulTempCounter );
01683         #else
01684             ulTempCounter = portGET_RUN_TIME_COUNTER_VALUE();
01685         #endif
01686
01687         /* Add the amount of time the task has been running to the accumulated
01688         time so far. The time the task started running was stored in
01689         ulTaskSwitchedInTime. Note that there is no overflow protection here
01690         so count values are only valid until the timer overflows. Generally
01691         this will be about 1 hour assuming a 1uS timer increment. */
01692         pxCurrentTCB->ulRunTimeCounter += ( ulTempCounter - ulTaskSwitchedInTime );
01693         ulTaskSwitchedInTime = ulTempCounter;
01694     }
01695 #endif
01696
01697     taskFIRST_CHECK_FOR_STACK_OVERFLOW();
01698     taskSECOND_CHECK_FOR_STACK_OVERFLOW();
01699
01700     /* Find the highest priority queue that contains ready tasks. */
01701     while( listLIST_IS_EMPTY( &( pxReadyTasksLists[ uxTopReadyPriority ] ) ) )
01702     {
01703         configASSERT( uxTopReadyPriority );
01704         --uxTopReadyPriority;
01705     }
01706
01707     /* listGET_OWNER_OF_NEXT_ENTRY walks through the list, so the tasks of the
01708     same priority get an equal share of the processor time. */
01709     listGET_OWNER_OF_NEXT_ENTRY( pxCurrentTCB, &( pxReadyTasksLists[ uxTopReadyPriority ] ) );
01710
01711     traceTASK_SWITCHED_IN();
01712     vWriteTraceToBuffer();
01713 }
01714 }
01715 /*-----*/
01716
01717 void vTaskPlaceOnEventList( const xList * const pxEventList, portTickType xTicksToWait )
01718 {
01719     portTickType xTimeToWake;
01720
01721     configASSERT( pxEventList );
01722
01723     /* THIS FUNCTION MUST BE CALLED WITH INTERRUPTS DISABLED OR THE
01724     SCHEDULER SUSPENDED. */
01725
01726     /* Place the event list item of the TCB in the appropriate event list.
01727     This is placed in the list in priority order so the highest priority task
01728     is the first to be woken by the event. */
01729     vListInsert( ( xList * ) pxEventList, ( xListItem * ) &( pxCurrentTCB->xEventListItem ) );
01730
01731     /* We must remove ourselves from the ready list before adding ourselves
01732     to the blocked list as the same list item is used for both lists. We have
01733     exclusive access to the ready lists as the scheduler is locked. */
01734     vListRemove( ( xListItem * ) &( pxCurrentTCB->xGenericListItem ) );
01735
01736     #if ( INCLUDE_vTaskSuspend == 1 )
01737

```

```

01738     {
01739         if( xTicksToWait == portMAX_DELAY )
01740     {
01741         /* Add ourselves to the suspended task list instead of a delayed task
01742         list to ensure we are not woken by a timing event. We will block
01743         indefinitely. */
01744         vListInsertEnd( ( xList * ) &xSuspendedTaskList, ( xListItem * ) &(
01745             pxCurrentTCB->xGenericListItem ) );
01746     }
01747     else
01748     {
01749         /* Calculate the time at which the task should be woken if the event does
01750         not occur. This may overflow but this doesn't matter. */
01751         xTimeToWake = xTickCount + xTicksToWait;
01752         prvAddCurrentTaskToDelayedList( xTimeToWake );
01753     }
01754 }
01755 {
01756     /* Calculate the time at which the task should be woken if the event does
01757     not occur. This may overflow but this doesn't matter. */
01758     xTimeToWake = xTickCount + xTicksToWait;
01759     prvAddCurrentTaskToDelayedList( xTimeToWake );
01760 }
01761 #endif
01762 }
01763 /*-----*/
01764
01765 #if configUSE_TIMERS == 1
01766
01767     void vTaskPlaceOnEventListRestricted( const xList * const pxEventList, portTickType xTicksToWait )
01768     {
01769         portTickType xTimeToWake;
01770
01771         configASSERT( pxEventList );
01772
01773         /* This function should not be called by application code hence the
01774         'Restricted' in its name. It is not part of the public API. It is
01775         designed for use by kernel code, and has special calling requirements -
01776         it should be called from a critical section. */
01777
01778
01779         /* Place the event list item of the TCB in the appropriate event list.
01780         In this case it is assume that this is the only task that is going to
01781         be waiting on this event list, so the faster vListInsertEnd() function
01782         can be used in place of vListInsert. */
01783         vListInsertEnd( ( xList * ) pxEventList, ( xListItem * ) &( pxCurrentTCB->xEventListItem ) );
01784
01785         /* We must remove this task from the ready list before adding it to the
01786         blocked list as the same list item is used for both lists. This
01787         function is called form a critical section. */
01788         vListRemove( ( xListItem * ) &( pxCurrentTCB->xGenericListItem ) );
01789
01790         /* Calculate the time at which the task should be woken if the event does
01791         not occur. This may overflow but this doesn't matter. */
01792         xTimeToWake = xTickCount + xTicksToWait;
01793         prvAddCurrentTaskToDelayedList( xTimeToWake );
01794     }
01795
01796 #endif /* configUSE_TIMERS */
01797 /*-----*/
01798
01799 signed portBASE_TYPE xTaskRemoveFromEventList( const xList * const pxEventList )
01800 {
01801     tskTCB *pxUnblockedTCB;
01802     portBASE_TYPE xReturn;
01803
01804     /* THIS FUNCTION MUST BE CALLED WITH INTERRUPTS DISABLED OR THE
01805     SCHEDULER SUSPENDED. It can also be called from within an ISR. */
01806
01807     /* The event list is sorted in priority order, so we can remove the
01808     first in the list, remove the TCB from the delayed list, and add
01809     it to the ready list.
01810
01811     If an event is for a queue that is locked then this function will never
01812     get called - the lock count on the queue will get modified instead. This
01813     means we can always expect exclusive access to the event list here.
01814
01815     This function assumes that a check has already been made to ensure that
01816     pxEventList is not empty. */
01817     pxUnblockedTCB = ( tskTCB * ) listGET_OWNER_OF_HEAD_ENTRY( pxEventList );
01818     configASSERT( pxUnblockedTCB );
01819     vListRemove( &( pxUnblockedTCB->xEventListItem ) );
01820
01821     if( uxSchedulerSuspended == ( unsigned portBASE_TYPE ) pdFALSE )
01822     {
01823         vListRemove( &( pxUnblockedTCB->xGenericListItem ) );

```

```

01824     prvAddTaskToReadyQueue( pxUnblockedTCB );
01825 }
01826 else
01827 {
01828     /* We cannot access the delayed or ready lists, so will hold this
01829     task pending until the scheduler is resumed. */
01830     vListInsertEnd( ( xList * ) &( xPendingReadyList ), &( pxUnblockedTCB->xEventListItem ) );
01831 }
01832
01833 if( pxUnblockedTCB->uxPriority >= pxCurrentTCB->uxPriority )
01834 {
01835     /* Return true if the task removed from the event list has
01836     a higher priority than the calling task. This allows
01837     the calling task to know if it should force a context
01838     switch now. */
01839     xReturn = pdTRUE;
01840 }
01841 else
01842 {
01843     xReturn = pdFALSE;
01844 }
01845
01846 return xReturn;
01847 }
01848 /*-----*/
01849
01850 void vTaskSetTimeOutState( xTimeOutType * const pxTimeOut )
01851 {
01852     configASSERT( pxTimeOut );
01853     pxTimeOut->xOverflowCount = xNumOfOverflows;
01854     pxTimeOut->xTimeOnEntering = xTickCount;
01855 }
01856 /*-----*/
01857
01858 portBASE_TYPE xTaskCheckForTimeOut( xTimeOutType * const pxTimeOut, portTickType * const pxTicksToWait
01859 )
01860 {
01861     portBASE_TYPE xReturn;
01862
01863     configASSERT( pxTimeOut );
01864     configASSERT( pxTicksToWait );
01865
01866     taskENTER_CRITICAL();
01867 {
01868     #if ( INCLUDE_vTaskSuspend == 1 )
01869         /* If INCLUDE_vTaskSuspend is set to 1 and the block time specified is
01870         the maximum block time then the task should block indefinitely, and
01871         therefore never time out. */
01872         if( *pxTicksToWait == portMAX_DELAY )
01873         {
01874             xReturn = pdFALSE;
01875         }
01876         else /* We are not blocking indefinitely, perform the checks below. */
01877     #endif
01878
01879     if( ( xNumOfOverflows != pxTimeOut->xOverflowCount ) && ( ( portTickType ) xTickCount >= ( portTickType ) pxTimeOut->xTimeOnEntering ) )
01880     {
01881         /* The tick count is greater than the time at which vTaskSetTimeout()
01882         was called, but has also overflowed since vTaskSetTimeout() was called.
01883         It must have wrapped all the way around and gone past us again. This
01884         passed since vTaskSetTimeout() was called. */
01885         xReturn = pdTRUE;
01886     }
01887     else if( ( ( portTickType ) ( ( portTickType ) xTickCount - ( portTickType ) pxTimeOut->xTimeOnEntering ) ) < ( portTickType ) *pxTicksToWait )
01888     {
01889         /* Not a genuine timeout. Adjust parameters for time remaining. */
01890         *pxTicksToWait -= ( ( portTickType ) xTickCount - ( portTickType ) pxTimeOut->xTimeOnEntering );
01891         vTaskSetTimeOutState( pxTimeOut );
01892         xReturn = pdFALSE;
01893     }
01894     {
01895         xReturn = pdTRUE;
01896     }
01897 }
01898 taskEXIT_CRITICAL();
01899
01900 return xReturn;
01901 }
01902 /*-----*/
01903
01904 void vTaskMissedYield( void )
01905 {
01906     xMissedYield = pdTRUE;

```

```

01907 }
01908
01909 /*
01910 * -----
01911 * The Idle task.
01912 * -----
01913 *
01914 * The portTASK_FUNCTION() macro is used to allow port/compiler specific
01915 * language extensions. The equivalent prototype for this function is:
01916 *
01917 * void prvIdleTask( void *pvParameters );
01918 *
01919 */
01920 static portTASK_FUNCTION( prvIdleTask, pvParameters )
01921 {
01922     /* Stop warnings. */
01923     ( void ) pvParameters;
01924
01925     for( ;; )
01926     {
01927         /* See if any tasks have been deleted. */
01928         prvCheckTasksWaitingTermination();
01929
01930         #if ( configUSE_PREEMPTION == 0 )
01931         {
01932             /* If we are not using preemption we keep forcing a task switch to
01933             see if any other task has become available. If we are using
01934             preemption we don't need to do this as any task becoming available
01935             will automatically get the processor anyway. */
01936             taskYIELD();
01937         }
01938         #endif
01939
01940         #if ( ( configUSE_PREEMPTION == 1 ) && ( configIDLE_SHOULD_YIELD == 1 ) )
01941         {
01942             /* When using preemption tasks of equal priority will be
01943             timesliced. If a task that is sharing the idle priority is ready
01944             to run then the idle task should yield before the end of the
01945             timeslice.
01946
01947             A critical region is not required here as we are just reading from
01948             the list, and an occasional incorrect value will not matter. If
01949             the ready list at the idle priority contains more than one task
01950             then a task other than the idle task is ready to execute. */
01951             if( listCURRENT_LIST_LENGTH( &( pxReadyTasksLists[ tskIDLE_PRIORITY ] ) ) > ( unsigned
01952                 portBASE_TYPE ) 1 )
01953             {
01954                 taskYIELD();
01955             }
01956         #endif
01957
01958         #if ( configUSE_IDLE_HOOK == 1 )
01959         {
01960             extern void vApplicationIdleHook( void );
01961
01962             /* Call the user defined function from within the idle task. This
01963             allows the application designer to add background functionality
01964             without the overhead of a separate task.
01965             NOTE: vApplicationIdleHook() MUST NOT, UNDER ANY CIRCUMSTANCES,
01966             CALL A FUNCTION THAT MIGHT BLOCK. */
01967             vApplicationIdleHook();
01968         }
01969     #endif
01970 }
01971 */lint !e715 pvParameters is not accessed but all task functions require the same prototype. */
01972
01973
01974
01975
01976
01977
01978
01979 /*
01980 * File private functions documented at the top of the file.
01981 */
01982
01983
01984
01985 static void prvInitialiseTCBVariables( tskTCB *pxTCB, const signed char * const pcName, unsigned
01986                                         portBASE_TYPE uxPriority, const xMemoryRegion * const xRegions, unsigned short usStackDepth )
01987 {
01988     /* Store the function name in the TCB. */
01989     #if configMAX_TASK_NAME_LEN > 1
01990     {
01991         /* Don't bring strncpy into the build unnecessarily. */
01992         strncpy( ( char * ) pxTCB->pctcTaskName, ( const char * ) pcName, ( unsigned short )

```

```

    configMAX_TASK_NAME_LEN );
01992 }
01993 #endif
01994 pxTCB->pcTaskName[ ( unsigned short ) configMAX_TASK_NAME_LEN - ( unsigned short ) 1 ] = ( signed
01995 char ) '\0';
01996 /* This is used as an array index so must ensure it's not too large. First
01997 remove the privilege bit if one is present. */
01998 if( uxPriority >= configMAX_PRIORITIES )
01999 {
02000     uxPriority = configMAX_PRIORITIES - ( unsigned portBASE_TYPE ) 1U;
02001 }
02002
02003 pxTCB->uxPriority = uxPriority;
02004 #if ( configUSE_MUTEXES == 1 )
02005 {
02006     pxTCB->uxBasePriority = uxPriority;
02007 }
02008 #endif
02009
02010 vListInitialiseItem( &( pxTCB->xGenericListItem ) );
02011 vListInitialiseItem( &( pxTCB->xEventListItem ) );
02012
02013 /* Set the pxTCB as a link back from the xListItem. This is so we can get
02014 back to the containing TCB from a generic item in a list. */
02015 listSET_LIST_ITEM_OWNER( &( pxTCB->xGenericListItem ), pxTCB );
02016
02017 /* Event lists are always in priority order. */
02018 listSET_LIST_ITEM_VALUE( &( pxTCB->xEventListItem ), configMAX_PRIORITIES - ( portTickType )
02019 uxPriority );
02020 listSET_LIST_ITEM_OWNER( &( pxTCB->xEventListItem ), pxTCB );
02021
02022 #if ( portCRITICAL_NESTING_IN_TCB == 1 )
02023 {
02024     pxTCB->uxCriticalNesting = ( unsigned portBASE_TYPE ) 0U;
02025 }
02026 #endif
02027 #if ( configUSE_APPLICATION_TASK_TAG == 1 )
02028 {
02029     pxTCB->pxTaskTag = NULL;
02030 }
02031 #endif
02032
02033 #if ( configGENERATE_RUN_TIME_STATS == 1 )
02034 {
02035     pxTCB->ulRunTimeCounter = 0UL;
02036 }
02037 #endif
02038
02039 #if ( portUSING_MPU_WRAPPERS == 1 )
02040 {
02041     vPortStoreTaskMPUSettings( &( pxTCB->xMPUSettings ), xRegions, pxTCB->pxStack, usStackDepth );
02042 }
02043 #else
02044 {
02045     ( void ) xRegions;
02046     ( void ) usStackDepth;
02047 }
02048 #endif
02049 }
02050 /*-----*/
02051
02052 #if ( portUSING_MPU_WRAPPERS == 1 )
02053
02054 void vTaskAllocateMPURegions( xTaskHandle xTaskToModify, const xMemoryRegion * const xRegions )
02055 {
02056     tskTCB *pxTCB;
02057
02058     if( xTaskToModify == pxCurrentTCB )
02059     {
02060         xTaskToModify = NULL;
02061     }
02062
02063     /* If null is passed in here then we are deleting ourselves. */
02064     pxTCB = prvGetTCBFromHandle( xTaskToModify );
02065
02066     vPortStoreTaskMPUSettings( &( pxTCB->xMPUSettings ), xRegions, NULL, 0 );
02067 }
02068 /*-----*/
02069 #endif
02070
02071 static void prvInitialiseTaskLists( void )
02072 {
02073     unsigned portBASE_TYPE uxPriority;
02074
02075     for( uxPriority = ( unsigned portBASE_TYPE ) 0U; uxPriority < configMAX_PRIORITIES; uxPriority++ )

```

```

02076     {
02077         vListInitialise( ( xList * ) &( pxReadyTasksLists[ uxPriority ] ) );
02078     }
02079
02080     vListInitialise( ( xList * ) &xDelayedTaskList1 );
02081     vListInitialise( ( xList * ) &xDelayedTaskList2 );
02082     vListInitialise( ( xList * ) &xPendingReadyList );
02083
02084 #if ( INCLUDE_vTaskDelete == 1 )
02085 {
02086     vListInitialise( ( xList * ) &xTasksWaitingTermination );
02087 }
02088#endif
02089
02090 #if ( INCLUDE_vTaskSuspend == 1 )
02091 {
02092     vListInitialise( ( xList * ) &xSuspendedTaskList );
02093 }
02094#endif
02095
02096 /* Start with pxDelayedTaskList using list1 and the pxOverflowDelayedTaskList
02097 using list2. */
02098 pxDelayedTaskList = &xDelayedTaskList1;
02099 pxOverflowDelayedTaskList = &xDelayedTaskList2;
02100 }
02101 /*-----*/
02102
02103 static void prvCheckTasksWaitingTermination( void )
02104 {
02105     #if ( INCLUDE_vTaskDelete == 1 )
02106     {
02107         portBASE_TYPE xListIsEmpty;
02108
02109         /* ucTasksDeleted is used to prevent vTaskSuspendAll() being called
02110 too often in the idle task. */
02111         if( uxTasksDeleted > ( unsigned portBASE_TYPE ) 0U )
02112         {
02113             vTaskSuspendAll();
02114             xListIsEmpty = listLIST_IS_EMPTY( &xTasksWaitingTermination );
02115             xTaskResumeAll();
02116
02117             if( xListIsEmpty == pdFALSE )
02118             {
02119                 tskTCB *pxTCB;
02120
02121                 taskENTER_CRITICAL();
02122                 {
02123                     pxTCB = ( tskTCB * ) listGET_OWNER_OF_HEAD_ENTRY( ( ( xList * )
02124 &xTasksWaitingTermination ) );
02125                     vListRemove( &( pxTCB->xGenericListItem ) );
02126                     --uxCurrentNumberOfTasks;
02127                     --uxTasksDeleted;
02128                 }
02129                 taskEXIT_CRITICAL();
02130
02131                 prvDeleteTCB( pxTCB );
02132             }
02133         }
02134     }
02135 }
02136 /*-----*/
02137
02138 static void prvAddCurrentTaskToDelayedList( portTickType xTimeToWake )
02139 {
02140     /* The list item will be inserted in wake time order. */
02141     listSET_LIST_ITEM_VALUE( &( pxCurrentTCB->xGenericListItem ), xTimeToWake );
02142
02143     if( xTimeToWake < xTickCount )
02144     {
02145         /* Wake time has overflowed. Place this item in the overflow list. */
02146         vListInsert( ( xList * ) pxOverflowDelayedTaskList, ( xListItem * ) &( pxCurrentTCB->xGenericListItem ) );
02147     }
02148     else
02149     {
02150         /* The wake time has not overflowed, so we can use the current block list. */
02151         vListInsert( ( xList * ) pxDelayedTaskList, ( xListItem * ) &( pxCurrentTCB->xGenericListItem ) );
02152
02153         /* If the task entering the blocked state was placed at the head of the
02154 list of blocked tasks then xNextTaskUnblockTime needs to be updated
02155 too. */
02156         if( xTimeToWake < xNextTaskUnblockTime )
02157         {
02158             xNextTaskUnblockTime = xTimeToWake;
02159         }

```

```

02160 }
02161 }
02162 /*-----*/
02163
02164 static tskTCB *prvAllocateTCBAndStack( unsigned short usStackDepth, portSTACK_TYPE *pxStackBuffer )
02165 {
02166     tskTCB *pxNewTCB;
02167
02168     /* Allocate space for the TCB. Where the memory comes from depends on
02169     the implementation of the port malloc function. */
02170     pxNewTCB = ( tskTCB * ) pvPortMalloc( sizeof( tskTCB ) );
02171
02172     if( pxNewTCB != NULL )
02173     {
02174         /* Allocate space for the stack used by the task being created.
02175         The base of the stack memory stored in the TCB so the task can
02176         be deleted later if required. */
02177         pxNewTCB->pxStack = ( portSTACK_TYPE * ) pvPortMallocAligned( ( ( size_t )usStackDepth ) *
02178             sizeof( portSTACK_TYPE ), pxStackBuffer );
02179
02180         if( pxNewTCB->pxStack == NULL )
02181         {
02182             /* Could not allocate the stack. Delete the allocated TCB. */
02183             vPortFree( pxNewTCB );
02184             pxNewTCB = NULL;
02185         }
02186         else
02187         {
02188             /* Just to help debugging. */
02189             memset( pxNewTCB->pxStack, ( int ) tskSTACK_FILL_BYTE, ( size_t ) usStackDepth * sizeof(
02190                 portSTACK_TYPE ) );
02191         }
02192     }
02193 }
02194 /*-----*/
02195
02196 #if ( configUSE_TRACE_FACILITY == 1 )
02197
02198     static void prvListTaskWithinSingleList( const signed char *pcWriteBuffer, xList *pxList, signed
02199     char cStatus )
02200     {
02201         volatile tskTCB *pxNextTCB, *pxFirstTCB;
02202         unsigned short usStackRemaining;
02203
02204         /* Write the details of all the TCB's in pxList into the buffer. */
02205         listGET_OWNER_OF_NEXT_ENTRY( pxFirstTCB, pxList );
02206         do
02207         {
02208             listGET_OWNER_OF_NEXT_ENTRY( pxNextTCB, pxList );
02209             #if ( portSTACK_GROWTH > 0 )
02210             {
02211                 usStackRemaining = usTaskCheckFreeStackSpace( ( unsigned char * )
02212                     pxNextTCB->pxEndOfStack );
02213             }
02214             #else
02215             {
02216                 usStackRemaining = usTaskCheckFreeStackSpace( ( unsigned char * ) pxNextTCB->pxStack );
02217             }
02218             #endif
02219             sprintf( pcStatusString, ( char * ) "%s\t%c\t%u\t%u\t%u\r\n", pxNextTCB->pcTaskName,
02220             cStatus, ( unsigned int ) pxNextTCB->uxPriority, usStackRemaining, ( unsigned int )
02221             pxNextTCB->uxTCBNumber );
02222             strcat( ( char * ) pcWriteBuffer, ( char * ) pcStatusString );
02223         } while( pxNextTCB != pxFirstTCB );
02224     }
02225 #endif
02226 /*-----*/
02227 #if ( configGENERATE_RUN_TIME_STATS == 1 )
02228
02229     static void prvGenerateRunTimeStatsForTasksInList( const signed char *pcWriteBuffer, xList
02230     *pxList, unsigned long ulTotalRunTime )
02231     {
02232         volatile tskTCB *pxNextTCB, *pxFirstTCB;
02233         unsigned long ulStatsAsPercentage;
02234
02235         /* Write the run time stats of all the TCB's in pxList into the buffer. */
02236         listGET_OWNER_OF_NEXT_ENTRY( pxFirstTCB, pxList );
02237         do
02238         {
02239             /* Get next TCB in from the list. */

```

```

02239     listGET_OWNER_OF_NEXT_ENTRY( pxNextTCB, pxList );
02240
02241     /* Divide by zero check. */
02242     if( ulTotalRunTime > OUL )
02243     {
02244         /* Has the task run at all? */
02245         if( pxNextTCB->ulRunTimeCounter == OUL )
02246         {
02247             /* The task has used no CPU time at all. */
02248             sprintf( pcStatsString, ( char * ) "%s\t\t0\t\t0%%\r\n", pxNextTCB->pcTaskName );
02249         }
02250     else
02251     {
02252         /* What percentage of the total run time has the task used?
02253            This will always be rounded down to the nearest integer.
02254            ulTotalRunTime has already been divided by 100. */
02255         ulStatsAsPercentage = pxNextTCB->ulRunTimeCounter / ulTotalRunTime;
02256
02257         if( ulStatsAsPercentage > OUL )
02258         {
02259             #ifdef portLU_PRINTF_SPECIFIER_REQUIRED
02260             {
02261                 sprintf( pcStatsString, ( char * ) "%s\t\t%lu\t\t%lu%%\r\n",
02262                     pxNextTCB->pcTaskName, pxNextTCB->ulRunTimeCounter, ulStatsAsPercentage );
02263             }
02264         #else
02265             /* sizeof( int ) == sizeof( long ) so a smaller
02266                printf() library can be used. */
02267             sprintf( pcStatsString, ( char * ) "%s\t\t%u\t\t%u%%\r\n",
02268                 pxNextTCB->pcTaskName, ( unsigned int ) pxNextTCB->ulRunTimeCounter, ( unsigned int )
02269                 ulStatsAsPercentage );
02270         }
02271     #endif
02272     }
02273     /* If the percentage is zero here then the task has
02274        consumed less than 1% of the total run time. */
02275     #ifdef portLU_PRINTF_SPECIFIER_REQUIRED
02276     {
02277         sprintf( pcStatsString, ( char * ) "%s\t\t%lu\t\t<1%%\r\n",
02278             pxNextTCB->pcTaskName, pxNextTCB->ulRunTimeCounter );
02279     }
02280     #else
02281         /* sizeof( int ) == sizeof( long ) so a smaller
02282            printf() library can be used. */
02283         sprintf( pcStatsString, ( char * ) "%s\t\t%u\t\t<1%%\r\n",
02284             pxNextTCB->pcTaskName, ( unsigned int ) pxNextTCB->ulRunTimeCounter );
02285     }
02286     }
02287
02288     strcat( ( char * ) pcWriteBuffer, ( char * ) pcStatsString );
02289 }
02290
02291 } while( pxNextTCB != pxFirstTCB );
02292
02293 }
02294
02295 #endif
02296 /*-----*/
02297
02298 #if ( ( configUSE_TRACE_FACILITY == 1 ) || ( INCLUDE_uxTaskGetStackHighWaterMark == 1 ) )
02299
02300     static unsigned short usTaskCheckFreeStackSpace( const unsigned char * pucStackByte )
02301     {
02302         register unsigned short usCount = 0U;
02303
02304         while( *pucStackByte == tskSTACK_FILL_BYTE )
02305         {
02306             pucStackByte -= portSTACK_GROWTH;
02307             usCount++;
02308         }
02309
02310         usCount /= sizeof( portSTACK_TYPE );
02311
02312         return usCount;
02313     }
02314
02315 #endif
02316 /*-----*/
02317
02318 #if ( INCLUDE_uxTaskGetStackHighWaterMark == 1 )
02319     unsigned portBASE_TYPE uxTaskGetStackHighWaterMark( xTaskHandle xTask )
02320

```

```
02321     {
02322     tskTCB *pxTCB;
02323     unsigned char *pcEndOfStack;
02324     unsigned portBASE_TYPE uxReturn;
02325
02326     pxTCB = prvGetTCBFromHandle( xTask );
02327
02328     #if portSTACK_GROWTH < 0
02329     {
02330         pcEndOfStack = ( unsigned char * ) pxTCB->pxStack;
02331     }
02332     #else
02333     {
02334         pcEndOfStack = ( unsigned char * ) pxTCB->pxEndOfStack;
02335     }
02336     #endif
02337
02338     uxReturn = ( unsigned portBASE_TYPE ) usTaskCheckFreeStackSpace( pcEndOfStack );
02339
02340     return uxReturn;
02341 }
02342
02343 #endif
02344 /*-----*/
02345
02346 #if ( INCLUDE_vTaskDelete == 1 )
02347
02348     static void prvDeleteTCB( tskTCB *pxTCB )
02349     {
02350         /* Free up the memory allocated by the scheduler for the task. It is up to
02351         the task to free any memory allocated at the application level. */
02352         vPortFreeAligned( pxTCB->pxStack );
02353         vPortFree( pxTCB );
02354     }
02355
02356 #endif
02357
02358 /*-----*/
02359 /*-----*/
02360
02361 #if ( ( INCLUDE_xTaskGetCurrentTaskHandle == 1 ) || ( configUSE_MUTEXES == 1 ) )
02362
02363     xTaskHandle xTaskGetCurrentTaskHandle( void )
02364     {
02365         xTaskHandle xReturn;
02366
02367         /* A critical section is not required as this is not called from
02368         an interrupt and the current TCB will always be the same for any
02369         individual execution thread. */
02370         xReturn = pxCurrentTCB;
02371
02372         return xReturn;
02373     }
02374
02375 #endif
02376
02377 /*-----*/
02378
02379 #if ( ( INCLUDE_xTaskGetSchedulerState == 1 ) || ( configUSE_TIMERS == 1 ) )
02380
02381     portBASE_TYPE xTaskGetSchedulerState( void )
02382     {
02383         portBASE_TYPE xReturn;
02384
02385         if( xSchedulerRunning == pdFALSE )
02386         {
02387             xReturn = taskSCHEDULER_NOT_STARTED;
02388         }
02389         else
02390         {
02391             if( uxSchedulerSuspended == ( unsigned portBASE_TYPE ) pdFALSE )
02392             {
02393                 xReturn = taskSCHEDULER_RUNNING;
02394             }
02395             else
02396             {
02397                 xReturn = taskSCHEDULER_SUSPENDED;
02398             }
02399         }
02400
02401         return xReturn;
02402     }
02403
02404 #endif
02405 /*-----*/
02406
02407 #if ( configUSE_MUTEXES == 1 )
```

```

02408 void vTaskPriorityInherit( xTaskHandle * const pxMutexHolder )
02409 {
02410     tskTCB * const pxTCB = ( tskTCB * ) pxMutexHolder;
02411     configASSERT( pxMutexHolder );
02412
02413     if( pxTCB->uxPriority < pxCurrentTCB->uxPriority )
02414     {
02415         /* Adjust the mutex holder state to account for its new priority. */
02416         listSET_LIST_ITEM_VALUE( &( pxTCB->xEventListItem ), configMAX_PRIORITIES - ( portTickType
02417             ) pxCurrentTCB->uxPriority );
02418
02419         /* If the task being modified is in the ready state it will need to
02420         be moved in to a new list. */
02421         if( listIS_CONTAINED_WITHIN( &( pxReadyTasksLists[ pxTCB->uxPriority ] ), &( pxTCB->xGenericListItem ) ) != pdFALSE )
02422         {
02423             vListRemove( &( pxTCB->xGenericListItem ) );
02424
02425             /* Inherit the priority before being moved into the new list. */
02426             pxTCB->uxPriority = pxCurrentTCB->uxPriority;
02427             prvAddTaskToReadyQueue( pxTCB );
02428         }
02429     }
02430     else
02431     {
02432         /* Just inherit the priority. */
02433         pxTCB->uxPriority = pxCurrentTCB->uxPriority;
02434     }
02435 }
02436 }
02437
02438 #endif
02439 /*-----*/
02440
02441 #if ( configUSE_MUTEXES == 1 )
02442
02443     void vTaskPriorityDisinherit( xTaskHandle * const pxMutexHolder )
02444     {
02445         tskTCB * const pxTCB = ( tskTCB * ) pxMutexHolder;
02446
02447         if( pxMutexHolder != NULL )
02448         {
02449             if( pxTCB->uxPriority != pxTCB->uxBasePriority )
02450             {
02451                 /* We must be the running task to be able to give the mutex back.
02452                 Remove ourselves from the ready list we currently appear in. */
02453                 vListRemove( &( pxTCB->xGenericListItem ) );
02454
02455                 /* Disinherit the priority before adding ourselves into the new
02456                 ready list. */
02457                 pxTCB->uxPriority = pxTCB->uxBasePriority;
02458                 listSET_LIST_ITEM_VALUE( &( pxTCB->xEventListItem ), configMAX_PRIORITIES - ( portTickType
02459                     ) pxTCB->uxPriority );
02460                 prvAddTaskToReadyQueue( pxTCB );
02461             }
02462         }
02463     }
02464 #endif
02465 /*-----*/
02466
02467 #if ( portCRITICAL_NESTING_IN_TCB == 1 )
02468
02469     void vTaskEnterCritical( void )
02470     {
02471         portDISABLE_INTERRUPTS();
02472
02473         if( xSchedulerRunning != pdFALSE )
02474         {
02475             ( pxCurrentTCB->uxCriticalNesting )++;
02476         }
02477     }
02478
02479 #endif
02480 /*-----*/
02481
02482 #if ( portCRITICAL_NESTING_IN_TCB == 1 )
02483
02484     void vTaskExitCritical( void )
02485     {
02486         if( xSchedulerRunning != pdFALSE )
02487         {
02488             if( pxCurrentTCB->uxCriticalNesting > 0U )
02489             {
02490                 ( pxCurrentTCB->uxCriticalNesting )--;
02491             }
02492         }
02493     }
02494
02495 #endif

```

```

02492     if( pxCurrentTCB->uxCriticalNesting == 0U )
02493     {
02494         portENABLE_INTERRUPTS();
02495     }
02496 }
02497 }
02498 }
02499
02500 #endif
02501 /*-----*/
02502
02503
02504
02505

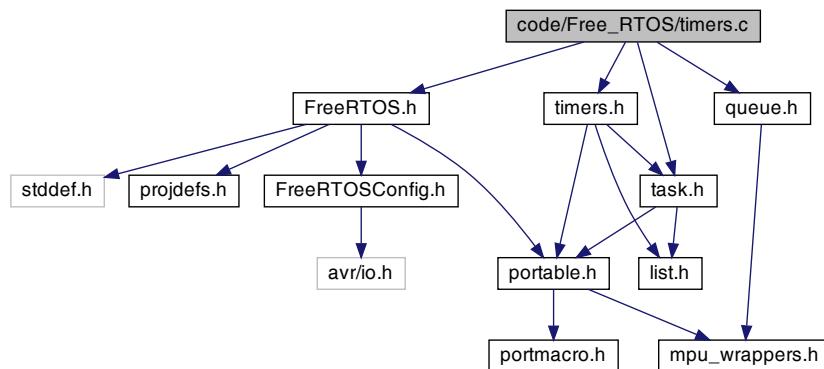
```

22.39 code/FreeRTOS/timers.c File Reference

```

#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
Include dependency graph for timers.c:

```



Macros

- #define MPU_WRAPPERS_INCLUDED_FROM_API_FILE

22.39.1 Macro Definition Documentation

22.39.1.1 MPU_WRAPPERS_INCLUDED_FROM_API_FILE #define MPU_WRAPPERS_INCLUDED_FROM_API_FILE

Definition at line 57 of file [timers.c](#).

22.40 timers.c

```

00001 /*          *
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005 ****
00006 *
00007 *     FreeRTOS tutorial books are available in pdf and paperback. *
00008 *     Complete, revised, and edited pdf reference manuals are also *
00009 *     available. *
00010 *
00011 *     Purchasing FreeRTOS documentation will not only help you, by *
00012 *     ensuring you get running as quickly as possible and with an *
00013 *     in-depth knowledge of how to use FreeRTOS, it will also help *
00014 *     the FreeRTOS project to continue with its mission of providing *
00015 *     professional grade, cross platform, de facto standard solutions *
00016 *     for microcontrollers - completely free of charge! *
00017 *
00018 *     >>> See http://www.FreeRTOS.org/Documentation for details. << *
00019 *
00020 *     Thank you for using FreeRTOS, and thank you for your support! *
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 =>>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039 by writing to Richard Barry, contact details for whom are available on the
00040 FreeRTOS WEB site.
00041
00042 1 tab == 4 spaces!
00043
00044 http://www.FreeRTOS.org - Documentation, latest information, license and
00045 contact details.
00046
00047 http://www.SafeRTOS.com - A version that is certified for use in safety
00048 critical systems.
00049
00050 http://www.OpenRTOS.com - Commercial support, development, porting,
00051 licensing and training services.
00052 */
00053
00054 /* Defining MPU_WRAPPERS_INCLUDED_FROM_API_FILE prevents task.h from redefining
00055 all the API functions to use the MPU wrappers. That should only be done when
00056 task.h is included from an application file. */
00057 #define MPU_WRAPPERS_INCLUDED_FROM_API_FILE
00058
00059 #include "FreeRTOS.h"
00060 #include "task.h"
00061 #include "queue.h"
00062 #include "timers.h"
00063
00064 #undef MPU_WRAPPERS_INCLUDED_FROM_API_FILE
00065
00066 /* This entire source file will be skipped if the application is not configured
00067 to include software timer functionality. This #if is closed at the very bottom
00068 of this file. If you want to include software timer functionality then ensure
00069 configUSE_TIMERS is set to 1 in FreeRTOSConfig.h. */
00070 #if ( configUSE_TIMERS == 1 )
00071
00072 /* Misc definitions. */
00073 #define tmrNO_DELAY      ( portTickType ) 0U
00074
00075 /* The definition of the timers themselves. */
00076 typedef struct tmrTimerControl
00077 {
00078     const signed char          *pcTimerName;      /*<< Text name. This is not used by the kernel, it is
00079     included simply to make debugging easier. */
00080     xListItem                  xTimerListItem;    /*<< Standard linked list item as used by all kernel
00081     features for event management. */
00082     portTickType                xTimerPeriodInTicks; /*<< How quickly and often the timer expires. */
00083     unsigned portBASE_TYPE      uxAutoReload;     /*<< Set to pdTRUE if the timer should be automatically
00084     restarted once expired. Set to pdFALSE if the timer is, in effect, a one shot timer. */
00085     void                      *pvTimerID;        /*<< An ID to identify the timer. This allows the

```

```

        timer to be identified when the same callback is used for multiple timers. */
00083     tmrTIMER_CALLBACK      pxCallbackFunction; /*<< The function that will be called when the timer
00084 } xTIMER;
00085
00086 /* The definition of messages that can be sent and received on the timer
00087 queue. */
00088 typedef struct tmrTimerQueueMessage
00089 {
00090     portBASE_TYPE          xMessageID;           /*<< The command being sent to the timer service task.
00091 */
00092     portTickType            xMessageValue;        /*<< An optional value used by a subset of commands,
00093     for example, when changing the period of a timer. */
00094     xTIMER *                pxTimer;              /*<< The timer to which the command will be applied. */
00095 } xTIMER_MESSAGE;
00096
00097 /* The list in which active timers are stored. Timers are referenced in expire
00098 time order, with the nearest expiry time at the front of the list. Only the
00099 timer service task is allowed to access xActiveTimerList. */
00100 PRIVILEGED_DATA static xList xActiveTimerList1;
00101 PRIVILEGED_DATA static xList xActiveTimerList2;
00102 PRIVILEGED_DATA static xList *pxCurrentTimerList;
00103 PRIVILEGED_DATA static xList *pxOverflowTimerList;
00104
00105 /* A queue that is used to send commands to the timer service task. */
00106 PRIVILEGED_DATA static xQueueHandle xTimerQueue = NULL;
00107
00108 #if ( INCLUDE_xTimerGetTimerDaemonTaskHandle == 1 )
00109     PRIVILEGED_DATA static xTaskHandle xTimerTaskHandle = NULL;
00110
00111 #endif
00112
00113 /*****
00114 */
00115 /*
00116 * Initialise the infrastructure used by the timer service task if it has not
00117 * been initialised already.
00118 */
00119 static void prvCheckForValidListAndQueue( void ) PRIVILEGED_FUNCTION;
00120
00121 /*
00122 * The timer service task (daemon). Timer functionality is controlled by this
00123 * task. Other tasks communicate with the timer service task using the
00124 * xTimerQueue queue.
00125 */
00126 static void prvTimerTask( void *pvParameters ) PRIVILEGED_FUNCTION;
00127
00128 /*
00129 * Called by the timer service task to interpret and process a command it
00130 * received on the timer queue.
00131 */
00132 static void prvProcessReceivedCommands( void ) PRIVILEGED_FUNCTION;
00133
00134 /*
00135 * Insert the timer into either xActiveTimerList1, or xActiveTimerList2,
00136 * depending on if the expire time causes a timer counter overflow.
00137 */
00138 static portBASE_TYPE prvInsertTimerInActiveList( xTIMER *pxTimer, portTickType xNextExpiryTime,
00139                                                 portTickType xTimeNow, portTickType xCommandTime ) PRIVILEGED_FUNCTION;
00140
00141 /*
00142 * An active timer has reached its expire time. Reload the timer if it is an
00143 * auto reload timer, then call its callback.
00144 */
00145 static void prvProcessExpiredTimer( portTickType xNextExpireTime, portTickType xTimeNow )
00146     PRIVILEGED_FUNCTION;
00147
00148 /*
00149 * The tick count has overflowed. Switch the timer lists after ensuring the
00150 * current timer list does not still reference some timers.
00151 */
00152 static void prvSwitchTimerLists( portTickType xLastTime ) PRIVILEGED_FUNCTION;
00153
00154 /*
00155 * Obtain the current tick count, setting *pxTimerListsWereSwitched to pdTRUE
00156 * if a tick count overflow occurred since prvSampleTimeNow() was last called.
00157 */
00158 static portTickType prvSampleTimeNow( portBASE_TYPE *pxTimerListsWereSwitched ) PRIVILEGED_FUNCTION;
00159
00160 /*
00161 * If the timer list contains any active timers then return the expire time of
00162 * the timer that will expire first and set *pxListWasEmpty to false. If the
00163 * timer list does not contain any timers then return 0 and set *pxListWasEmpty
00164 * to pdTRUE.
00165 */

```

```

00164 static portTickType prvGetNextExpireTime( portBASE_TYPE *pxListWasEmpty ) PRIVILEGED_FUNCTION;
00165
00166 /*
00167  * If a timer has expired, process it. Otherwise, block the timer service task
00168  * until either a timer does expire or a command is received.
00169 */
00170 static void prvProcessTimerOrBlockTask( portTickType xNextExpireTime, portBASE_TYPE xListWasEmpty )
00171     PRIVILEGED_FUNCTION;
00172 /*-----*/
00173
00174 portBASE_TYPE xTimerCreateTimerTask( void )
00175 {
00176     portBASE_TYPE xReturn = pdFAIL;
00177
00178     /* This function is called when the scheduler is started if
00179      configUSE_TIMERS is set to 1. Check that the infrastructure used by the
00180      timer service task has been created/initialised. If timers have already
00181      been created then the initialisation will already have been performed. */
00182     prvCheckForValidListAndQueue();
00183
00184     if( xTimerQueue != NULL )
00185     {
00186         #if ( INCLUDE_xTimerGetTimerDaemonTaskHandle == 1 )
00187         {
00188             /* Create the timer task, storing its handle in xTimerTaskHandle so
00189              it can be returned by the xTimerGetTimerDaemonTaskHandle() function. */
00190             xReturn = xTaskCreate( prvTimerTask, ( const signed char * ) "Tmr Svc", ( unsigned short )
00191 configTIMER_TASK_STACK_DEPTH, NULL, ( unsigned portBASE_TYPE ) configTIMER_TASK_PRIORITY,
00192 &xTimerTaskHandle );
00193         }
00194         #else
00195         {
00196             /* Create the timer task without storing its handle. */
00197             xReturn = xTaskCreate( prvTimerTask, ( const signed char * ) "Tmr Svc", ( unsigned short )
00198 configTIMER_TASK_STACK_DEPTH, NULL, ( unsigned portBASE_TYPE ) configTIMER_TASK_PRIORITY, NULL );
00199         }
00200         #endif
00201     }
00202     configASSERT( xReturn );
00203     return xReturn;
00204 }
00205 /*-----*/
00206 xTimerHandle xTimerCreate( const signed char *pcTimerName, portTickType xTimerPeriodInTicks, unsigned
00207 portBASE_TYPE uxAutoReload, void *pvTimerID, tmrTIMER_CALLBACK pxCallbackFunction )
00208 {
00209     xTIMER *pxNewTimer;
00210
00211     /* Allocate the timer structure. */
00212     if( xTimerPeriodInTicks == ( portTickType ) 0U )
00213     {
00214         pxNewTimer = NULL;
00215         configASSERT( ( xTimerPeriodInTicks > 0 ) );
00216     }
00217     else
00218     {
00219         pxNewTimer = ( xTIMER * ) pvPortMalloc( sizeof( xTIMER ) );
00220         if( pxNewTimer != NULL )
00221         {
00222             /* Ensure the infrastructure used by the timer service task has been
00223              created/initialised. */
00224             prvCheckForValidListAndQueue();
00225
00226             /* Initialise the timer structure members using the function parameters. */
00227             pxNewTimer->pcTimerName = pcTimerName;
00228             pxNewTimer->xTimerPeriodInTicks = xTimerPeriodInTicks;
00229             pxNewTimer->uxAutoReload = uxAutoReload;
00230             pxNewTimer->pvTimerID = pvTimerID;
00231             pxNewTimer->pxCallbackFunction = pxCallbackFunction;
00232             vListInitialiseItem( &( pxNewTimer->xTimerListItem ) );
00233
00234             traceTIMER_CREATE( pxNewTimer );
00235         }
00236     }
00237
00238 }
00239
00240     return ( xTimerHandle ) pxNewTimer;
00241 }
00242 /*-----*/
00243
00244 portBASE_TYPE xTimerGenericCommand( xTimerHandle xTimer, portBASE_TYPE xCommandID, portTickType
00245 xOptionalValue, portBASE_TYPE *pxHigherPriorityTaskWoken, portTickType xBlockTime )

```

```

00245 {
00246     portBASE_TYPE xReturn = pdFAIL;
00247     xTIMER_MESSAGE xMessage;
00248
00249     /* Send a message to the timer service task to perform a particular action
00250     on a particular timer definition. */
00251     if( xTimerQueue != NULL )
00252     {
00253         /* Send a command to the timer service task to start the xTimer timer. */
00254         xMessage.xMessageID = xCommandID;
00255         xMessage.xMessageValue = xOptionalValue;
00256         xMessage.pxTimer = ( xTIMER * ) xTimer;
00257
00258         if( pxHigherPriorityTaskWoken == NULL )
00259         {
00260             if( xTaskGetSchedulerState() == taskSCHEDULER_RUNNING )
00261             {
00262                 xReturn = xQueueSendToBack( xTimerQueue, &xMessage, xBlockTime );
00263             }
00264             else
00265             {
00266                 xReturn = xQueueSendToBack( xTimerQueue, &xMessage, tmrNO_DELAY );
00267             }
00268         }
00269         else
00270         {
00271             xReturn = xQueueSendToBackFromISR( xTimerQueue, &xMessage, pxHigherPriorityTaskWoken );
00272         }
00273
00274         traceTIMER_COMMAND_SEND( xTimer, xCommandID, xOptionalValue, xReturn );
00275     }
00276
00277     return xReturn;
00278 }
00279 /*-----*/
00280
00281 #if ( INCLUDE_xTimerGetTimerDaemonTaskHandle == 1 )
00282
00283     xTaskHandle xTimerGetTimerDaemonTaskHandle( void )
00284     {
00285         /* If xTimerGetTimerDaemonTaskHandle() is called before the scheduler has been
00286         started, then xTimerTaskHandle will be NULL. */
00287         configASSERT( ( xTimerTaskHandle != NULL ) );
00288         return xTimerTaskHandle;
00289     }
00290
00291 #endif
00292 /*-----*/
00293
00294 static void prvProcessExpiredTimer( portTickType xNextExpireTime, portTickType xTimeNow )
00295 {
00296     xTIMER *pxTimer;
00297     portBASE_TYPE xResult;
00298
00299     /* Remove the timer from the list of active timers. A check has already
00300     been performed to ensure the list is not empty. */
00301     pxTimer = ( xTIMER * ) listGET_OWNER_OF_HEAD_ENTRY( pxCurrentTimerList );
00302     vListRemove( &( pxTimer->xTimerListItem ) );
00303     traceTIMER_EXPIRED( pxTimer );
00304
00305     /* If the timer is an auto reload timer then calculate the next
00306     expiry time and re-insert the timer in the list of active timers. */
00307     if( pxTimer->uxAutoReload == ( unsigned portBASE_TYPE ) pdTRUE )
00308     {
00309         /* This is the only time a timer is inserted into a list using
00310         a time relative to anything other than the current time. It
00311         will therefore be inserted into the correct list relative to
00312         the time this task thinks it is now, even if a command to
00313         switch lists due to a tick count overflow is already waiting in
00314         the timer queue. */
00315         if( prvInsertTimerInActiveList( pxTimer, ( xNextExpireTime + pxTimer->xTimerPeriodInTicks ),
00316             xTimeNow, xNextExpireTime ) == pdTRUE )
00317         {
00318             /* The timer expired before it was added to the active timer
00319             list. Reload it now. */
00320             xResult = xTimerGenericCommand( pxTimer, tmrCOMMAND_START, xNextExpireTime, NULL,
00321                 tmrNO_DELAY );
00322             configASSERT( xResult );
00323         }
00324
00325     /* Call the timer callback. */
00326     pxTimer->pxCallbackFunction( ( xTimerHandle ) pxTimer );
00327 }
00328 /*-----*/
00329

```

```

00330 static void prvTimerTask( void *pvParameters )
00331 {
00332     portTickType xNextExpireTime;
00333     portBASE_TYPE xListWasEmpty;
00334
00335     /* Just to avoid compiler warnings. */
00336     ( void ) pvParameters;
00337
00338     for( ; ; )
00339     {
00340         /* Query the timers list to see if it contains any timers, and if so,
00341         obtain the time at which the next timer will expire. */
00342         xNextExpireTime = prvGetNextExpireTime( &xListWasEmpty );
00343
00344         /* If a timer has expired, process it. Otherwise, block this task
00345         until either a timer does expire, or a command is received. */
00346         prvProcessTimerOrBlockTask( xNextExpireTime, xListWasEmpty );
00347
00348         /* Empty the command queue. */
00349         prvProcessReceivedCommands();
00350     }
00351 }
00352 /*-----*/
00353
00354 static void prvProcessTimerOrBlockTask( portTickType xNextExpireTime, portBASE_TYPE xListWasEmpty )
00355 {
00356     portTickType xTimeNow;
00357     portBASE_TYPE xTimerListsWereSwitched;
00358
00359     vTaskSuspendAll();
00360     {
00361         /* Obtain the time now to make an assessment as to whether the timer
00362         has expired or not. If obtaining the time causes the lists to switch
00363         then don't process this timer as any timers that remained in the list
00364         when the lists were switched will have been processed within the
00365         prvSampleTimeNow() function. */
00366         xTimeNow = prvSampleTimeNow( &xTimerListsWereSwitched );
00367         if( xTimerListsWereSwitched == pdFALSE )
00368         {
00369             /* The tick count has not overflowed, has the timer expired? */
00370             if( ( xListWasEmpty == pdFALSE ) && ( xNextExpireTime <= xTimeNow ) )
00371             {
00372                 xTaskResumeAll();
00373                 prvProcessExpiredTimer( xNextExpireTime, xTimeNow );
00374             }
00375             else
00376             {
00377                 /* The tick count has not overflowed, and the next expire
00378                 time has not been reached yet. This task should therefore
00379                 block to wait for the next expire time or a command to be
00380                 received - whichever comes first. The following line cannot
00381                 be reached unless xNextExpireTime > xTimeNow, except in the
00382                 case when the current timer list is empty. */
00383                 vQueueWaitForMessageRestricted( xTimerQueue, ( xNextExpireTime - xTimeNow ) );
00384
00385                 if( xTaskResumeAll() == pdFALSE )
00386                 {
00387                     /* Yield to wait for either a command to arrive, or the block time
00388                     to expire. If a command arrived between the critical section being
00389                     exited and this yield then the yield will not cause the task
00390                     to block. */
00391                     portYIELD_WITHIN_API();
00392                 }
00393             }
00394         }
00395         else
00396         {
00397             xTaskResumeAll();
00398         }
00399     }
00400 }
00401 /*-----*/
00402
00403 static portTickType prvGetNextExpireTime( portBASE_TYPE *pxListIsEmpty )
00404 {
00405     portTickType xNextExpireTime;
00406
00407     /* Timers are listed in expiry time order, with the head of the list
00408     referencing the task that will expire first. Obtain the time at which
00409     the timer with the nearest expiry time will expire. If there are no
00410     active timers then just set the next expire time to 0. That will cause
00411     this task to unblock when the tick count overflows, at which point the
00412     timer lists will be switched and the next expiry time can be
00413     re-assessed. */
00414     *pxListIsEmpty = listLIST_IS_EMPTY( pxCurrentTimerList );
00415     if( *pxListIsEmpty == pdFALSE )
00416     {

```

```

00417     xNextExpireTime = listGET_ITEM_VALUE_OF_HEAD_ENTRY( pxCurrentTimerList );
00418 }
00419 else
00420 {
00421     /* Ensure the task unblocks when the tick count rolls over. */
00422     xNextExpireTime = ( portTickType ) OU;
00423 }
00424
00425 return xNextExpireTime;
00426 }
00427 /*****
00428
00429 static portTickType prvSampleTimeNow( portBASE_TYPE *pxTimerListsWereSwitched )
00430 {
00431     portTickType xTimeNow;
00432     static portTickType xLastTime = ( portTickType ) OU;
00433
00434     xTimeNow = xTaskGetTickCount();
00435
00436     if( xTimeNow < xLastTime )
00437     {
00438         prvSwitchTimerLists( xLastTime );
00439         *pxTimerListsWereSwitched = pdTRUE;
00440     }
00441     else
00442     {
00443         *pxTimerListsWereSwitched = pdFALSE;
00444     }
00445
00446     xLastTime = xTimeNow;
00447
00448     return xTimeNow;
00449 }
00450 /*****
00451
00452 static portBASE_TYPE prvInsertTimerInActiveList( xTIMER *pxTimer, portTickType xNextExpiryTime,
00453                                                 portTickType xTimeNow, portTickType xCommandTime )
00454 {
00455     portBASE_TYPE xProcessTimerNow = pdFALSE;
00456
00457     listSET_LIST_ITEM_VALUE( &( pxTimer->xTimerListItem ), xNextExpiryTime );
00458     listSET_LIST_ITEM_OWNER( &( pxTimer->xTimerListItem ), pxTimer );
00459
00460     if( xNextExpiryTime <= xTimeNow )
00461     {
00462         /* Has the expiry time elapsed between the command to start/reset a
00463            timer was issued, and the time the command was processed? */
00464         if( ( ( portTickType ) ( xTimeNow - xCommandTime ) ) >= pxTimer->xTimerPeriodInTicks )
00465         {
00466             /* The time between a command being issued and the command being
00467                processed actually exceeds the timers period. */
00468             xProcessTimerNow = pdTRUE;
00469         }
00470         else
00471         {
00472             vListInsert( pxOverflowTimerList, &( pxTimer->xTimerListItem ) );
00473         }
00474     }
00475     else
00476     {
00477         if( ( xTimeNow < xCommandTime ) && ( xNextExpiryTime >= xCommandTime ) )
00478         {
00479             /* If, since the command was issued, the tick count has overflowed
00480                but the expiry time has not, then the timer must have already passed
00481                its expiry time and should be processed immediately. */
00482             xProcessTimerNow = pdTRUE;
00483         }
00484     }
00485     vListInsert( pxCurrentTimerList, &( pxTimer->xTimerListItem ) );
00486 }
00487
00488 return xProcessTimerNow;
00489 }
00490 /*****
00491
00492
00493 static void prvProcessReceivedCommands( void )
00494 {
00495     xTIMER_MESSAGE xMessage;
00496     xTIMER *pxTimer;
00497     portBASE_TYPE xTimerListsWereSwitched, xResult;
00498     portTickType xTimeNow;
00499
00500     /* In this case the xTimerListsWereSwitched parameter is not used, but it
00501        must be present in the function call. */
00502     xTimeNow = prvSampleTimeNow( &xTimerListsWereSwitched );

```

```

00503
00504     while( xQueueReceive( xTimerQueue, &xMessage, tmrNO_DELAY ) != pdFAIL )
00505     {
00506         pxTimer = xMessage.pxTimer;
00507
00508         /* Is the timer already in a list of active timers? When the command
00509         is trmCOMMAND_PROCESS_TIMER_OVERFLOW, the timer will be NULL as the
00510         command is to the task rather than to an individual timer. */
00511         if( pxTimer != NULL )
00512         {
00513             if( listIS_CONTAINED_WITHIN( NULL, &( pxTimer->xTimerListItem ) ) == pdFALSE )
00514             {
00515                 /* The timer is in a list, remove it. */
00516                 vListRemove( &( pxTimer->xTimerListItem ) );
00517             }
00518         }
00519
00520         traceTIMER_COMMAND RECEIVED( pxTimer, xMessage.xMessageID, xMessage.xMessageValue );
00521
00522         switch( xMessage.xMessageID )
00523         {
00524             case tmrCOMMAND_START :
00525                 /* Start or restart a timer. */
00526                 if( prvInsertTimerInActiveList( pxTimer, xMessage.xMessageValue +
pxTimer->xTimerPeriodInTicks, xTimeNow, xMessage.xMessageValue ) == pdTRUE )
00527                 {
00528                     /* The timer expired before it was added to the active timer
00529                     list. Process it now. */
00530                     pxTimer->pxCallbackFunction( ( xTimerHandle ) pxTimer );
00531
00532                     if( pxTimer->uxAutoReload == ( unsigned portBASE_TYPE ) pdTRUE )
00533                     {
00534                         xResult = xTimerGenericCommand( pxTimer, tmrCOMMAND_START,
xMessage.xMessageValue + pxTimer->xTimerPeriodInTicks, NULL, tmrNO_DELAY );
00535                         configASSERT( xResult );
00536                         ( void ) xResult;
00537                     }
00538                 }
00539                 break;
00540
00541             case tmrCOMMAND_STOP :
00542                 /* The timer has already been removed from the active list.
00543                 There is nothing to do here. */
00544                 break;
00545
00546             case tmrCOMMAND_CHANGE_PERIOD :
00547                 pxTimer->xTimerPeriodInTicks = xMessage.xMessageValue;
00548                 configASSERT( ( pxTimer->xTimerPeriodInTicks > 0 ) );
00549                 prvInsertTimerInActiveList( pxTimer, ( xTimeNow + pxTimer->xTimerPeriodInTicks ),
xTimeNow, xTimeNow );
00550                 break;
00551
00552             case tmrCOMMAND_DELETE :
00553                 /* The timer has already been removed from the active list,
00554                 just free up the memory. */
00555                 vPortFree( pxTimer );
00556                 break;
00557
00558             default :
00559                 /* Don't expect to get here. */
00560                 break;
00561         }
00562     }
00563 }
00564 /***** */
00565
00566 static void prvSwitchTimerLists( portTickType xLastTime )
00567 {
00568     portTickType xNextExpireTime, xReloadTime;
00569     xList *pxTemp;
00570     xTIMER *pxTimer;
00571     portBASE_TYPE xResult;
00572
00573     /* Remove compiler warnings if configASSERT() is not defined. */
00574     ( void ) xLastTime;
00575
00576     /* The tick count has overflowed. The timer lists must be switched.
00577     If there are any timers still referenced from the current timer list
00578     then they must have expired and should be processed before the lists
00579     are switched. */
00580     while( listLIST_IS_EMPTY( pxCurrentTimerList ) == pdFALSE )
00581     {
00582         xNextExpireTime = listGET_ITEM_VALUE_OF_HEAD_ENTRY( pxCurrentTimerList );
00583
00584         /* Remove the timer from the list. */
00585         pxTimer = ( xTIMER * ) listGET_OWNER_OF_HEAD_ENTRY( pxCurrentTimerList );
00586         vListRemove( &( pxTimer->xTimerListItem ) );

```

```

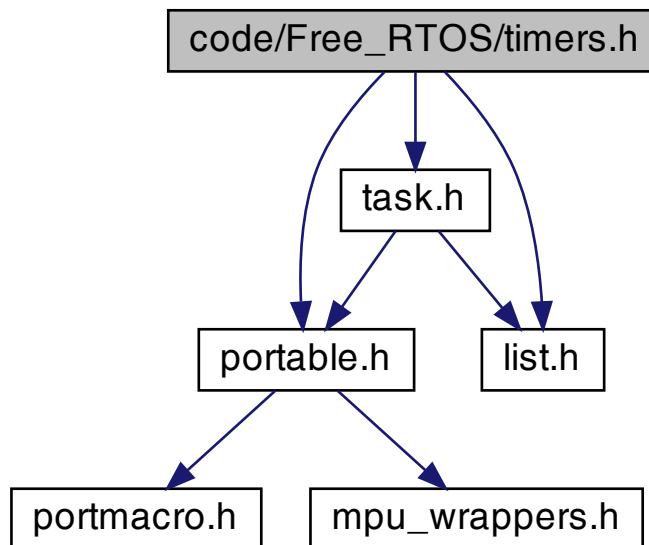
00587
00588     /* Execute its callback, then send a command to restart the timer if
00589     it is an auto-reload timer. It cannot be restarted here as the lists
00590     have not yet been switched. */
00591     pxTimer->pxCallbackFunction( ( xTimerHandle ) pxTimer );
00592
00593     if( pxTimer->uxAutoReload == ( unsigned portBASE_TYPE ) pdTRUE )
00594     {
00595         /* Calculate the reload value, and if the reload value results in
00596         the timer going into the same timer list then it has already expired
00597         and the timer should be re-inserted into the current list so it is
00598         processed again within this loop. Otherwise a command should be sent
00599         to restart the timer to ensure it is only inserted into a list after
00600         the lists have been swapped. */
00601         xReloadTime = ( xNextExpireTime + pxTimer->xTimerPeriodInTicks );
00602         if( xReloadTime > xNextExpireTime )
00603         {
00604             listSET_LIST_ITEM_VALUE( &( pxTimer->xTimerListItem ), xReloadTime );
00605             listSET_LIST_ITEM_OWNER( &( pxTimer->xTimerListItem ), pxTimer );
00606             vListInsert( pxCurrentTimerList, &( pxTimer->xTimerListItem ) );
00607         }
00608         else
00609         {
00610             xResult = xTimerGenericCommand( pxTimer, tmrCOMMAND_START, xNextExpireTime, NULL,
00611                                         tmrNO_DELAY );
00612             configASSERT( xResult );
00613             ( void ) xResult;
00614         }
00615     }
00616
00617     pxTemp = pxCurrentTimerList;
00618     pxCurrentTimerList = pxOverflowTimerList;
00619     pxOverflowTimerList = pxTemp;
00620 }
00621 /*****
00622
00623 static void prvCheckForValidListAndQueue( void )
00624 {
00625     /* Check that the list from which active timers are referenced, and the
00626     queue used to communicate with the timer service, have been
00627     initialised. */
00628     taskENTER_CRITICAL();
00629     {
00630         if( xTimerQueue == NULL )
00631         {
00632             vListInitialise( &xActiveTimerList1 );
00633             vListInitialise( &xActiveTimerList2 );
00634             pxCurrentTimerList = &xActiveTimerList1;
00635             pxOverflowTimerList = &xActiveTimerList2;
00636             xTimerQueue = xQueueCreate( ( unsigned portBASE_TYPE ) configTIMER_QUEUE_LENGTH, sizeof(
00637                 xTIMER_MESSAGE ) );
00638         }
00639         taskEXIT_CRITICAL();
00640     }
00641 /*****
00642
00643 portBASE_TYPE xTimerIsTimerActive( xTimerHandle xTimer )
00644 {
00645     portBASE_TYPE xTimerIsInActiveList;
00646     xTIMER *pxTimer = ( xTIMER * ) xTimer;
00647
00648     /* Is the timer in the list of active timers? */
00649     taskENTER_CRITICAL();
00650     {
00651         /* Checking to see if it is in the NULL list in effect checks to see if
00652         it is referenced from either the current or the overflow timer lists in
00653         one go, but the logic has to be reversed, hence the '!'. */
00654         xTimerIsInActiveList = !( listIS_CONTAINED_WITHIN( NULL, &( pxTimer->xTimerListItem ) ) );
00655     }
00656     taskEXIT_CRITICAL();
00657
00658     return xTimerIsInActiveList;
00659 }
00660 /*****
00661
00662 void *pvTimerGetTimerID( xTimerHandle xTimer )
00663 {
00664     xTIMER *pxTimer = ( xTIMER * ) xTimer;
00665
00666     return pxTimer->pvTimerID;
00667 }
00668 /*****
00669
00670 /* This entire source file will be skipped if the application is not configured
00671 to include software timer functionality. If you want to include software timer

```

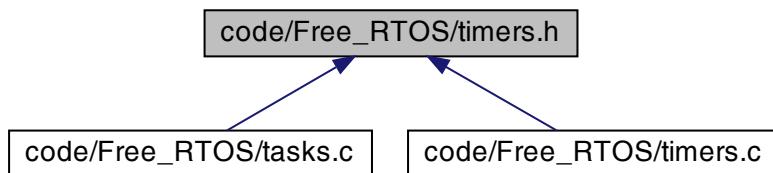
```
00672 functionality then ensure configUSE_TIMERS is set to 1 in FreeRTOSConfig.h. */
00673 #endif /* configUSE_TIMERS == 1 */
```

22.41 code/Free_RTOS/timers.h File Reference

```
#include "portable.h"
#include "list.h"
#include "task.h"
Include dependency graph for timers.h:
```



This graph shows which files directly or indirectly include this file:



Macros

- #define tmrCOMMAND_START 0
- #define tmrCOMMAND_STOP 1

- `#define tmrCOMMAND_CHANGE_PERIOD 2`
- `#define tmrCOMMAND_DELETE 3`
- `#define xTimerStart(xTimer, xBlockTime) xTimerGenericCommand((xTimer), tmrCOMMAND_START, (xTaskGetTickCount()), NULL, (xBlockTime))`
- `#define xTimerStop(xTimer, xBlockTime) xTimerGenericCommand((xTimer), tmrCOMMAND_STOP, 0U, NULL, (xBlockTime))`
- `#define xTimerChangePeriod(xTimer, xNewPeriod, xBlockTime) xTimerGenericCommand((xTimer), tmrCOMMAND_CHANGE_PERIOD, (xNewPeriod), NULL, (xBlockTime))`
- `#define xTimerDelete(xTimer, xBlockTime) xTimerGenericCommand((xTimer), tmrCOMMAND_DELETE, 0U, NULL, (xBlockTime))`
- `#define xTimerReset(xTimer, xBlockTime) xTimerGenericCommand((xTimer), tmrCOMMAND_START, (xTaskGetTickCount()), NULL, (xBlockTime))`
- `#define xTimerStartFromISR(xTimer, pxHigherPriorityTaskWoken) xTimerGenericCommand((xTimer), tmrCOMMAND_START, (xTaskGetTickCountFromISR()), (pxHigherPriorityTaskWoken), 0U)`
- `#define xTimerStopFromISR(xTimer, pxHigherPriorityTaskWoken) xTimerGenericCommand((xTimer), tmrCOMMAND_STOP, 0, (pxHigherPriorityTaskWoken), 0U)`
- `#define xTimerChangePeriodFromISR(xTimer, xNewPeriod, pxHigherPriorityTaskWoken) xTimerGenericCommand((xTimer), tmrCOMMAND_CHANGE_PERIOD, (xNewPeriod), (pxHigherPriorityTaskWoken), 0U)`
- `#define xTimerResetFromISR(xTimer, pxHigherPriorityTaskWoken) xTimerGenericCommand((xTimer), tmrCOMMAND_START, (xTaskGetTickCountFromISR()), (pxHigherPriorityTaskWoken), 0U)`

Typedefs

- `typedef void * xTimerHandle`
- `typedef void(* tmrTIMER_CALLBACK)(xTimerHandle xTimer)`

Functions

- `xTimerHandle xTimerCreate (const signed char *pcTimerName, portTickType xTimerPeriodInTicks, unsigned portBASE_TYPE uxAutoReload, void *pvTimerID, tmrTIMER_CALLBACK pxCallbackFunction) PRIVILEGED_FUNCTION`
- `void * pvTimerGetTimerID (xTimerHandle xTimer) PRIVILEGED_FUNCTION`
- `portBASE_TYPE xTimerIsTimerActive (xTimerHandle xTimer) PRIVILEGED_FUNCTION`
- `xTaskHandle xTimerGetTimerDaemonTaskHandle (void)`
- `portBASE_TYPE xTimerCreateTimerTask (void) PRIVILEGED_FUNCTION`
- `portBASE_TYPE xTimerGenericCommand (xTimerHandle xTimer, portBASE_TYPE xCommandID, portTickType xOptionalValue, portBASE_TYPE *pxHigherPriorityTaskWoken, portTickType xBlockTime) PRIVILEGED_FUNCTION`

22.41.1 Macro Definition Documentation

22.41.1.1 tmrCOMMAND_CHANGE_PERIOD `#define tmrCOMMAND_CHANGE_PERIOD 2`

Definition at line 75 of file [timers.h](#).

22.41.1.2 tmrCOMMAND_DELETE #define tmrCOMMAND_DELETE 3

Definition at line 76 of file [timers.h](#).

22.41.1.3 tmrCOMMAND_START #define tmrCOMMAND_START 0

Definition at line 73 of file [timers.h](#).

22.41.1.4 tmrCOMMAND_STOP #define tmrCOMMAND_STOP 1

Definition at line 74 of file [timers.h](#).

```
22.41.1.5 xTimerChangePeriod #define xTimerChangePeriod(  
    xTimer,  
    xNewPeriod,  
    xBlockTime ) xTimerGenericCommand( ( xTimer ), tmrCOMMAND_CHANGE_PERIOD, ( xNew←  
Period ), NULL, ( xBlockTime ) )
```

```
portBASE_TYPE xTimerChangePeriod( xTimerHandle xTimer, portTickType xNewPeriod, portTickType xBlockTime  
);
```

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

[xTimerChangePeriod\(\)](#) changes the period of a timer that was previously created using the [xTimerCreate\(\)](#) API function.

[xTimerChangePeriod\(\)](#) can be called to change the period of an active or dormant state timer.

The configUSE_TIMERS configuration constant must be set to 1 for [xTimerChangePeriod\(\)](#) to be available.

Parameters

<i>xTimer</i>	The handle of the timer that is having its period changed.
<i>xNewPeriod</i>	The new period for <i>xTimer</i> . Timer periods are specified in tick periods, so the constant portTICK_RATE_MS can be used to convert a time that has been specified in milliseconds. For example, if the timer must expire after 100 ticks, then <i>xNewPeriod</i> should be set to 100. Alternatively, if the timer must expire after 500ms, then <i>xNewPeriod</i> can be set to (500 / portTICK_RATE_MS) provided configTICK_RATE_HZ is less than or equal to 1000.
<i>xBlockTime</i>	Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the change period command to be successfully sent to the timer command queue, should the queue already be full when xTimerChangePeriod() was called. <i>xBlockTime</i> is ignored if xTimerChangePeriod() is called before the scheduler is started.

Returns

pdFAIL will be returned if the change period command could not be sent to the timer command queue even after xBlockTime ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Example usage:

```
// This function assumes xTimer has already been created. If the timer // referenced by xTimer is already active when it is called, then the timer // is deleted. If the timer referenced by xTimer is not active when it is // called, then the period of the timer is set to 500ms and the timer is // started. void vAFunction( xTimerHandle xTimer ) { if( xTimerIsTimerActive( xTimer ) != pdFALSE ) // or more simply and equivalently "if( xTimerIsTimerActive( xTimer ) )" { // xTimer is already active - delete it. xTimerDelete( xTimer ); } else { // xTimer is not active, change its period to 500ms. This will also // cause the timer to start. Block for a maximum of 100 ticks if the // change period command cannot immediately be sent to the timer // command queue. if( xTimerChangePeriod( xTimer, 500 / portTICK_← RATE_MS, 100 ) == pdPASS ) { // The command was successfully sent. } else { // The command could not be sent, even after waiting for 100 ticks // to pass. Take appropriate action here. } } }
```

Definition at line 461 of file [timers.h](#).

22.41.1.6 xTimerChangePeriodFromISR

```
#define xTimerChangePeriodFromISR( xTimer,
                                  xNewPeriod,
                                  pxHigherPriorityTaskWoken ) xTimerGenericCommand( ( xTimer ), tmrCOMMAND_CHANGE_PERIOD,
( xNewPeriod ), ( pxHigherPriorityTaskWoken ), 0U )
```

portBASE_TYPE xTimerChangePeriodFromISR(xTimerHandle xTimer, portTickType xNewPeriod, portBASE_← TYPE *pxHigherPriorityTaskWoken);

A version of [xTimerChangePeriod\(\)](#) that can be called from an interrupt service routine.

Parameters

<i>xTimer</i>	The handle of the timer that is having its period changed.
<i>xNewPeriod</i>	The new period for <i>xTimer</i> . Timer periods are specified in tick periods, so the constant portTICK_RATE_MS can be used to convert a time that has been specified in milliseconds. For example, if the timer must expire after 100 ticks, then <i>xNewPeriod</i> should be set to 100. Alternatively, if the timer must expire after 500ms, then <i>xNewPeriod</i> can be set to (500 / portTICK_RATE_MS) provided configTICK_RATE_HZ is less than or equal to 1000.
<i>pxHigherPriorityTaskWoken</i>	The timer service/daemon task spends most of its time in the Blocked state, waiting for messages to arrive on the timer command queue. Calling xTimerChangePeriodFromISR() writes a message to the timer command queue, so has the potential to transition the timer service/ daemon task out of the Blocked state. If calling xTimerChangePeriodFromISR() causes the timer service/daemon task to leave the Blocked state, and the timer service/daemon task has a priority equal to or greater than the currently executing task (the task that was interrupted), then <i>*pxHigherPriorityTaskWoken</i> will get set to pdTRUE internally within the xTimerChangePeriodFromISR() function. If xTimerChangePeriodFromISR() sets this value to pdTRUE then a context switch should be performed before the interrupt exits.

Returns

pdFAIL will be returned if the command to change the timers period could not be sent to the timer command queue. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Example usage:

```
// This scenario assumes xTimer has already been created and started. When // an interrupt occurs, the period of xTimer should be changed to 500ms.

// The interrupt service routine that changes the period of xTimer. void vAnExampleInterruptServiceRoutine( void )
{ portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;

    // The interrupt has occurred - change the period of xTimer to 500ms.
    // xHigherPriorityTaskWoken was set to pdFALSE where it was defined
    // (within this function). As this is an interrupt service routine, only
    // FreeRTOS API functions that end in "FromISR" can be used.
    if( xTimerChangePeriodFromISR( xTimer, &xHigherPriorityTaskWoken ) != pdPASS )
    {
        // The command to change the timers period was not executed
        // successfully. Take appropriate action here.
    }

    // If xHigherPriorityTaskWoken equals pdTRUE, then a context switch
    // should be performed. The syntax required to perform a context switch
    // from inside an ISR varies from port to port, and from compiler to
    // compiler. Inspect the demos for the port you are using to find the
    // actual syntax required.
    if( xHigherPriorityTaskWoken != pdFALSE )
    {
        // Call the interrupt safe yield function here (actual function
        // depends on the FreeRTOS port being used.
    }
}

}
```

Definition at line 840 of file [timers.h](#).

22.41.1.7 xTimerDelete #define xTimerDelete(

```
    xTimer,
    xBlockTime ) xTimerGenericCommand( ( xTimer ), tmrCOMMAND_DELETE, 0U, NULL, ( xleftrightarrow
BlockTime ) )
```

```
portBASE_TYPE xTimerDelete( xTimerHandle xTimer, portTickType xBlockTime );
```

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task though a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

[xTimerDelete\(\)](#) deletes a timer that was previously created using the [xTimerCreate\(\)](#) API function.

The configUSE_TIMERS configuration constant must be set to 1 for [xTimerDelete\(\)](#) to be available.

Parameters

<i>xTimer</i>	The handle of the timer being deleted.
<i>xBlockTime</i>	Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the delete command to be successfully sent to the timer command queue, should the queue already be full when xTimerDelete() was called. <i>xBlockTime</i> is ignored if xTimerDelete() is called before the scheduler is started.

Returns

`pdFAIL` will be returned if the delete command could not be sent to the timer command queue even after *xBlockTime* ticks had passed. `pdPASS` will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the `configTIMER_TASK_PRIORITY` configuration constant.

Example usage:

See the [xTimerChangePeriod\(\)](#) API function example usage scenario.

Definition at line 499 of file [timers.h](#).

22.41.1.8 xTimerReset `#define xTimerReset(`

```
xTimer,  
         xBlockTime ) xTimerGenericCommand\( \( xTimer \), tmrCOMMAND\_START, \( xTaskGetTickCount\(\) \)  
\), NULL, \( xBlockTime \) \)
```

```
portBASE_TYPE xTimerReset\( xTimerHandle xTimer, portTickType xBlockTime \);
```

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the `configTIMER_QUEUE_LENGTH` configuration constant.

[xTimerReset\(\)](#) re-starts a timer that was previously created using the [xTimerCreate\(\)](#) API function. If the timer had already been started and was already in the active state, then [xTimerReset\(\)](#) will cause the timer to re-evaluate its expiry time so that it is relative to when [xTimerReset\(\)](#) was called. If the timer was in the dormant state then [xTimerReset\(\)](#) has equivalent functionality to the [xTimerStart\(\)](#) API function.

Resetting a timer ensures the timer is in the active state. If the timer is not stopped, deleted, or reset in the mean time, the callback function associated with the timer will get called 'n' ticks after [xTimerReset\(\)](#) was called, where 'n' is the timers defined period.

It is valid to call [xTimerReset\(\)](#) before the scheduler has been started, but when this is done the timer will not actually start until the scheduler is started, and the timers expiry time will be relative to when the scheduler is started, not relative to when [xTimerReset\(\)](#) was called.

The configUSE_TIMERS configuration constant must be set to 1 for [xTimerReset\(\)](#) to be available.

Parameters

<i>xTimer</i>	The handle of the timer being reset/started/restarted.
<i>xBlockTime</i>	Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the reset command to be successfully sent to the timer command queue, should the queue already be full when xTimerReset() was called. <i>xBlockTime</i> is ignored if xTimerReset() is called before the scheduler is started.

Returns

pdFAIL will be returned if the reset command could not be sent to the timer command queue even after xBlockTime ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when [xTimerStart\(\)](#) is actually called. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Example usage:

```
// When a key is pressed, an LCD back-light is switched on. If 5 seconds pass // without a key being pressed, then the LCD back-light is switched off. In // this case, the timer is a one-shot timer.

xTimerHandle xBacklightTimer = NULL;

// The callback function assigned to the one-shot timer. In this case the // parameter is not used. void vBacklightTimerCallback( xTimerHandle pxTimer ) { // The timer expired, therefore 5 seconds must have passed since a key // was pressed. Switch off the LCD back-light. vSetBacklightState( BACKLIGHT_OFF ); }

// The key press event handler. void vKeyPressEventHandler( char cKey ) { // Ensure the LCD back-light is on, then reset the timer that is // responsible for turning the back-light off after 5 seconds of // key inactivity. Wait 10 ticks for the command to be successfully sent // if it cannot be sent immediately. vSetBacklightState( BACKLIGHT_ON ); if( xTimerReset( xBacklightTimer, 100 ) != pdPASS ) { // The reset command was not executed successfully. Take appropriate // action here. }

// Perform the rest of the key processing here. }

void main( void ) { long x;

// Create then start the one-shot timer that is responsible for turning
// the back-light off if no keys are pressed within a 5 second period.
xBacklightTimer = xTimerCreate( "BacklightTimer",           // Just a text name, not used by the kernel.
                               ( 5000 / portTICK_RATE_MS ), // The timer period in ticks.
                               pdFALSE,                  // The timer is a one-shot timer.
                               0,                        // The id is not used by the callback so can take
                               vBacklightTimerCallback   // The callback function that switches the LCD back-light
                           );

if( xBacklightTimer == NULL )
{
    // The timer was not created.
}
else
{
    // Start the timer. No block time is specified, and even if one was
    // it would be ignored because the scheduler has not yet been
    // started.
    if( xTimerStart( xBacklightTimer, 0 ) != pdPASS )
    {
        // The timer could not be set into the Active state.
    }
}

// ...
// Create tasks here.
// ...

// Starting the scheduler will start the timer running as it has already
// been set into the active state.
xTaskStartScheduler();

// Should not reach here.
for( ; ); }

}
```

Definition at line 622 of file [timers.h](#).

```
22.41.1.9 xTimerResetFromISR #define xTimerResetFromISR( xTimer,  
                                pxHigherPriorityTaskWoken ) xTimerGenericCommand( ( xTimer ), tmrCOMMAND_START,  
( xTaskGetTickCountFromISR() ), ( pxHigherPriorityTaskWoken ), 0U )
```

portBASE_TYPE xTimerResetFromISR(xTimerHandle xTimer, portBASE_TYPE *pxHigherPriorityTaskWoken);

A version of [xTimerReset\(\)](#) that can be called from an interrupt service routine.

Parameters

<i>xTimer</i>	The handle of the timer that is to be started, reset, or restarted.
<i>pxHigherPriorityTaskWoken</i>	The timer service/daemon task spends most of its time in the Blocked state, waiting for messages to arrive on the timer command queue. Calling xTimerResetFromISR() writes a message to the timer command queue, so has the potential to transition the timer service/daemon task out of the Blocked state. If calling xTimerResetFromISR() causes the timer service/daemon task to leave the Blocked state, and the timer service/ daemon task has a priority equal to or greater than the currently executing task (the task that was interrupted), then * <i>pxHigherPriorityTaskWoken</i> will get set to pdTRUE internally within the xTimerResetFromISR() function. If xTimerResetFromISR() sets this value to pdTRUE then a context switch should be performed before the interrupt exits.

Returns

pdFAIL will be returned if the reset command could not be sent to the timer command queue. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when [xTimerResetFromISR\(\)](#) is actually called. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Example usage:

```
// This scenario assumes xBacklightTimer has already been created. When a // key is pressed, an LCD back-light  
is switched on. If 5 seconds pass // without a key being pressed, then the LCD back-light is switched off. In // this  
case, the timer is a one-shot timer, and unlike the example given for // the xTimerReset\(\) function, the key press  
event handler is an interrupt // service routine.  
  
// The callback function assigned to the one-shot timer. In this case the // parameter is not used. void vBacklight←  
TimerCallback( xTimerHandle pxTimer ) { // The timer expired, therefore 5 seconds must have passed since a key  
// was pressed. Switch off the LCD back-light. vSetBacklightState( BACKLIGHT_OFF ); }  
  
// The key press interrupt service routine. void vKeyPressEventInterruptHandler( void ) { portBASE_TYPE xHigher←  
PriorityTaskWoken = pdFALSE;  
  
// Ensure the LCD back-light is on, then reset the timer that is  
// responsible for turning the back-light off after 5 seconds of  
// key inactivity. This is an interrupt service routine so can only  
// call FreeRTOS API functions that end in "FromISR".  
vSetBacklightState( BACKLIGHT_ON );  
  
// xTimerStartFromISR() or xTimerResetFromISR() could be called here  
// as both cause the timer to re-calculate its expiry time.  
// xHigherPriorityTaskWoken was initialised to pdFALSE when it was  
// declared (in this function).  
if( xTimerResetFromISR( xBacklightTimer, &xHigherPriorityTaskWoken ) != pdPASS )  
{  
    // The reset command was not executed successfully. Take appropriate  
    // action here.
```

```

}

// Perform the rest of the key processing here.

// If xHigherPriorityTaskWoken equals pdTRUE, then a context switch
// should be performed. The syntax required to perform a context switch
// from inside an ISR varies from port to port, and from compiler to
// compiler. Inspect the demos for the port you are using to find the
// actual syntax required.
if( xHigherPriorityTaskWoken != pdFALSE )
{
    // Call the interrupt safe yield function here (actual function
    // depends on the FreeRTOS port being used.
}

}

```

Definition at line 925 of file [timers.h](#).

22.41.1.10 xTimerStart

```
#define xTimerStart(
    xTimer,
    xBlockTime )  xTimerGenericCommand( ( xTimer ), tmrCOMMAND_START, ( xTaskGetTickCount() )
), NULL, ( xBlockTime ) )
```

portBASE_TYPE xTimerStart(xTimerHandle xTimer, portTickType xBlockTime);

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

[xTimerStart\(\)](#) starts a timer that was previously created using the [xTimerCreate\(\)](#) API function. If the timer had already been started and was already in the active state, then [xTimerStart\(\)](#) has equivalent functionality to the [xTimerReset\(\)](#) API function.

Starting a timer ensures the timer is in the active state. If the timer is not stopped, deleted, or reset in the meantime, the callback function associated with the timer will get called 'n' ticks after [xTimerStart\(\)](#) was called, where 'n' is the timers defined period.

It is valid to call [xTimerStart\(\)](#) before the scheduler has been started, but when this is done the timer will not actually start until the scheduler is started, and the timers expiry time will be relative to when the scheduler is started, not relative to when [xTimerStart\(\)](#) was called.

The configUSE_TIMERS configuration constant must be set to 1 for [xTimerStart\(\)](#) to be available.

Parameters

<i>xTimer</i>	The handle of the timer being started/restarted.
<i>xBlockTime</i>	Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the start command to be successfully sent to the timer command queue, should the queue already be full when xTimerStart() was called. <i>xBlockTime</i> is ignored if xTimerStart() is called before the scheduler is started.

Returns

pdFAIL will be returned if the start command could not be sent to the timer command queue even after `xBlockTime` ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when `xTimerStart()` is actually called. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Example usage:

See the `xTimerCreate()` API function example usage scenario.

Definition at line 340 of file `timers.h`.

22.41.1.11 xTimerStartFromISR #define `xTimerStartFromISR(`

```
    xTimer,
    pxHigherPriorityTaskWoken ) xTimerGenericCommand( ( xTimer ), tmrCOMMAND_START,
( xTaskGetTickCountFromISR() ), ( pxHigherPriorityTaskWoken ), 0U )
```

`portBASE_TYPE xTimerStartFromISR(xTimerHandle xTimer, portBASE_TYPE *pxHigherPriorityTaskWoken);`

A version of `xTimerStart()` that can be called from an interrupt service routine.

Parameters

<code>xTimer</code>	The handle of the timer being started/restarted.
<code>pxHigherPriorityTaskWoken</code>	The timer service/daemon task spends most of its time in the Blocked state, waiting for messages to arrive on the timer command queue. Calling <code>xTimerStartFromISR()</code> writes a message to the timer command queue, so has the potential to transition the timer service/daemon task out of the Blocked state. If calling <code>xTimerStartFromISR()</code> causes the timer service/daemon task to leave the Blocked state, and the timer service/daemon task has a priority equal to or greater than the currently executing task (the task that was interrupted), then <code>*pxHigherPriorityTaskWoken</code> will get set to pdTRUE internally within the <code>xTimerStartFromISR()</code> function. If <code>xTimerStartFromISR()</code> sets this value to pdTRUE then a context switch should be performed before the interrupt exits.

Returns

pdFAIL will be returned if the start command could not be sent to the timer command queue. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when `xTimerStartFromISR()` is actually called. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Example usage:

```
// This scenario assumes xBacklightTimer has already been created. When a // key is pressed, an LCD back-light is switched on. If 5 seconds pass // without a key being pressed, then the LCD back-light is switched off. In // this case, the timer is a one-shot timer, and unlike the example given for // the xTimerReset() function, the key press event handler is an interrupt // service routine.
```

```
// The callback function assigned to the one-shot timer. In this case the // parameter is not used. void vBacklightCallback( xTimerHandle pxTimer ) { // The timer expired, therefore 5 seconds must have passed since a key was pressed. Switch off the LCD back-light. vSetBacklightState( BACKLIGHT_OFF ); }

// The key press interrupt service routine. void vKeyPressEventInterruptHandler( void ) { portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;

// Ensure the LCD back-light is on, then restart the timer that is responsible for turning the back-light off after 5 seconds of key inactivity. This is an interrupt service routine so can only call FreeRTOS API functions that end in "FromISR".
vSetBacklightState( BACKLIGHT_ON );

// xTimerStartFromISR() or xTimerResetFromISR() could be called here as both cause the timer to re-calculate its expiry time.
// xHigherPriorityTaskWoken was initialised to pdFALSE when it was declared (in this function).
if( xTimerStartFromISR( xBacklightTimer, &xHigherPriorityTaskWoken ) != pdPASS )
{
    // The start command was not executed successfully. Take appropriate action here.
}

// Perform the rest of the key processing here.

// If xHigherPriorityTaskWoken equals pdTRUE, then a context switch should be performed. The syntax required to perform a context switch from inside an ISR varies from port to port, and from compiler to compiler. Inspect the demos for the port you are using to find the actual syntax required.
if( xHigherPriorityTaskWoken != pdFALSE )
{
    // Call the interrupt safe yield function here (actual function depends on the FreeRTOS port being used.
}

}
```

Definition at line 706 of file [timers.h](#).

```
22.41.1.12 xTimerStop #define xTimerStop(
    xTimer,
    xBlockTime ) xTimerGenericCommand( ( xTimer ), tmrCOMMAND_STOP, 0U, NULL, ( xBlockTime ) )
```

```
portBASE_TYPE xTimerStop( xTimerHandle xTimer, portTickType xBlockTime );
```

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

[xTimerStop\(\)](#) stops a timer that was previously started using either of the [The xTimerStart\(\)](#), [xTimerReset\(\)](#), [xTimerStartFromISR\(\)](#), [xTimerResetFromISR\(\)](#), [xTimerChangePeriod\(\)](#) or [xTimerChangePeriodFromISR\(\)](#) API functions.

Stopping a timer ensures the timer is not in the active state.

The configUSE_TIMERS configuration constant must be set to 1 for [xTimerStop\(\)](#) to be available.

Parameters

<i>xTimer</i>	The handle of the timer being stopped.
<i>xBlockTime</i>	Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the stop command to be successfully sent to the timer command queue, should the queue already be full when xTimerStop() was called. <i>xBlockTime</i> is ignored if xTimerStop() is called before the scheduler is started.

Returns

`pdFAIL` will be returned if the stop command could not be sent to the timer command queue even after *xBlockTime* ticks had passed. `pdPASS` will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the `configTIMER_TASK_PRIORITY` configuration constant.

Example usage:

See the [xTimerCreate\(\)](#) API function example usage scenario.

Definition at line 382 of file [timers.h](#).

22.41.1.13 xTimerStopFromISR #define xTimerStopFromISR(

```
    xTimer,
    pxHigherPriorityTaskWoken ) xTimerGenericCommand( ( xTimer ), tmrCOMMAND_STOP,
0, ( pxHigherPriorityTaskWoken ), 0U )
```

`portBASE_TYPE xTimerStopFromISR(xTimerHandle xTimer, portBASE_TYPE *pxHigherPriorityTaskWoken);`

A version of [xTimerStop\(\)](#) that can be called from an interrupt service routine.

Parameters

<i>xTimer</i>	The handle of the timer being stopped.
<i>pxHigherPriorityTaskWoken</i>	The timer service/daemon task spends most of its time in the Blocked state, waiting for messages to arrive on the timer command queue. Calling xTimerStopFromISR() writes a message to the timer command queue, so has the potential to transition the timer service/daemon task out of the Blocked state. If calling xTimerStopFromISR() causes the timer service/daemon task to leave the Blocked state, and the timer service/daemon task has a priority equal to or greater than the currently executing task (the task that was interrupted), then <i>*pxHigherPriorityTaskWoken</i> will get set to <code>pdTRUE</code> internally within the xTimerStopFromISR() function. If xTimerStopFromISR() sets this value to <code>pdTRUE</code> then a context switch should be performed before the interrupt exits.

Returns

`pdFAIL` will be returned if the stop command could not be sent to the timer command queue. `pdPASS` will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the `configTIMER_TASK_PRIORITY` configuration constant.

Example usage:

```
// This scenario assumes xTimer has already been created and started. When // an interrupt occurs, the timer  
// should be simply stopped.  
// The interrupt service routine that stops the timer. void vAnExampleInterruptServiceRoutine( void ) { portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;  
  
    // The interrupt has occurred - simply stop the timer.  
    // xHigherPriorityTaskWoken was set to pdFALSE where it was defined  
    // (within this function). As this is an interrupt service routine, only  
    // FreeRTOS API functions that end in "FromISR" can be used.  
    if( xTimerStopFromISR( xTimer, &xHigherPriorityTaskWoken ) != pdPASS )  
    {  
        // The stop command was not executed successfully. Take appropriate  
        // action here.  
    }  
  
    // If xHigherPriorityTaskWoken equals pdTRUE, then a context switch  
    // should be performed. The syntax required to perform a context switch  
    // from inside an ISR varies from port to port, and from compiler to  
    // compiler. Inspect the demos for the port you are using to find the  
    // actual syntax required.  
    if( xHigherPriorityTaskWoken != pdFALSE )  
    {  
        // Call the interrupt safe yield function here (actual function  
        // depends on the FreeRTOS port being used.  
    }  
}
```

Definition at line 768 of file [timers.h](#).

22.41.2 Typedef Documentation

22.41.2.1 tmrTIMER_CALLBACK `typedef void(* tmrTIMER_CALLBACK)(xTimerHandle xTimer)`

Definition at line 91 of file [timers.h](#).

22.41.2.2 xTimerHandle `typedef void* xTimerHandle`

Type by which software timers are referenced. For example, a call to [xTimerCreate\(\)](#) returns an xTimerHandle variable that can then be used to reference the subject timer in calls to other software timer API functions (for example, [xTimerStart\(\)](#), [xTimerReset\(\)](#), etc.).

Definition at line 88 of file [timers.h](#).

22.41.3 Function Documentation

22.41.3.1 pvTimerGetTimerID() `void* pvTimerGetTimerID(xTimerHandle xTimer)`

`void *pvTimerGetTimerID(xTimerHandle xTimer);`

Returns the ID assigned to the timer.

IDs are assigned to timers using the pvTimerID parameter of the call to [xTimerCreated\(\)](#) that was used to create the timer.

If the same callback function is assigned to multiple timers then the timer ID can be used within the callback function to identify which timer actually expired.

Parameters

<i>xTimer</i>	The timer being queried.
---------------	--------------------------

Returns

The ID assigned to the timer being queried.

Example usage:

See the [xTimerCreate\(\)](#) API function example usage scenario.

22.41.3.2 xTimerCreate() *xTimerHandle xTimerCreate(*

```
    const signed char * pcTimerName,
    portTickType xTimerPeriodInTicks,
    unsigned portBASE_TYPE uxAutoReload,
    void * pvTimerID,
    tmrTIMER_CALLBACK pxCallbackFunction )
```

```
xTimerHandle xTimerCreate( const signed char *pcTimerName, portTickType xTimerPeriod, unsigned portBASE←
←_TYPE uxAutoReload, void * pvTimerID, tmrTIMER_CALLBACK pxCallbackFunction );
```

Creates a new software timer instance. This allocates the storage required by the new timer, initialises the new timers internal state, and returns a handle by which the new timer can be referenced.

Timers are created in the dormant state. The [xTimerStart\(\)](#), [xTimerReset\(\)](#), [xTimerStartFromISR\(\)](#), [xTimerResetFromISR\(\)](#), [xTimerChangePeriod\(\)](#) and [xTimerChangePeriodFromISR\(\)](#) API functions can all be used to transition a timer into the active state.

Parameters

<i>pcTimerName</i>	A text name that is assigned to the timer. This is done purely to assist debugging. The kernel itself only ever references a timer by its handle, and never by its name.
<i>xTimerPeriod</i>	The timer period. The time is defined in tick periods so the constant portTICK_RATE_MS can be used to convert a time that has been specified in milliseconds. For example, if the timer must expire after 100 ticks, then xTimerPeriod should be set to 100. Alternatively, if the timer must expire after 500ms, then xPeriod can be set to (500 / portTICK_RATE_MS) provided configTICK_RATE_HZ is less than or equal to 1000.
<i>uxAutoReload</i>	If uxAutoReload is set to pdTRUE then the timer will expire repeatedly with a frequency set by the xTimerPeriod parameter. If uxAutoReload is set to pdFALSE then the timer will be a one-shot timer and enter the dormant state after it expires.
<i>pvTimerID</i>	An identifier that is assigned to the timer being created. Typically this would be used in the timer callback function to identify which timer expired when the same callback function is assigned to more than one timer.
<i>pxCallbackFunction</i>	The function to call when the timer expires. Callback functions must have the prototype defined by tmrTIMER_CALLBACK, which is "void vCallbackFunction(xTimerHandle xTimer);".

Returns

If the timer is successfully created then a handle to the newly created timer is returned. If the timer cannot be created (because either there is insufficient FreeRTOS heap remaining to allocate the timer structures, or the timer period was set to 0) then 0 is returned.

Example usage:

```
#define NUM_TIMERS 5

// An array to hold handles to the created timers. xTimerHandle xTimers[ NUM_TIMERS ];

// An array to hold a count of the number of times each timer expires. long lExpireCounters[ NUM_TIMERS ] = { 0 };

// Define a callback function that will be used by multiple timer instances. // The callback function does nothing but
// count the number of times the // associated timer expires, and stop the timer once the timer has expired // 10 times.
void vTimerCallback( xTimerHandle pxTimer ) { long lArrayIndex; const long xMaxExpiryCountBeforeStopping = 10;

    // Optionally do something if the pxTimer parameter is NULL.
    configASSERT( pxTimer );

    // Which timer expired?
    lArrayIndex = ( long ) pvTimerGetTimerID( pxTimer );

    // Increment the number of times that pxTimer has expired.
    lExpireCounters[ lArrayIndex ] += 1;

    // If the timer has expired 10 times then stop it from running.
    if( lExpireCounters[ lArrayIndex ] == xMaxExpiryCountBeforeStopping )
    {
        // Do not use a block time if calling a timer API function from a
        // timer callback function, as doing so could cause a deadlock!
        xTimerStop( pxTimer, 0 );
    }
}

void main( void ) { long x;

    // Create then start some timers. Starting the timers before the scheduler
    // has been started means the timers will start running immediately that
    // the scheduler starts.
    for( x = 0; x < NUM_TIMERS; x++ )
    {
        xTimers[ x ] = xTimerCreate(      "Timer",           // Just a text name, not used by the kernel.
                                         ( 100 * x ),       // The timer period in ticks.
                                         pdTRUE,            // The timers will auto-reload themselves when they expire
                                         ( void * ) x,       // Assign each timer a unique id equal to its array index
                                         vTimerCallback      // Each timer calls the same callback when it expires.
                                         );
        if( xTimers[ x ] == NULL )
        {
            // The timer was not created.
        }
        else
        {
            // Start the timer. No block time is specified, and even if one was
            // it would be ignored because the scheduler has not yet been
            // started.
            if( xTimerStart( xTimers[ x ], 0 ) != pdPASS )
            {
                // The timer could not be set into the Active state.
            }
        }
    }

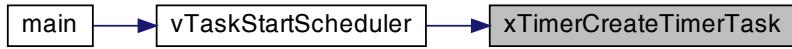
    // ...
    // Create tasks here.
    // ...

    // Starting the scheduler will start the timers running as they have already
    // been set into the active state.
    xTaskStartScheduler();

    // Should not reach here.
    for( ;; );
}
```

```
22.41.3.3 xTimerCreateTimerTask() portBASE_TYPE xTimerCreateTimerTask (
    void )
```

Here is the caller graph for this function:



```
22.41.3.4 xTimerGenericCommand() portBASE_TYPE xTimerGenericCommand (
    xTimerHandle xTimer,
    portBASE_TYPE xCommandID,
    portTickType xOptionalValue,
    portBASE_TYPE * pxHigherPriorityTaskWoken,
    portTickType xBlockTime )
```

```
22.41.3.5 xTimerGetTimerDaemonTaskHandle() xTaskHandle xTimerGetTimerDaemonTaskHandle (
    void )
```

`xTimerGetTimerDaemonTaskHandle()` is only available if `INCLUDE_xTimerGetTimerDaemonTaskHandle` is set to 1 in `FreeRTOSConfig.h`.

Simply returns the handle of the timer service/daemon task. It is not valid to call `xTimerGetTimerDaemonTaskHandle()` before the scheduler has been started.

```
22.41.3.6 xTimerIsTimerActive() portBASE_TYPE xTimerIsTimerActive (
    xTimerHandle xTimer )
```

```
portBASE_TYPE xTimerIsTimerActive( xTimerHandle xTimer );
```

Queries a timer to see if it is active or dormant.

A timer will be dormant if: 1) It has been created but not started, or 2) It is an expired on-shot timer that has not been restarted.

Timers are created in the dormant state. The `xTimerStart()`, `xTimerReset()`, `xTimerStartFromISR()`, `xTimerResetFromISR()`, `xTimerChangePeriod()` and `xTimerChangePeriodFromISR()` API functions can all be used to transition a timer into the active state.

Parameters

<code>xTimer</code>	The timer being queried.
---------------------	--------------------------

Returns

pdFALSE will be returned if the timer is dormant. A value other than pdFALSE will be returned if the timer is active.

Example usage:

```
// This function assumes xTimer has already been created. void vAFunction( xTimerHandle xTimer ) { if( xTimerIsTimerActive( xTimer ) != pdFALSE ) // or more simply and equivalently "if( xTimerIsTimerActive( xTimer ) )" { // xTimer is active, do something. } else { // xTimer is not active, do something else. } }
```

22.42 timers.h

```
00001 /*
00002   FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
00003
00004
00005 ****
00006 *
00007 *   FreeRTOS tutorial books are available in pdf and paperback.
00008 *   Complete, revised, and edited pdf reference manuals are also
00009 *   available.
00010 *
00011 *   Purchasing FreeRTOS documentation will not only help you, by
00012 *   ensuring you get running as quickly as possible and with an
00013 *   in-depth knowledge of how to use FreeRTOS, it will also help
00014 *   the FreeRTOS project to continue with its mission of providing
00015 *   professional grade, cross platform, de facto standard solutions
00016 *   for microcontrollers - completely free of charge!
00017 *
00018 *   >> See http://www.FreeRTOS.org/Documentation for details. <<
00019 *
00020 *   Thank you for using FreeRTOS, and thank you for your support!
00021 *
00022 ****
00023
00024
00025 This file is part of the FreeRTOS distribution.
00026
00027 FreeRTOS is free software; you can redistribute it and/or modify it under
00028 the terms of the GNU General Public License (version 2) as published by the
00029 Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
00030 >>NOTE<< The modification to the GPL is included to allow you to
00031 distribute a combined work that includes FreeRTOS without being obliged to
00032 provide the source code for proprietary components outside of the FreeRTOS
00033 kernel. FreeRTOS is distributed in the hope that it will be useful, but
00034 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
00035 or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
00036 more details. You should have received a copy of the GNU General Public
00037 License and the FreeRTOS license exception along with FreeRTOS; if not it
00038 can be viewed here: http://www.freertos.org/a00114.html and also obtained
00039 by writing to Richard Barry, contact details for whom are available on the
00040 FreeRTOS WEB site.
00041
00042 1 tab == 4 spaces!
00043
00044 http://www.FreeRTOS.org - Documentation, latest information, license and
00045 contact details.
00046
00047 http://www.SafeRTOS.com - A version that is certified for use in safety
00048 critical systems.
00049
00050 http://www.OpenRTOS.com - Commercial support, development, porting,
00051 licensing and training services.
00052 */
00053
00054
00055 #ifndef TIMERS_H
00056 #define TIMERS_H
00057
00058 #ifndef INC_FREERTOS_H
00059     #error "include FreeRTOS.h must appear in source files before include timers.h"
00060 #endif
00061
00062 #include "portable.h"
00063 #include "list.h"
00064 #include "task.h"
00065
00066 #ifdef __cplusplus
00067 extern "C" {
```

```

00068 #endif
00069
00070 /* IDs for commands that can be sent/received on the timer queue. These are to
00071 be used solely through the macros that make up the public software timer API,
00072 as defined below. */
00073 #define tmrCOMMAND_START 0
00074 #define tmrCOMMAND_STOP 1
00075 #define tmrCOMMAND_CHANGE_PERIOD 2
00076 #define tmrCOMMAND_DELETE 3
00077
00078 /-----
00079 * MACROS AND DEFINITIONS
00080 -----*/
00081
00082 typedef void * xTimerHandle;
00083
00084 /* Define the prototype to which timer callback functions must conform. */
00085 typedef void (*tmrTIMER_CALLBACK)( xTimerHandle xTimer );
00086
00087 xTimerHandle xTimerCreate( const signed char *pcTimerName, portTickType xTimerPeriodInTicks, unsigned
00088 portBASE_TYPE uxAutoReload, void * pvTimerID, tmrTIMER_CALLBACK pxCallbackFunction )
00089 PRIVILEGED_FUNCTION;
00090
00091 void *pvTimerGetTimerID( xTimerHandle xTimer ) PRIVILEGED_FUNCTION;
00092
00093 portBASE_TYPE xTimerIsTimerActive( xTimerHandle xTimer ) PRIVILEGED_FUNCTION;
00094
00095 xTaskHandle xTimerGetTimerDaemonTaskHandle( void );
00096
00097 #define xTimerStart( xTimer, xBlockTime ) xTimerGenericCommand( ( xTimer ), tmrCOMMAND_START, (
00098 xTaskGetTickCount() ), NUL, ( xBlockTime ) )
00099
00100 #define xTimerStop( xTimer, xBlockTime ) xTimerGenericCommand( ( xTimer ), tmrCOMMAND_STOP, 0U, NULL,
00101 ( xBlockTime ) )
00102
00103 #define xTimerChangePeriod( xTimer, xNewPeriod, xBlockTime ) xTimerGenericCommand( ( xTimer ),
00104 tmrCOMMAND_CHANGE_PERIOD, ( xNewPeriod ), NULL, ( xBlockTime ) )
00105
00106 #define xTimerDelete( xTimer, xBlockTime ) xTimerGenericCommand( ( xTimer ), tmrCOMMAND_DELETE, 0U,
00107 NULL, ( xBlockTime ) )
00108
00109 #define xTimerReset( xTimer, xBlockTime ) xTimerGenericCommand( ( xTimer ), tmrCOMMAND_START, (
00110 xTaskGetTickCount() ), NUL, ( xBlockTime ) )
00111
00112 #define xTimerStartFromISR( xTimer, pxHigherPriorityTaskWoken ) xTimerGenericCommand( ( xTimer ),
00113 tmrCOMMAND_START, ( xTaskGetTickCountFromISR() ), ( pxHigherPriorityTaskWoken ), 0U )
00114
00115 #define xTimerStopFromISR( xTimer, pxHigherPriorityTaskWoken ) xTimerGenericCommand( ( xTimer ),
00116 tmrCOMMAND_STOP, 0, ( pxHigherPriorityTaskWoken ), 0U )
00117
00118 #define xTimerChangePeriodFromISR( xTimer, xNewPeriod, pxHigherPriorityTaskWoken )
00119 xTimerGenericCommand( ( xTimer ), tmrCOMMAND_CHANGE_PERIOD, ( xNewPeriod ), (
00120 pxHigherPriorityTaskWoken ), 0U )
00121
00122 #define xTimerResetFromISR( xTimer, pxHigherPriorityTaskWoken ) xTimerGenericCommand( ( xTimer ),
00123 tmrCOMMAND_START, ( xTaskGetTickCountFromISR() ), ( pxHigherPriorityTaskWoken ), 0U )
00124
00125
00126 /*
00127 * Functions beyond this part are not part of the public API and are intended
00128 * for use by the kernel only.
00129 */
00130
00131 portBASE_TYPE xTimerCreateTimerTask( void ) PRIVILEGED_FUNCTION;
00132 portBASE_TYPE xTimerGenericCommand( xTimerHandle xTimer, portBASE_TYPE xCommandID, portTickType
00133 xOptionalValue, portBASE_TYPE *pxHigherPriorityTaskWoken, portTickType xBlockTime )
00134 PRIVILEGED_FUNCTION;
00135
00136 #ifdef __cplusplus
00137 #endif
00138 #endif /* TIMERS_H */
00139
00140

```

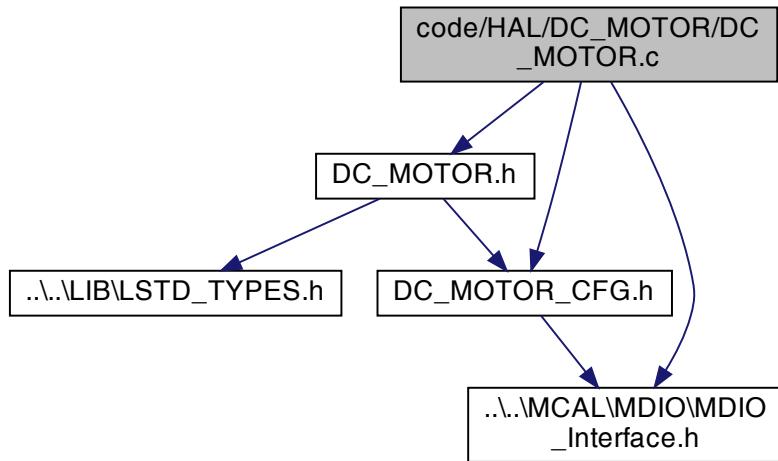
22.43 code/HAL/DC_MOTOR/DC_MOTOR.c File Reference

```

#include "DC_MOTOR.h"
#include "DC_MOTOR_CFG.h"
#include "../../MCAL/MDIO/MDIO_Interface.h"

```

Include dependency graph for DC_MOTOR.c:



Functions

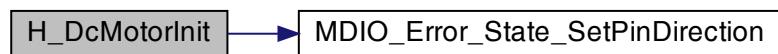
- void [H_DcMotorInit](#) (void)
- void [H_DcMotorStart](#) ([u8](#) u8_local_direction)
- void [H_DcMotorStop](#) (void)

22.43.1 Function Documentation

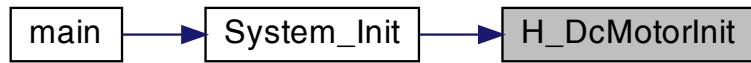
22.43.1.1 H_DcMotorInit() `void H_DcMotorInit (`
`void)`

Definition at line 13 of file [DC_MOTOR.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



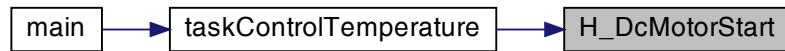
22.43.1.2 H_DcMotorStart() `void H_DcMotorStart (u8 u8_local_direction)`

Definition at line 20 of file [DC_MOTOR.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.43.1.3 H_DcMotorStop() `void H_DcMotorStop (void)`

Definition at line 37 of file [DC_MOTOR.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.44 DC_MOTOR.c

```

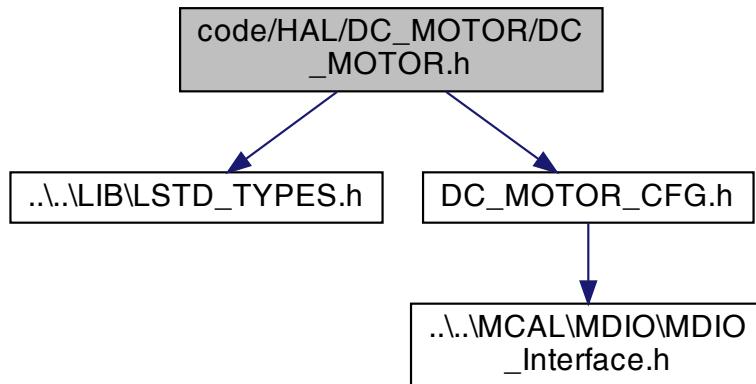
00001 /*
00002 * DC_MOTOR.c
00003 *
00004 * Created: 4/18/2022 9:46:43 AM
00005 * Author: Dell
00006 */
00007
00008 #include "DC_MOTOR.h"
00009 #include "DC_MOTOR_CFG.h"
00010 //##include "../../../MCAL/TIMER_0/TIMER_0.h"
00011 #include "../../../MCAL/MDIO/MDIO_Interface.h"
00012
00013 void H_DcMotorInit(void)
00014 {
00015     MDIO_Error_State_SetPinDirection(IN_1, DC_M_PORT, PIN_OUTPUT);
00016     //MDIO_Error_State_SetPinDirection(IN_2, DC_M_PORT, PIN_OUTPUT);
00017     //M_Pwm0Init();
00018 }
00019
00020 void H_DcMotorStart(u8 u8_local_direction)
00021 {
00022     //switch(u8_local_direction)
00023     //{
00024         //case CLK_W:
00025             MDIO_Error_State_SetPinValue(IN_1, DC_M_PORT, PIN_HIGH);
00026             //MDIO_Error_State_SetPinValue(IN_2, DC_M_PORT, PIN_LOW);
00027         //break;
00028         //case A_CLK_W:
00029             //MDIO_Error_State_SetPinValue(IN_1, DC_M_PORT, PIN_LOW);
00030             //MDIO_Error_State_SetPinValue(IN_2, DC_M_PORT, PIN_HIGH);
00031         //break;
00032         //default:
00033         //break;
00034     //}
00035 }
00036
00037 void H_DcMotorStop(void)
00038 {
00039     MDIO_Error_State_SetPinValue(IN_1, DC_M_PORT, PIN_LOW);
00040     //MDIO_Error_State_SetPinValue(IN_2, DC_M_PORT, PIN_LOW);
00041 }
  
```

22.45 code/HAL/DC_MOTOR/DC_MOTOR.h File Reference

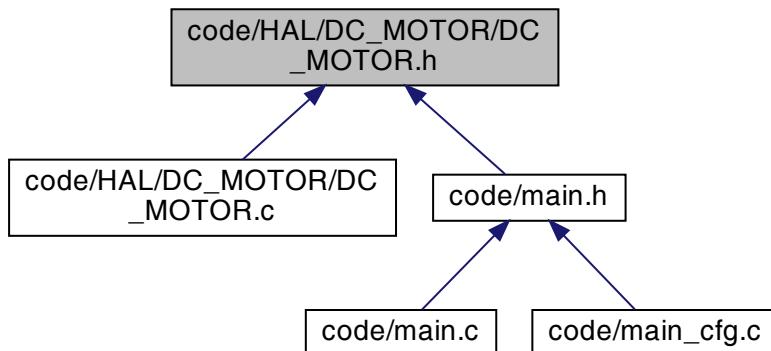
```

#include "..\..\LIB\LSTD_TYPES.h"
#include "DC_MOTOR_CFG.h"
  
```

Include dependency graph for DC_MOTOR.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define CLK_W 1`
- `#define A_CLK_W 2`

Functions

- `void H_DcMotorInit (void)`
- `void H_DcMotorSetDirection (u8)`
- `void H_DcMotorSetSpeed (u32)`
- `void H_DcMotorStart (u8)`
- `void H_DcMotorStop (void)`

22.45.1 Macro Definition Documentation

22.45.1.1 A_CLK_W #define A_CLK_W 2

Definition at line 22 of file [DC_MOTOR.h](#).

22.45.1.2 CLK_W #define CLK_W 1

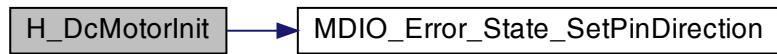
Definition at line 21 of file [DC_MOTOR.h](#).

22.45.2 Function Documentation

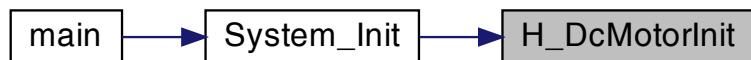
22.45.2.1 H_DcMotorInit() void H_DcMotorInit (void)

Definition at line 13 of file [DC_MOTOR.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.45.2.2 H_DcMotorSetDirection() void H_DcMotorSetDirection (u8)

22.45.2.3 H_DcMotorSetSpeed() void H_DcMotorSetSpeed (
 32)

22.45.2.4 H_DcMotorStart() void H_DcMotorStart (
 8 u8_local_direction)

Definition at line 20 of file [DC_MOTOR.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



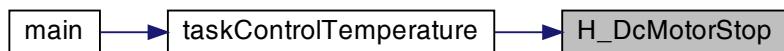
22.45.2.5 H_DcMotorStop() void H_DcMotorStop (
 void)

Definition at line 37 of file [DC_MOTOR.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.46 DC_MOTOR.h

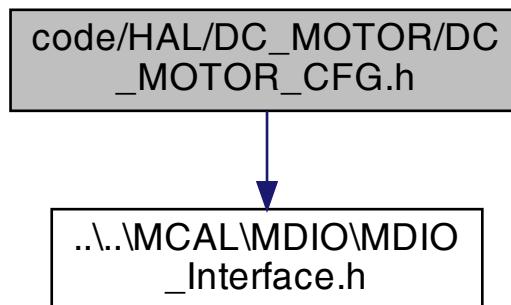
```

00001 /*
00002  * DC_MOTOR.h
00003 *
00004  * Created: 4/18/2022 9:46:58 AM
00005  * Author: Dell
00006 */
00007
00008
00009 #ifndef DC_MOTOR_H_
00010 #define DC_MOTOR_H_
00011
00012 #include"..\..\LIB\LSTD_TYPES.h"
00013 #include"DC_MOTOR_CFG.h"
00014
00015 void H_DcMotorInit(void);
00016 void H_DcMotorSetDirection(u8);
00017 void H_DcMotorSetSpeed(u32);
00018 void H_DcMotorStart(u8);
00019 void H_DcMotorStop(void);
00020
00021 #define CLK_W      1
00022 #define A_CLK_W    2
00023
00024
00025
00026#endif /* DC_MOTOR_H_ */

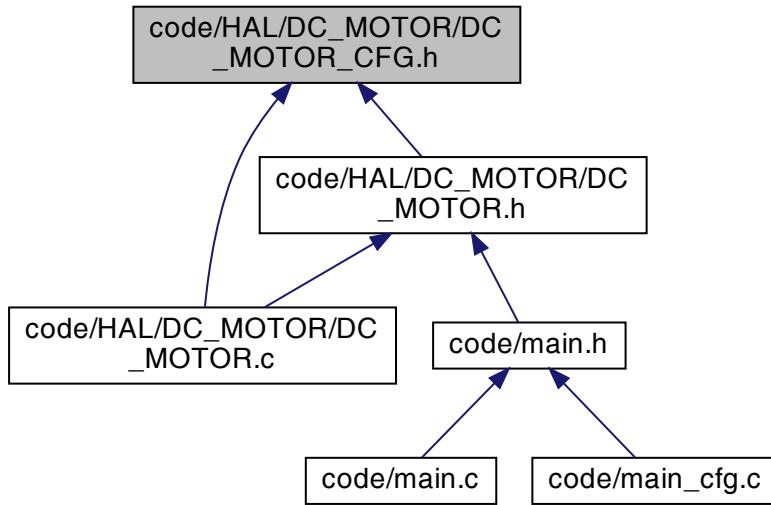
```

22.47 code/HAL/DC_MOTOR/DC_MOTOR_CFG.h File Reference

#include "..\..\MCAL\MDIO\MDIO_Interface.h"
 Include dependency graph for DC_MOTOR_CFG.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define IN_1 PIN0`
- `#define IN_2 PIN1`
- `#define DC_M_PORT MDIO_PORTB`

22.47.1 Macro Definition Documentation

22.47.1.1 DC_M_PORT `#define DC_M_PORT MDIO_PORTB`

Definition at line 17 of file [DC_MOTOR_CFG.h](#).

22.47.1.2 IN_1 `#define IN_1 PIN0`

Definition at line 14 of file [DC_MOTOR_CFG.h](#).

22.47.1.3 IN_2 `#define IN_2 PIN1`

Definition at line 15 of file [DC_MOTOR_CFG.h](#).

22.48 DC_MOTOR_CFG.h

```

00001 /*
00002  * DC_MOTOR_CFG.h
00003  *
00004  * Created: 4/18/2022 9:47:17 AM
00005  * Author: Dell
00006 */
00007
00008
00009 #ifndef DC_MOTOR_CFG_H_
00010 #define DC_MOTOR_CFG_H_
00011
00012 #include"../../MCAL\MDIO\MDIO_Interface.h"
00013
00014 #define IN_1          PIN0
00015 #define IN_2          PIN1
00016
00017 #define DC_M_PORT      MDIO_PORTB
00018
00019
00020#endif /* DC_MOTOR_CFG_H_ */

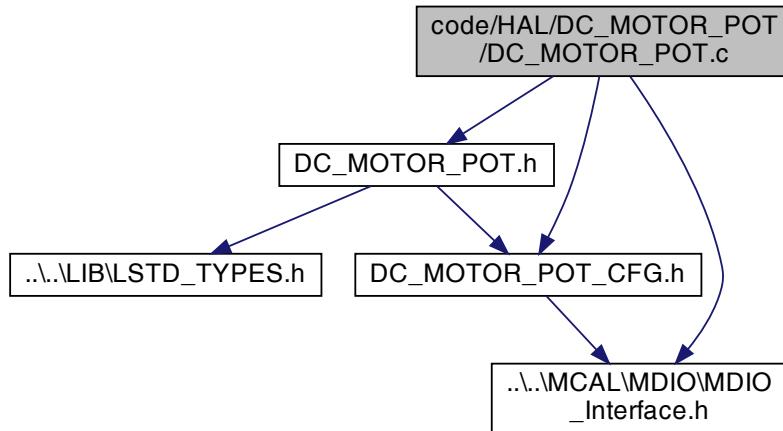
```

22.49 code/HAL/DC_MOTOR_POT/DC_MOTOR_POT.c File Reference

```

#include "DC_MOTOR_POT.h"
#include "../../MCAL/MDIO/MDIO_Interface.h"
#include "DC_MOTOR_POT_CFG.h"
Include dependency graph for DC_MOTOR_POT.c:

```



Functions

- void [H_DcMotorPotInit](#) (void)
- void [H_DcMotorPotStart](#) (u8 u8_local_direction)
- void [H_DcMotorPotStop](#) (void)

22.49.1 Function Documentation

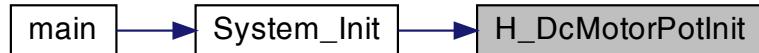
```
22.49.1.1 H_DcMotorPotInit() void H_DcMotorPotInit (
    void )
```

Definition at line 13 of file [DC_MOTOR_POT.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



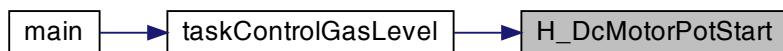
```
22.49.1.2 H_DcMotorPotStart() void H_DcMotorPotStart (
    u8 u8_local_direction )
```

Definition at line 20 of file [DC_MOTOR_POT.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.49.1.3 H_DcMotorPotStop() void H_DcMotorPotStop (void)

Definition at line 37 of file [DC_MOTOR_POT.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.50 DC_MOTOR_POT.c

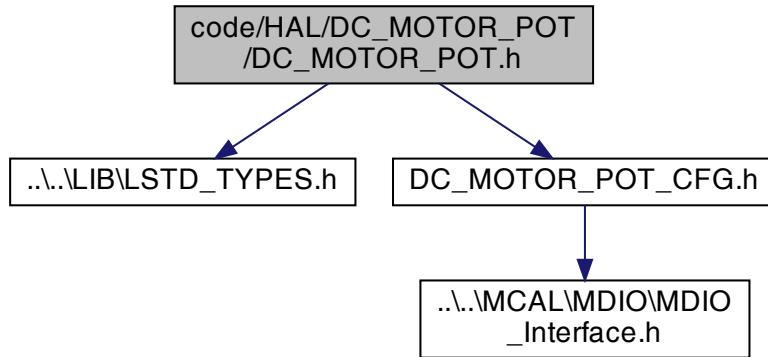
```

00001 /*
00002 * DC_MOTOR.c
00003 *
00004 * Created: 4/18/2022 9:46:43 AM
00005 * Author: Dell
00006 */
00007
00008 #include "DC_MOTOR_POT.h"
00009
00010 #include "../../MCAL/MDIO/MDIO_Interface.h"
00011 #include "DC_MOTOR_POT_CFG.h"
00012
00013 void H_DcMotorPotInit(void)
00014 {
00015     MDIO_Error_State_SetPinDirection(IN_1_POT, DC_M_POT_PORT, PIN_OUTPUT);
00016     //MDIO_Error_State_SetPinDirection(IN_2, DC_M_PORT, PIN_OUTPUT);
00017     //M_Pwm0Init();
00018 }
00019
00020 void H_DcMotorPotStart(u8 u8_local_direction)
00021 {
00022     //switch(u8_local_direction)
00023     //{
00024         //case CLK_W:
00025             MDIO_Error_State_SetPinValue(IN_1_POT, DC_M_POT_PORT, PIN_HIGH);
00026             //MDIO_Error_State_SetPinValue(IN_2, DC_M_PORT, PIN_LOW);
00027         //break;
00028         //case A_CLK_W:
00029             //MDIO_Error_State_SetPinValue(IN_1, DC_M_PORT, PIN_LOW);
00030             //MDIO_Error_State_SetPinValue(IN_2, DC_M_PORT, PIN_HIGH);
00031         //break;
00032         //default:
00033         //break;
00034     //}
00035 }
00036
00037 void H_DcMotorPotStop(void)
00038 {
00039     MDIO_Error_State_SetPinValue(IN_1_POT, DC_M_POT_PORT, PIN_LOW);
00040     //MDIO_Error_State_SetPinValue(IN_2, DC_M_PORT, PIN_LOW);
00041 }

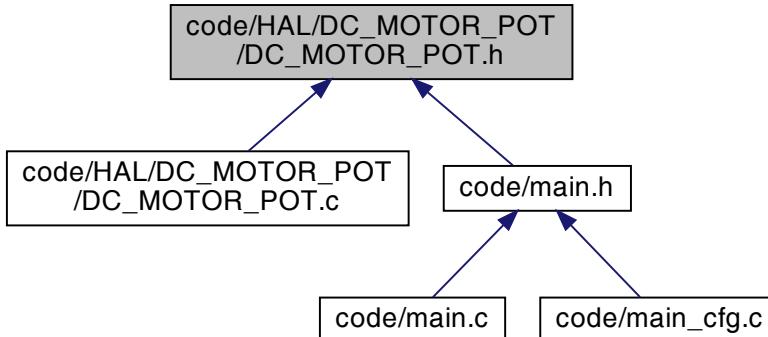
```

22.51 code/HAL/DC_MOTOR_POT/DC_MOTOR_POT.h File Reference

```
#include "..\..\LIB\LSTD_TYPES.h"
#include "DC_MOTOR_POT_CFG.h"
Include dependency graph for DC_MOTOR_POT.h:
```



This graph shows which files directly or indirectly include this file:



Macros

- #define CLK_W 1
- #define A_CLK_W 2

Functions

- void `H_DcMotorPotInit` (void)
- void `H_DcMotorSetDirection` (u8)
- void `H_DcMotorSetSpeed` (u32)
- void `H_DcMotorPotStart` (u8)
- void `H_DcMotorPotStop` (void)

22.51.1 Macro Definition Documentation

22.51.1.1 A_CLK_W #define A_CLK_W 2

Definition at line 22 of file [DC_MOTOR_POT.h](#).

22.51.1.2 CLK_W #define CLK_W 1

Definition at line 21 of file [DC_MOTOR_POT.h](#).

22.51.2 Function Documentation

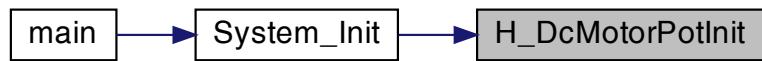
22.51.2.1 H_DcMotorPotInit() void H_DcMotorPotInit (void)

Definition at line 13 of file [DC_MOTOR_POT.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
22.51.2.2 H_DcMotorPotStart() void H_DcMotorPotStart (
    u8 u8_local_direction )
```

Definition at line 20 of file [DC_MOTOR_POT.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



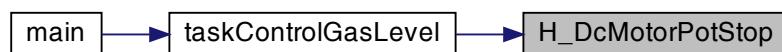
```
22.51.2.3 H_DcMotorPotStop() void H_DcMotorPotStop (
    void )
```

Definition at line 37 of file [DC_MOTOR_POT.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.51.2.4 H_DcMotorSetDirection() void H_DcMotorSetDirection (
 u8)

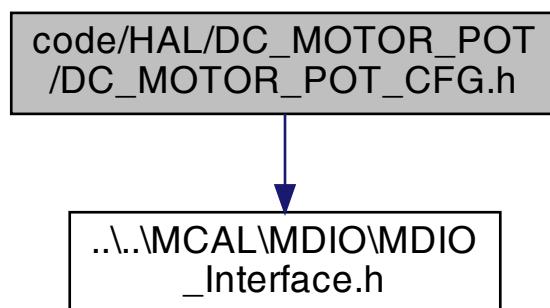
22.51.2.5 H_DcMotorSetSpeed() void H_DcMotorSetSpeed (
 u32)

22.52 DC_MOTOR_POT.h

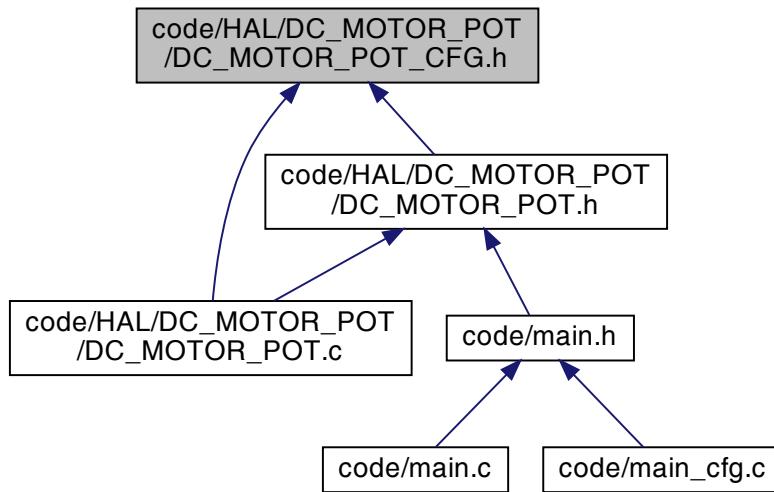
```
00001 /*
00002  * DC_MOTOR.h
00003  *
00004  * Created: 4/18/2022 9:46:58 AM
00005  * Author: Dell
00006 */
00007
00008
00009 #ifndef DC_MOTOR_POT_H_
00010 #define DC_MOTOR_POT_H_
00011
00012 #include "..\..\LIB\LSTD_TYPES.h"
00013 #include "DC_MOTOR_POT_CFG.h"
00014
00015 void H_DcMotorPotInit(void);
00016 void H_DcMotorSetDirection(u8);
00017 void H_DcMotorSetSpeed(u32);
00018 void H_DcMotorPotStart(u8);
00019 void H_DcMotorPotStop(void);
00020
00021 #define CLK_W      1
00022 #define A_CLK_W    2
00023
00024
00025
00026 #endif /* DC_MOTOR_H_ */
```

22.53 code/HAL/DC_MOTOR_POT/DC_MOTOR_POT_CFG.h File Reference

```
#include "..\..\MCAL\MDIO\MDIO_Interface.h"
Include dependency graph for DC_MOTOR_POT_CFG.h:
```



This graph shows which files directly or indirectly include this file:



Macros

- `#define IN_1_POT PIN7`
- `#define DC_M_POT_PORT MDIO_PORTD`

22.53.1 Macro Definition Documentation

22.53.1.1 DC_M_POT_PORT `#define DC_M_POT_PORT MDIO_PORTD`

Definition at line 17 of file [DC_MOTOR_POT_CFG.h](#).

22.53.1.2 IN_1_POT `#define IN_1_POT PIN7`

Definition at line 14 of file [DC_MOTOR_POT_CFG.h](#).

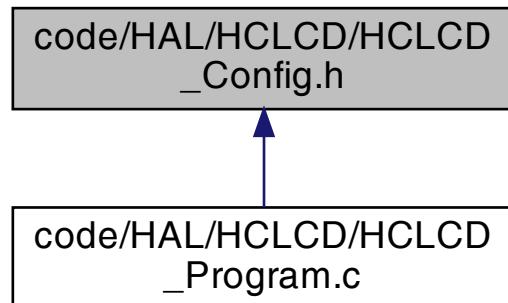
22.54 DC_MOTOR_POT_CFG.h

```

00001 /*
00002 * DC_MOTOR_CFG.h
00003 *
00004 * Created: 4/18/2022 9:47:17 AM
00005 * Author: Dell
00006 */
00007
00008
00009 #ifndef DC_MOTOR_POT_CFG_H_
00010 #define DC_MOTOR_POT_CFG_H_
00011
00012 #include"..\..\MCAL\MDIO\MDIO_Interface.h"
00013
00014 #define IN_1_POT          PIN7
00015 //#define IN_2          PIN1
00016
00017 #define DC_M_POT_PORT      MDIO_PORTD
00018
00019
00020 #endif /* DC_MOTOR_CFG_H_ */
  
```

22.55 code/HAL/HCLCD/HCLCD_Config.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- `#define RS PIN1`
- `#define E PIN2`
- `#define CONTROL_PORT MDIO_PORTA`
- `#define DATA_PORT MDIO_PORTA`
- `#define HCLCD_PINSTART PIN3`
- `#define HCLCD_PINEND PIN6`
- `#define HCLCD_FUNCTION_SET FUNCTION_SET_4BITS_2LINES`
- `#define HCLCD_DISPLAY_ON_OFF DISPLAY_ON_CURSOR_OFF`
- `#define HCLCD_ENTRY_MODE_SET ENTRY_MODE_SET_INCREASE`

22.55.1 Macro Definition Documentation

22.55.1.1 CONTROL_PORT `#define CONTROL_PORT MDIO_PORTA`

Definition at line 9 of file [HCLCD_Config.h](#).

22.55.1.2 DATA_PORT `#define DATA_PORT MDIO_PORTA`

Definition at line 12 of file [HCLCD_Config.h](#).

22.55.1.3 E #define E PIN2

Definition at line 6 of file [HCLCD_Config.h](#).

22.55.1.4 HCLCD_DISPLAY_ON_OFF #define HCLCD_DISPLAY_ON_OFF DISPLAY_ON_CURSOR_OFF

Definition at line 32 of file [HCLCD_Config.h](#).

22.55.1.5 HCLCD_ENTRY_MODE_SET #define HCLCD_ENTRY_MODE_SET ENTRY_MODE_SET_INCREASE

Definition at line 40 of file [HCLCD_Config.h](#).

22.55.1.6 HCLCD_FUNCTION_SET #define HCLCD_FUNCTION_SET FUNCTION_SET_4BITS_2LINES

Definition at line 23 of file [HCLCD_Config.h](#).

22.55.1.7 HCLCD_PINEND #define HCLCD_PINEND PIN6

Definition at line 15 of file [HCLCD_Config.h](#).

22.55.1.8 HCLCD_PINSTART #define HCLCD_PINSTART PIN3

Definition at line 14 of file [HCLCD_Config.h](#).

22.55.1.9 RS #define RS PIN1

Definition at line 4 of file [HCLCD_Config.h](#).

22.56 HCLCD_Config.h

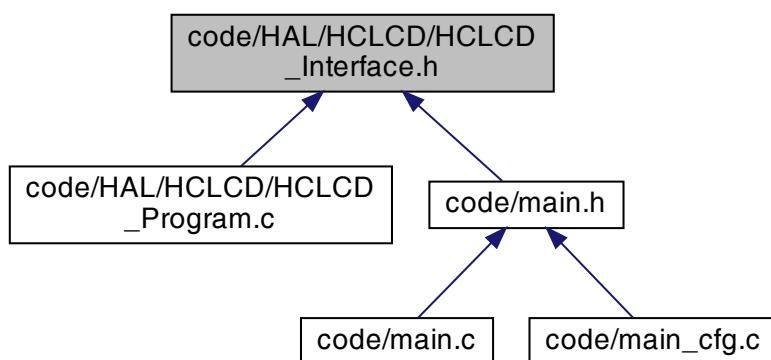
```

00001
00002
00003 /*Configure RS,RW,E pin -->(PIN0 TO PIN7) */
00004 #define RS  PIN1
00005 //#define RW  PIN1
00006 #define E   PIN2
00007
00008 /*Configure Control port -->(MDIO_PORTA,MDIO_PORTB,MDIO_PORTC,MDIO_PORTD) */
00009 #define CONTROL_PORT    MDIO_PORTA
00010
00011 /*Configure Data port -->(MDIO_PORTA,MDIO_PORTB,MDIO_PORTC,MDIO_PORTD) */
00012 #define DATA_PORT      MDIO_PORTA
00013
00014 #define HCLCD_PINSTART  PIN3
00015 #define HCLCD_PINEND   PIN6
00016
00017 /*Function set options:
00018 * 1- FUNCTION_SET_8BITS_2LINES
00019 * 2- FUNCTION_SET_8BITS_1LINES
00020 * 3- FUNCTION_SET_4BITS_2LINES
00021 * 4- FUNCTION_SET_4BITS_1LINES
00022 */
00023 #define HCLCD_FUNCTION_SET   FUNCTION_SET_4BITS_2LINES
00024
00025 /*DISPLAY ON OFF options:
00026 * 1- DISPLAY_ON_CURSOR_ON_BLINKING_ON
00027 * 2- DISPLAY_ON_CURSOR_ON_BLINKING_OFF
00028 * 3- DISPLAY_ON_CURSOR_OFF
00029 * 4- DISPLAY_OFF
00030 */
00031
00032 #define HCLCD_DISPLAY_ON_OFF  DISPLAY_ON_CURSOR_OFF
00033
00034 /*HLCD ENTRY MODE SET Options:
00035 * 1- ENTRY_MODE_SET_INCREASE_WITH_SHIFT
00036 * 2- ENTRY_MODE_SET_INCREASE
00037 * 3- ENTRY_MODE_SET_DECREASE_WITH_SHIFT
00038 * 4- ENTRY_MODE_SET_DECREASE
00039 */
00040 #define HCLCD_ENTRY_MODE_SET  ENTRY_MODE_SET_INCREASE
00041

```

22.57 code/HAL/HCLCD/HCLCD_Interface.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define DISPLAY_CLEAR 0b00000001
- #define HCLCD_LINE1 1
- #define HCLCD_LINE2 2

Functions

- void `HCLCD_Vid8Bits_Init` (void)
- void `HCLCD_Vid4Bits_Init` (void)
- void `HCLCD_VidWriteChar_8Bits` (`u8 Copy_u8Data`)
- void `HCLCD_VidSendChar_4Bits` (`u8 Copy_u8Data`)
- void `HCLCD_VidWriteCommand_8Bits` (`u8 Copy_u8Command`)
- void `HCLCD_VidWriteCommand_4Bits` (`u8 Copy_u8Command`)
- void `HCLCD_VidWriteString_8Bits` (`u8 *PCopy_u8String`)
- void `HCLCD_VidWriteString_4Bits` (`u8 *PCopy_u8String`)
- void `HCLCD_VidWriteNumber_8Bits` (`u32 Copy_u8Number`)
- void `HCLCD_VidWriteNumber_4Bits` (`u32 Copy_u8Number`)
- void `HCLCD_VidSetPosition` (`u8 Copy_u8LineNumber`, `u8 Copy_u8PositionNumber`)
- void `HCLCD_VidSetPosition_4BitsMode` (`u8 Copy_u8LineNumber`, `u8 Copy_u8PositionNumber`)

22.57.1 Macro Definition Documentation

22.57.1.1 DISPLAY_CLEAR `#define DISPLAY_CLEAR 0b00000001`

Definition at line 5 of file `HCLCD_Interface.h`.

22.57.1.2 HCLCD_LINE1 `#define HCLCD_LINE1 1`

Definition at line 7 of file `HCLCD_Interface.h`.

22.57.1.3 HCLCD_LINE2 `#define HCLCD_LINE2 2`

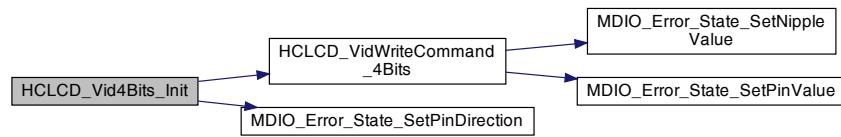
Definition at line 8 of file `HCLCD_Interface.h`.

22.57.2 Function Documentation

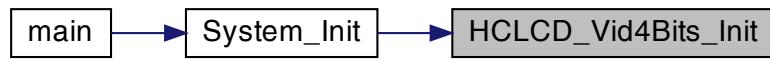
22.57.2.1 HCLCD_Vid4Bits_Init() void HCLCD_Vid4Bits_Init (void)

Definition at line 126 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



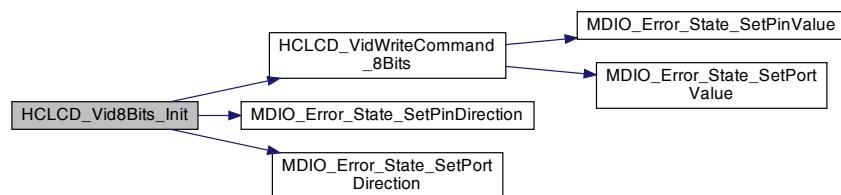
Here is the caller graph for this function:



22.57.2.2 HCLCD_Vid8Bits_Init() void HCLCD_Vid8Bits_Init (void)

Definition at line 26 of file [HCLCD_Program.c](#).

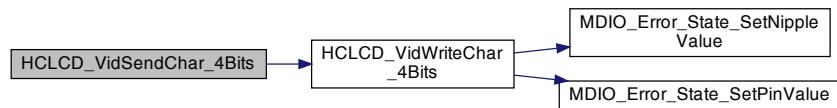
Here is the call graph for this function:



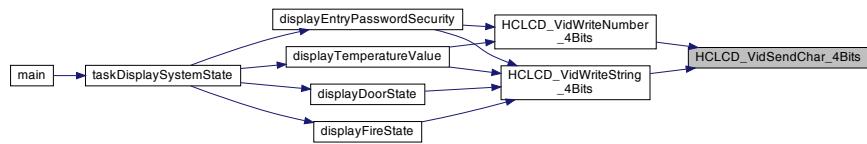
22.57.2.3 HCLCD_VidSendChar_4Bits() void HCLCD_VidSendChar_4Bits (
u8 Copy_u8Data)

Definition at line 171 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.57.2.4 HCLCD_VidSetPosition() void HCLCD_VidSetPosition (
u8 Copy_u8LineNumber,
u8 Copy_u8PositionNumber)

Definition at line 95 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



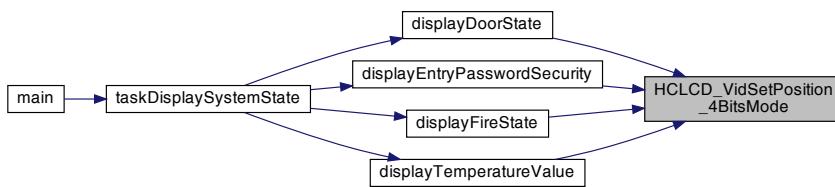
```
22.57.2.5 HCLCD_VidSetPosition_4BitsMode() void HCLCD_VidSetPosition_4BitsMode (
    u8 Copy_u8LineNumber,
    u8 Copy_u8PositionNumber )
```

Definition at line 207 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



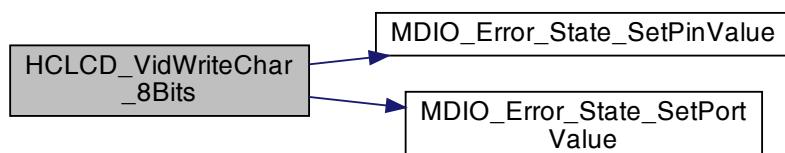
Here is the caller graph for this function:



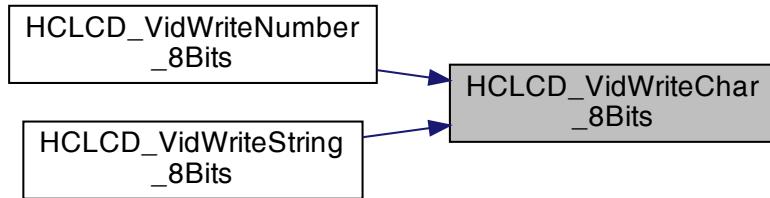
```
22.57.2.6 HCLCD_VidWriteChar_8Bits() void HCLCD_VidWriteChar_8Bits (
    u8 Copy_u8Data )
```

Definition at line 48 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



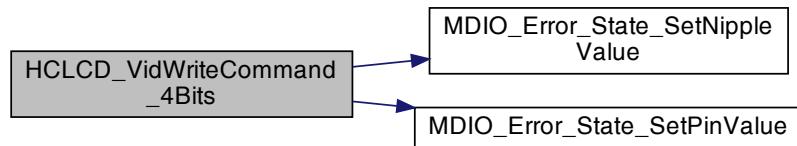
Here is the caller graph for this function:



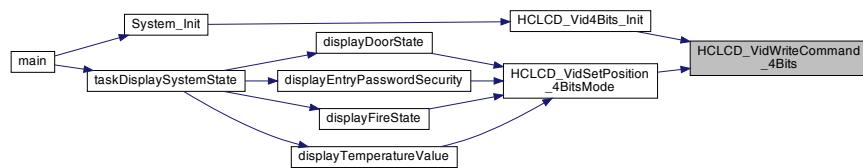
22.57.2.7 HCLCD_VidWriteCommand_4Bits() `void HCLCD_VidWriteCommand_4Bits (u8 Copy_u8Command)`

Definition at line 111 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



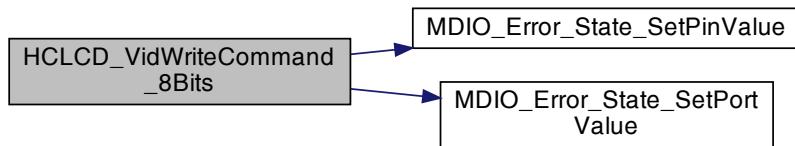
Here is the caller graph for this function:



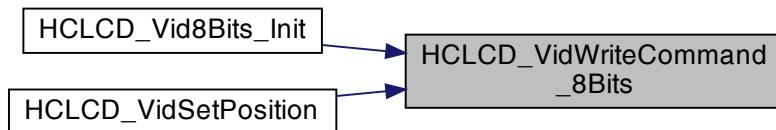
22.57.2.8 HCLCD_VidWriteCommand_8Bits() void HCLCD_VidWriteCommand_8Bits (
 8 Copy_u8Command)

Definition at line 10 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



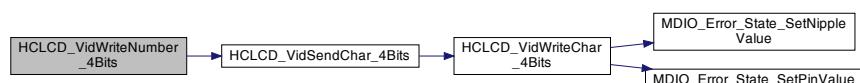
Here is the caller graph for this function:



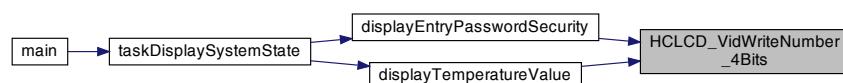
22.57.2.9 HCLCD_VidWriteNumber_4Bits() void HCLCD_VidWriteNumber_4Bits (
 32 Copy_u8Number)

Definition at line 185 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



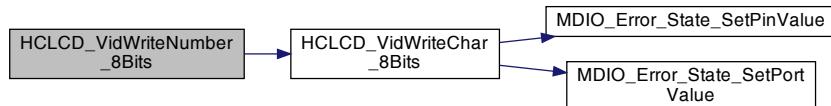
Here is the caller graph for this function:



```
22.57.2.10 HCLCD_VidWriteNumber_8Bits() void HCLCD_VidWriteNumber_8Bits (
    u32 Copy_u8Number )
```

Definition at line 73 of file [HCLCD_Program.c](#).

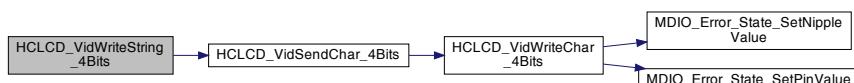
Here is the call graph for this function:



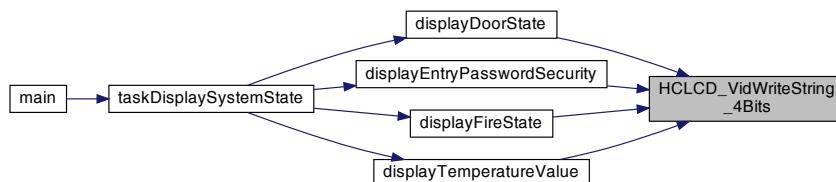
```
22.57.2.11 HCLCD_VidWriteString_4Bits() void HCLCD_VidWriteString_4Bits (
    u8 * PCopy_u8String )
```

Definition at line 176 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



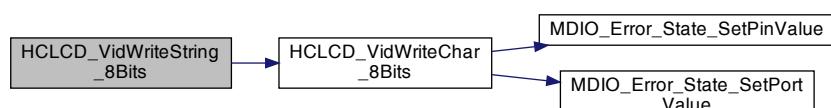
Here is the caller graph for this function:



```
22.57.2.12 HCLCD_VidWriteString_8Bits() void HCLCD_VidWriteString_8Bits (
    u8 * PCopy_u8String )
```

Definition at line 64 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



22.58 HCLCD_Interface.h

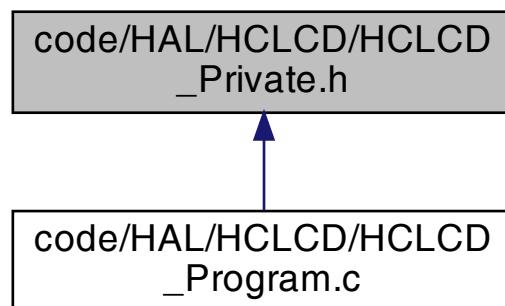
```

00001 #ifndef HCLCD_INTERFACE_H_
00002 #define HCLCD_INTERFACE_H_
00003
00004 /*LCD DISPLAY CLEAR Command --> No options*/
00005 #define DISPLAY_CLEAR    0b00000001
00006
00007 #define HCLCD_LINE1      1
00008 #define HCLCD_LINE2      2
00009
00010 void HCLCD_Vid8Bits_Init(void);
00011 void HCLCD_Vid4Bits_Init(void);
00012
00013 void HCLCD_VidWriteChar_8Bits(u8 Copy_u8Data);
00014
00015 void HCLCD_VidSendChar_4Bits(u8 Copy_u8Data);
00016
00017 void HCLCD_VidWriteCommand_8Bits(u8 Copy_u8Command);
00018 void HCLCD_VidWriteCommand_4Bits(u8 Copy_u8Command);
00019
00020
00021 void HCLCD_VidWriteString_8Bits(u8* PCopy_u8String);
00022 void HCLCD_VidWriteString_4Bits(u8* PCopy_u8String);
00023
00024 void HCLCD_VidWriteNumber_8Bits(u32 Copy_u8Number);
00025 void HCLCD_VidWriteNumber_4Bits(u32 Copy_u8Number);
00026
00027 void HCLCD_VidSetPosition(u8 Copy_u8LineNumber , u8 Copy_u8PositionNumber);
00028
00029 void HCLCD_VidSetPosition_4BitsMode(u8 Copy_u8LineNumber , u8 Copy_u8PositionNumber);
00030#endif

```

22.59 code/HAL/HCLCD/HCLCD_Private.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define FUNCTION_SET_8BITS_2LINES 0b00111000
- #define FUNCTION_SET_8BITS_1LINES 0b00110100
- #define FUNCTION_SET_4BITS_2LINES 0b00101000
- #define FUNCTION_SET_4BITS_1LINES 0b00100100
- #define DISPLAY_ON_CURSOR_ON_BLINKING_ON 0b00001111
- #define DISPLAY_ON_CURSOR_ON_BLINKING_OFF 0b00001110
- #define DISPLAY_ON_CURSOR_OFF 0b00001100
- #define DISPLAY_OFF 0b00001000

- #define ENTRY_MODE_SET_INCREASE_WITH_SHIFT 0b00000111
- #define ENTRY_MODE_SET_INCREASE 0b00000110
- #define ENTRY_MODE_SET_DECREASE_WITH_SHIFT 0b00000101
- #define ENTRY_MODE_SET_DECREASE 0b00000100
- #define LINE1_OFFSET_ADDRESS 128
- #define LINE2_OFFSET_ADDRESS 192

Functions

- void HCLCD_VidWriteChar_4Bits (u8 Copy_u8Data)

22.59.1 Macro Definition Documentation

22.59.1.1 DISPLAY_OFF #define DISPLAY_OFF 0b00001000

Definition at line 15 of file [HCLCD_Private.h](#).

22.59.1.2 DISPLAY_ON_CURSOR_OFF #define DISPLAY_ON_CURSOR_OFF 0b00001100

Definition at line 14 of file [HCLCD_Private.h](#).

22.59.1.3 DISPLAY_ON_CURSOR_ON_BLINKING_OFF #define DISPLAY_ON_CURSOR_ON_BLINKING_OFF 0b00001110

Definition at line 13 of file [HCLCD_Private.h](#).

22.59.1.4 DISPLAY_ON_CURSOR_ON_BLINKING_ON #define DISPLAY_ON_CURSOR_ON_BLINKING_ON 0b00001111

Definition at line 12 of file [HCLCD_Private.h](#).

22.59.1.5 ENTRY_MODE_SET_DECREASE #define ENTRY_MODE_SET_DECREASE 0b00000100

Definition at line 21 of file [HCLCD_Private.h](#).

22.59.1.6 ENTRY_MODE_SET_DECREASE_WITH_SHIFT #define ENTRY_MODE_SET_DECREASE_WITH_←
SHIFT 0b00000101

Definition at line 20 of file [HCLCD_Private.h](#).

22.59.1.7 ENTRY_MODE_SET_INCREASE #define ENTRY_MODE_SET_INCREASE 0b00000110

Definition at line 19 of file [HCLCD_Private.h](#).

22.59.1.8 ENTRY_MODE_SET_INCREASE_WITH_SHIFT #define ENTRY_MODE_SET_INCREASE_WITH_←
SHIFT 0b00000111

Definition at line 18 of file [HCLCD_Private.h](#).

22.59.1.9 FUNCTION_SET_4BITS_1LINES #define FUNCTION_SET_4BITS_1LINES 0b00100100

Definition at line 9 of file [HCLCD_Private.h](#).

22.59.1.10 FUNCTION_SET_4BITS_2LINES #define FUNCTION_SET_4BITS_2LINES 0b00101000

Definition at line 8 of file [HCLCD_Private.h](#).

22.59.1.11 FUNCTION_SET_8BITS_1LINES #define FUNCTION_SET_8BITS_1LINES 0b00110100

Definition at line 7 of file [HCLCD_Private.h](#).

22.59.1.12 FUNCTION_SET_8BITS_2LINES #define FUNCTION_SET_8BITS_2LINES 0b00111000

Definition at line 6 of file [HCLCD_Private.h](#).

22.59.1.13 LINE1_OFFSET_ADDRESS #define LINE1_OFFSET_ADDRESS 128

Definition at line 23 of file [HCLCD_Private.h](#).

22.59.1.14 LINE2_OFFSET_ADDRESS #define LINE2_OFFSET_ADDRESS 192

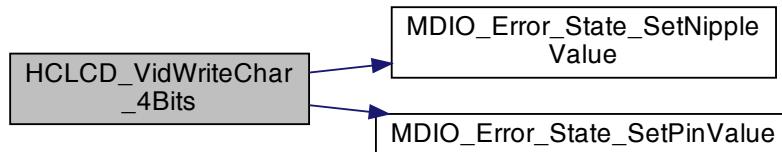
Definition at line 24 of file [HCLCD_Private.h](#).

22.59.2 Function Documentation

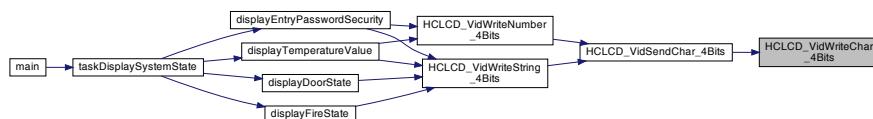
22.59.2.1 HCLCD_VidWriteChar_4Bits() void HCLCD_VidWriteChar_4Bits (u8 Copy_u8Data)

Definition at line 156 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.60 HCLCD_Private.h

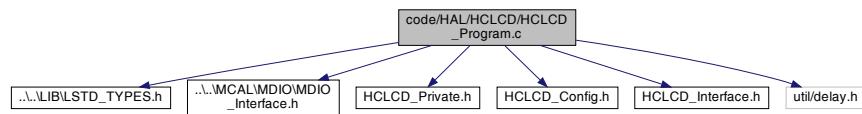
```

00001
00002 #ifndef HCLCD_PRIVATE_H_
00003 #define HCLCD_PRIVATE_H_
00004
00005 /*Function set Modes*/
00006 #define FUNCTION_SET_8BITS_2LINES      0b00111000
00007 #define FUNCTION_SET_8BITS_1LINES      0b00110100
00008 #define FUNCTION_SET_4BITS_2LINES      0b00101000
00009 #define FUNCTION_SET_4BITS_1LINES      0b00100100
00010
00011 /*Display on/off Modes*/
00012 #define DISPLAY_ON_CURSOR_ON_BLINKING_ON 0b00001111
00013 #define DISPLAY_ON_CURSOR_ON_BLINKING_OFF 0b00001110
00014 #define DISPLAY_ON_CURSOR_OFF          0b00001100
00015 #define DISPLAY_OFF                  0b00001000
00016
00017 /*ENTRY MODE*/
00018 #define ENTRY_MODE_SET_INCREASE_WITH_SHIFT 0b000000111
00019 #define ENTRY_MODE_SET_INCREASE        0b000000110
00020 #define ENTRY_MODE_SET_DECREASE_WITH_SHIFT 0b000000101
00021 #define ENTRY_MODE_SET_DECREASE       0b000000100
00022
00023 #define LINE1_OFFSET_ADDRESS     128
00024 #define LINE2_OFFSET_ADDRESS     192
00025
00026 void HCLCD_VidWriteChar_4Bits(u8 Copy_u8Data);
00027
00028 #endif
  
```

22.61 code/HAL/HCLCD/HCLCD_Program.c File Reference

```
#include "...\\LIB\\LSTD_TYPES.h"
#include "...\\MCAL\\MDIO\\MDIO_Interface.h"
#include "HCLCD_Private.h"
#include "HCLCD_Config.h"
#include "HCLCD_Interface.h"
#include "util/delay.h"

Include dependency graph for HCLCD_Program.c:
```



Functions

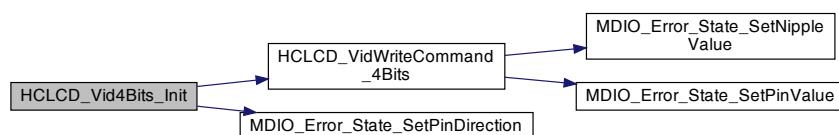
- void [HCLCD_VidWriteCommand_8Bits \(u8 Copy_u8Command\)](#)
- void [HCLCD_Vid8Bits_Init \(void\)](#)
- void [HCLCD_VidWriteChar_8Bits \(u8 Copy_u8Data\)](#)
- void [HCLCD_VidWriteString_8Bits \(u8 *PCopy_u8String\)](#)
- void [HCLCD_VidWriteNumber_8Bits \(u32 Copy_u8Number\)](#)
- void [HCLCD_VidSetPosition \(u8 Copy_u8LineNumber, u8 Copy_u8PositionNumber\)](#)
- void [HCLCD_VidWriteCommand_4Bits \(u8 Copy_u8Command\)](#)
- void [HCLCD_Vid4Bits_Init \(void\)](#)
- void [HCLCD_VidWriteChar_4Bits \(u8 Copy_u8Data\)](#)
- void [HCLCD_VidSendChar_4Bits \(u8 Copy_u8Data\)](#)
- void [HCLCD_VidWriteString_4Bits \(u8 *PCopy_u8String\)](#)
- void [HCLCD_VidWriteNumber_4Bits \(u32 Copy_u8Number\)](#)
- void [HCLCD_VidSetPosition_4BitsMode \(u8 Copy_u8LineNumber, u8 Copy_u8PositionNumber\)](#)

22.61.1 Function Documentation

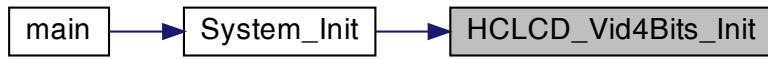
22.61.1.1 HCLCD_Vid4Bits_Init() `void HCLCD_Vid4Bits_Init (void)`

Definition at line 126 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



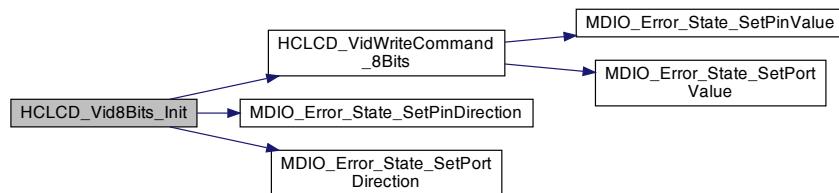
Here is the caller graph for this function:



22.61.1.2 HCLCD_Vid8Bits_Init() `void HCLCD_Vid8Bits_Init (void)`

Definition at line 26 of file [HCLCD_Program.c](#).

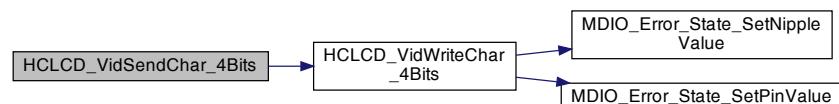
Here is the call graph for this function:



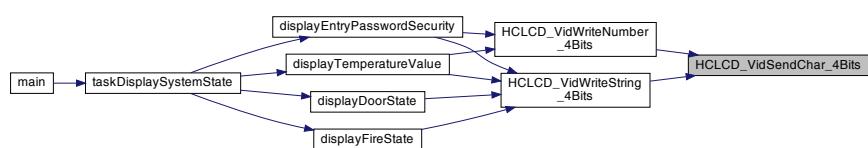
22.61.1.3 HCLCD_VidSendChar_4Bits() `void HCLCD_VidSendChar_4Bits (u8 Copy_u8Data)`

Definition at line 171 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
22.61.1.4 HCLCD_VidSetPosition() void HCLCD_VidSetPosition (
    u8 Copy_u8LineNumber,
    u8 Copy_u8PositionNumber )
```

Definition at line 95 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



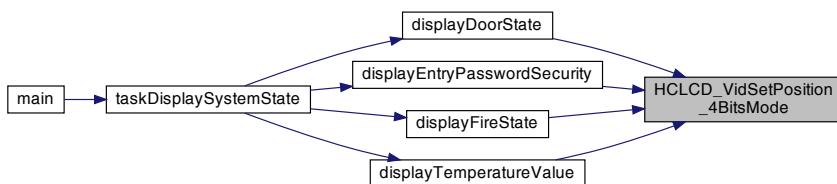
```
22.61.1.5 HCLCD_VidSetPosition_4BitsMode() void HCLCD_VidSetPosition_4BitsMode (
    u8 Copy_u8LineNumber,
    u8 Copy_u8PositionNumber )
```

Definition at line 207 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



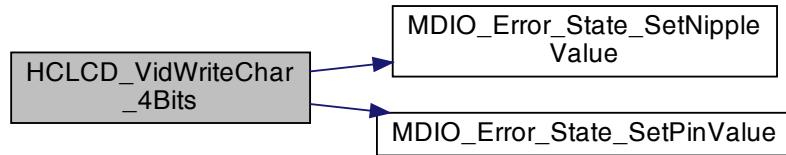
Here is the caller graph for this function:



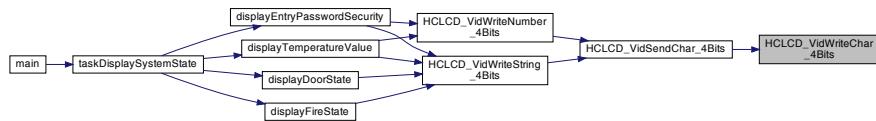
```
22.61.1.6 HCLCD_VidWriteChar_4Bits() void HCLCD_VidWriteChar_4Bits (
    u8 Copy_u8Data )
```

Definition at line 156 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



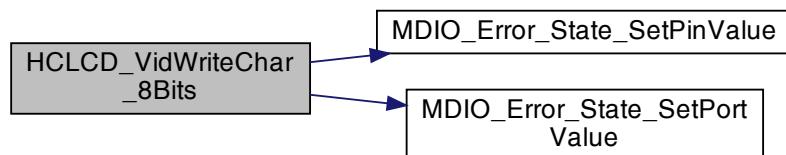
Here is the caller graph for this function:



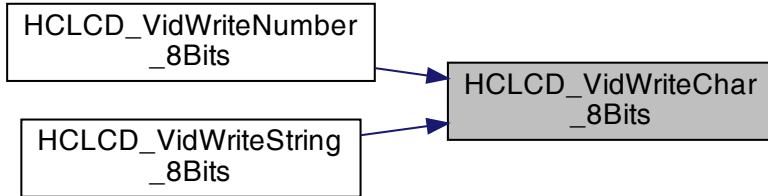
```
22.61.1.7 HCLCD_VidWriteChar_8Bits() void HCLCD_VidWriteChar_8Bits (
    u8 Copy_u8Data )
```

Definition at line 48 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



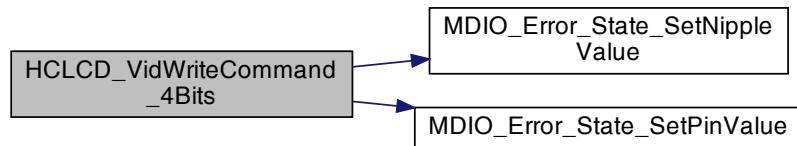
Here is the caller graph for this function:



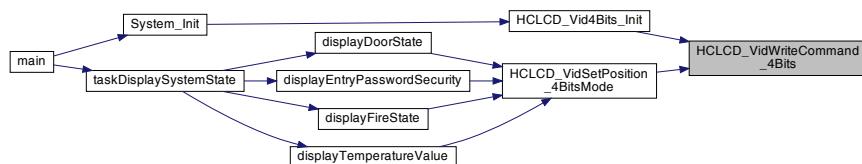
22.61.1.8 HCLCD_VidWriteCommand_4Bits() `void HCLCD_VidWriteCommand_4Bits (u8 Copy_u8Command)`

Definition at line 111 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



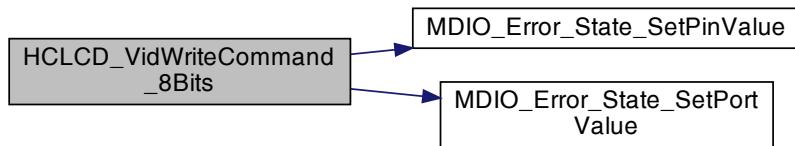
Here is the caller graph for this function:



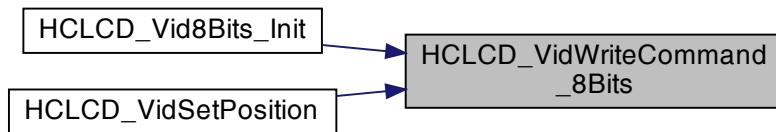
```
22.61.1.9 HCLCD_VidWriteCommand_8Bits() void HCLCD_VidWriteCommand_8Bits (
    u8 Copy_u8Command )
```

Definition at line 10 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



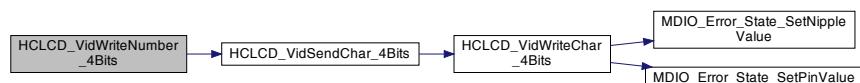
Here is the caller graph for this function:



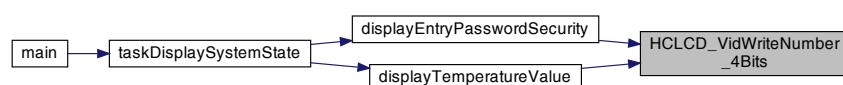
```
22.61.1.10 HCLCD_VidWriteNumber_4Bits() void HCLCD_VidWriteNumber_4Bits (
    u32 Copy_u8Number )
```

Definition at line 185 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



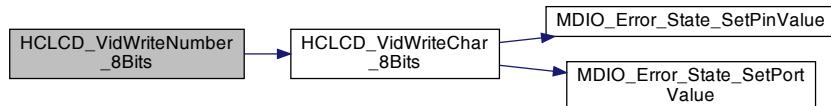
Here is the caller graph for this function:



22.61.1.11 HCLCD_VidWriteNumber_8Bits() void HCLCD_VidWriteNumber_8Bits (
 u32 Copy_u8Number)

Definition at line 73 of file [HCLCD_Program.c](#).

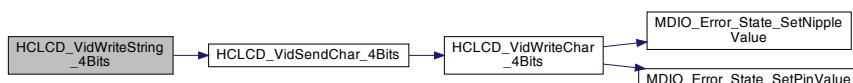
Here is the call graph for this function:



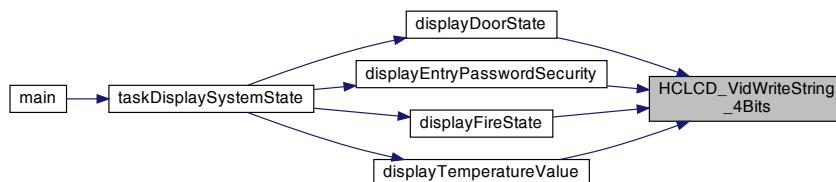
22.61.1.12 HCLCD_VidWriteString_4Bits() void HCLCD_VidWriteString_4Bits (
 u8 * PCopy_u8String)

Definition at line 176 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



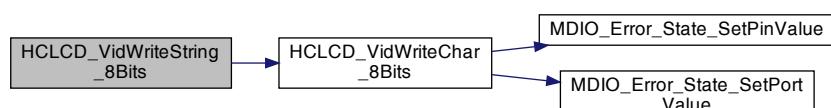
Here is the caller graph for this function:



22.61.1.13 HCLCD_VidWriteString_8Bits() void HCLCD_VidWriteString_8Bits (
 u8 * PCopy_u8String)

Definition at line 64 of file [HCLCD_Program.c](#).

Here is the call graph for this function:



22.62 HCLCD_Program.c

```

00001 #include"..\..\LIB\LSTD_TYPES.h"
00002 #include"..\..\MCAL\MDIO\MDIO_Interface.h"
00003
00004 #include"HCLCD_Private.h"
00005 #include"HCLCD_Config.h"
00006 #include"HCLCD_Interface.h"
00007
00008 #include"util/delay.h"
00009
0010 void HCLCD_VidWriteCommand_8Bits(u8 Copy_u8Command)
0011 {
0012     /*select Command register--> Write zero on Rs pin*/
0013     MDIO_Error_State_SetPinValue(RS, CONTROL_PORT, PIN_LOW);
0014     /*select Write mode--> Write zero on RW pin*/
0015     //MDIO_Error_State_SetPinValue(RW, CONTROL_PORT, PIN_LOW);
0016     /*Send Command ON port Data*/
0017     MDIO_Error_State_SetPortValue(DATA_PORT, Copy_u8Command);
0018     /*Send Enable*/
0019     MDIO_Error_State_SetPinValue(E, CONTROL_PORT, PIN_HIGH);
0020     _delay_ms(2);
0021     MDIO_Error_State_SetPinValue(E, CONTROL_PORT, PIN_LOW);
0022     _delay_ms(2);
0023     MDIO_Error_State_SetPinValue(E, CONTROL_PORT, PIN_HIGH);
0024 }
0025
0026 void HCLCD_Vid8Bits_Init(void)
0027 {
0028     /*LCD Data and control port intialization*/
0029     MDIO_Error_State_SetPortDirection(DATA_PORT, PORT_OUTPUT);
0030     MDIO_Error_State_SetPinDirection(RS, CONTROL_PORT, PIN_OUTPUT);
0031     //MDIO_Error_State_SetPinDirection(RW, CONTROL_PORT, PIN_OUTPUT);
0032     MDIO_Error_State_SetPinDirection(E, CONTROL_PORT, PIN_OUTPUT);
0033
0034     /*wait to 30ms*/
0035     _delay_ms(30);
0036     /*send function set command*/
0037     HCLCD_VidWriteCommand_8Bits(HCLCD_FUNCTION_SET);
0038     _delay_ms(1);
0039     /*send Display on/off command*/
0040     HCLCD_VidWriteCommand_8Bits(HCLCD_DISPLAY_ON_OFF);
0041     _delay_ms(1);
0042     /*send Display Clear command*/
0043     HCLCD_VidWriteCommand_8Bits(DISPLAY_CLEAR);
0044     _delay_ms(2);
0045     /*send Entry Mode set command*/
0046     HCLCD_VidWriteCommand_8Bits(HCLCD_ENTRY_MODE_SET);
0047 }
0048 void HCLCD_VidWriteChar_8Bits(u8 Copy_u8Data)
0049 {
0050     /*select Data register--> Write one on Rs pin*/
0051     MDIO_Error_State_SetPinValue(RS, CONTROL_PORT, PIN_HIGH);
0052     /*select Write mode--> Write zero on RW pin*/
0053     //MDIO_Error_State_SetPinValue(RW, CONTROL_PORT, PIN_LOW);
0054     /*Send Command ON port Data*/
0055     MDIO_Error_State_SetPortValue(DATA_PORT, Copy_u8Data);
0056     /*Send Enable*/
0057     MDIO_Error_State_SetPinValue(E, CONTROL_PORT, PIN_HIGH);
0058     _delay_ms(2);
0059     MDIO_Error_State_SetPinValue(E, CONTROL_PORT, PIN_LOW);
0060     _delay_ms(2);
0061     MDIO_Error_State_SetPinValue(E, CONTROL_PORT, PIN_HIGH);
0062 }
0063
0064 void HCLCD_VidWriteString_8Bits(u8* PCopy_u8String)
0065 {
0066     u8 Loc_u8Count=0;
0067     while (PCopy_u8String[Loc_u8Count]!=NULL)
0068     {
0069         HCLCD_VidWriteChar_8Bits(PCopy_u8String[Loc_u8Count]);
0070         Loc_u8Count++;
0071     }
0072 }
0073 void HCLCD_VidWriteNumber_8Bits(u32 Copy_u8Number)
0074 {
0075     u8 ARR_Digits[10];
0076     u8 LOC_Count=0;
0077     if(Copy_u8Number>0)
0078     {
0079         while(Copy_u8Number!=0)
0080         {
0081             ARR_Digits[LOC_Count]=Copy_u8Number%10;
0082             Copy_u8Number/=10;
0083             LOC_Count++;
0084         }
0085         for(s8 i=LOC_Count-1;i>=0;i--)

```

```

00086     {
00087         HCLCD_VidWriteChar_8Bits(ARR_Digits[i] +'0');
00088     }
00089 }
00090 else
00091 {
00092     HCLCD_VidWriteChar_8Bits('0');
00093 }
00094 }
00095 void HCLCD_VidSetPosition(u8 Copy_u8LineNumber , u8 Copy_u8PositionNumber)
00096 {
00097     if((Copy_u8LineNumber==HCLCD_LINE1)&&((Copy_u8PositionNumber>=0)&&(Copy_u8PositionNumber<16)))
00098     {
00099         HCLCD_VidWriteCommand_8Bits((LINE1_OFFSET_ADDRESS+Copy_u8PositionNumber));
00100     }
00101 else
00102     if((Copy_u8LineNumber==HCLCD_LINE2)&&((Copy_u8PositionNumber>=0)&&(Copy_u8PositionNumber<16)))
00103     {
00104         HCLCD_VidWriteCommand_8Bits((LINE2_OFFSET_ADDRESS+Copy_u8PositionNumber));
00105     }
00106 else
00107     /*Do Nothing*/
00108 }
00109 }
00110
00111 void HCLCD_VidWriteCommand_4Bits(u8 Copy_u8Command)
00112 {
00113     u8 LOC_u8CopyCommand;
00114     /*select Command register--> Write zero on Rs pin*/
00115     MDIO_Error_State_SetPinValue(RS,CONTROL_PORT,PIN_LOW);
00116     /*select Write mode--> Write zero on RW pin*/
00117     //MDIO_Error_State_SetPinValue(RW,CONTROL_PORT,PIN_LOW);
00118     /*Send Command*/
00119     LOC_u8CopyCommand=(Copy_u8Command&0x0F)«HCLCD_PINSTART;
00120     MDIO_Error_State_SetNippleValue(HCLCD_PINSTART,DATA_PORT,LOC_u8CopyCommand);
00121     /*Send Enable*/
00122     MDIO_Error_State_SetPinValue(E,CONTROL_PORT,PIN_HIGH);
00123     _delay_ms(2);
00124     MDIO_Error_State_SetPinValue(E,CONTROL_PORT,PIN_LOW);
00125 }
00126 void HCLCD_Vid4Bits_Init(void)
00127 {
00128     s8 LOC_u8PinCount;
00129     /*LCD Data and control port initialization*/
00130     for(LOC_u8PinCount=HCLCD_PINEND;LOC_u8PinCount>=HCLCD_PINSTART;LOC_u8PinCount--)
00131     {
00132         MDIO_Error_State_SetPinDirection(LOC_u8PinCount,DATA_PORT,PIN_OUTPUT);
00133     }
00134     MDIO_Error_State_SetPinDirection(RS,CONTROL_PORT,PIN_OUTPUT);
00135     //MDIO_Error_State_SetPinDirection(RW,CONTROL_PORT,PIN_OUTPUT);
00136     MDIO_Error_State_SetPinDirection(E,CONTROL_PORT,PIN_OUTPUT);
00137     /*wait to 30ms*/
00138     _delay_ms(30);
00139     /*send function set command*/
00140     HCLCD_VidWriteCommand_4Bits(HCLCD_FUNCTION_SET»4);
00141     HCLCD_VidWriteCommand_4Bits(HCLCD_FUNCTION_SET»4);
00142     HCLCD_VidWriteCommand_4Bits(HCLCD_FUNCTION_SET);
00143     _delay_ms(1);
00144     /*send Display on/off command*/
00145     HCLCD_VidWriteCommand_4Bits(HCLCD_DISPLAY_ON_OFF»4);
00146     HCLCD_VidWriteCommand_4Bits(HCLCD_DISPLAY_ON_OFF);
00147     _delay_ms(1);
00148     /*send Display Clear command*/
00149     HCLCD_VidWriteCommand_4Bits(DISPLAY_CLEAR»4);
00150     HCLCD_VidWriteCommand_4Bits(DISPLAY_CLEAR);
00151     _delay_ms(2);
00152     /*send Entry Mode set command*/
00153     HCLCD_VidWriteCommand_4Bits(HCLCD_ENTRY_MODE_SET»4);
00154     HCLCD_VidWriteCommand_4Bits(HCLCD_ENTRY_MODE_SET);
00155 }
00156 void HCLCD_VidWriteChar_4Bits(u8 Copy_u8Data)
00157 {
00158     u8 LOC_u8CopyData;
00159     /*select Data register--> Write One on Rs pin*/
00160     MDIO_Error_State_SetPinValue(RS,CONTROL_PORT,PIN_HIGH);
00161     /*select Write mode--> Write zero on RW pin*/
00162     //MDIO_Error_State_SetPinValue(RW,CONTROL_PORT,PIN_LOW);
00163     /*Send Data*/
00164     LOC_u8CopyData=(Copy_u8Data&0x0F)«HCLCD_PINSTART;
00165     MDIO_Error_State_SetNippleValue(HCLCD_PINSTART,DATA_PORT,LOC_u8CopyData);
00166     /*Send Enable*/
00167     MDIO_Error_State_SetPinValue(E,CONTROL_PORT,PIN_HIGH);
00168     _delay_ms(20);
00169     MDIO_Error_State_SetPinValue(E,CONTROL_PORT,PIN_LOW);
00170 }
00171 void HCLCD_VidSendChar_4Bits(u8 Copy_u8Data)

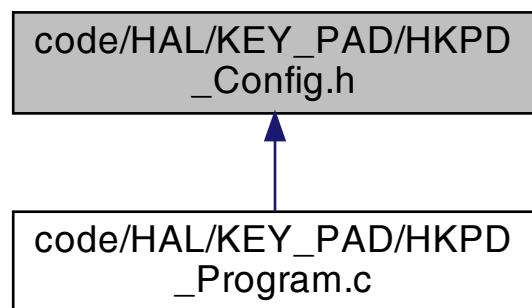
```

```

00172 {
00173     HCLCD_VidWriteChar_4Bits(Copy_u8Data»4);
00174     HCLCD_VidWriteChar_4Bits(Copy_u8Data);
00175 }
00176 void HCLCD_VidWriteString_4Bits(u8* PCopy_u8String)
00177 {
00178     u8 Loc_u8Count=0;
00179     while (PCopy_u8String[Loc_u8Count]!=NULL)
00180     {
00181         HCLCD_VidSendChar_4Bits(PCopy_u8String[Loc_u8Count]);
00182         Loc_u8Count++;
00183     }
00184 }
00185 void HCLCD_VidWriteNumber_4Bits(u32 Copy_u8Number)
00186 {
00187     u8 ARR_Digits[10];
00188     u8 LOC_Count=0;
00189     if(Copy_u8Number>0)
00190     {
00191         while(Copy_u8Number!=0)
00192         {
00193             ARR_Digits[LOC_Count]=Copy_u8Number%10;
00194             Copy_u8Number/=10;
00195             LOC_Count++;
00196         }
00197         for(s8 i=LOC_Count-1;i>=0;i--)
00198         {
00199             HCLCD_VidSendChar_4Bits(ARR_Digits[i]+‘0’);
00200         }
00201     }
00202     else
00203     {
00204         HCLCD_VidSendChar_4Bits(‘0’);
00205     }
00206 }
00207 void HCLCD_VidSetPosition_4BitsMode(u8 Copy_u8LineNumber , u8 Copy_u8PositionNumber)
00208 {
00209     if((Copy_u8LineNumber==HCLCD_LINE1)&&((Copy_u8PositionNumber>=0)&&(Copy_u8PositionNumber<16)))
00210     {
00211         HCLCD_VidWriteCommand_4Bits((LINE1_OFFSET_ADDRESS+Copy_u8PositionNumber)»4);
00212         HCLCD_VidWriteCommand_4Bits((LINE1_OFFSET_ADDRESS+Copy_u8PositionNumber));
00213     }
00214     else
00215     if((Copy_u8LineNumber==HCLCD_LINE2)&&((Copy_u8PositionNumber>=0)&&(Copy_u8PositionNumber<16)))
00216     {
00217         HCLCD_VidWriteCommand_4Bits((LINE2_OFFSET_ADDRESS+Copy_u8PositionNumber)»4);
00218         HCLCD_VidWriteCommand_4Bits((LINE2_OFFSET_ADDRESS+Copy_u8PositionNumber));
00219     }
00220     else
00221     /*Do Nothing*/
00222 }
00223 }
```

22.63 code/HAL/KEY_PAD/HKPD_Config.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define ROW_PORT MDIO_PORTB
- #define ROW_PIN0 PIN4
- #define ROW_PIN1 PIN5
- #define ROW_PIN2 PIN6
- #define ROW_PIN3 PIN7
- #define COL_PORT MDIO_PORTD
- #define COL_PIN0 PIN2
- #define COL_PIN1 PIN3
- #define COL_PIN2 PIN4
- #define COL_PIN3 PIN5
- #define COL_INIT 2
- #define COL_END 5
- #define ROW_INIT 4
- #define ROW_END 7

22.63.1 Macro Definition Documentation

22.63.1.1 COL_END #define COL_END 5

Definition at line 30 of file [HKPD_Config.h](#).

22.63.1.2 COL_INIT #define COL_INIT 2

Definition at line 29 of file [HKPD_Config.h](#).

22.63.1.3 COL_PIN0 #define COL_PIN0 PIN2

Definition at line 23 of file [HKPD_Config.h](#).

22.63.1.4 COL_PIN1 #define COL_PIN1 PIN3

Definition at line 24 of file [HKPD_Config.h](#).

22.63.1.5 COL_PIN2 #define COL_PIN2 PIN4

Definition at line 25 of file [HKPD_Config.h](#).

22.63.1.6 COL_PIN3 #define COL_PIN3 PIN5

Definition at line 26 of file [HKPD_Config.h](#).

22.63.1.7 COL_PORT #define COL_PORT MDIO_PORTD

Definition at line 21 of file [HKPD_Config.h](#).

22.63.1.8 ROW_END #define ROW_END 7

Definition at line 33 of file [HKPD_Config.h](#).

22.63.1.9 ROW_INIT #define ROW_INIT 4

Definition at line 32 of file [HKPD_Config.h](#).

22.63.1.10 ROW_PIN0 #define ROW_PIN0 PIN4

Definition at line 11 of file [HKPD_Config.h](#).

22.63.1.11 ROW_PIN1 #define ROW_PIN1 PIN5

Definition at line 12 of file [HKPD_Config.h](#).

22.63.1.12 ROW_PIN2 #define ROW_PIN2 PIN6

Definition at line 13 of file [HKPD_Config.h](#).

22.63.1.13 ROW_PIN3 #define ROW_PIN3 PIN7

Definition at line 14 of file [HKPD_Config.h](#).

22.63.1.14 ROW_PORT #define ROW_PORT MDIO_PORTB

Definition at line 9 of file [HKPD_Config.h](#).

22.64 HKPD_Config.h

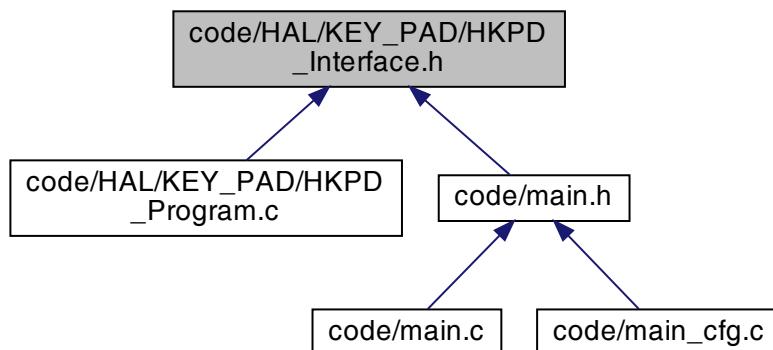
```

00001 #ifndef HKPD_CONFIG_H_
00002 #define HKPD_CONFIG_H_
00003
00004 /*Row Port Options
00005 * 1- MDIO_PORTA
00006 * 2- MDIO_PORTB
00007 * 3- MDIO_PORTC
00008 * 4- MDIO_PORTD*/
00009 #define ROW_PORT      MDIO_PORTB
00010 /*Row Pins Options: From PIN0 To PIN7*/
00011 #define ROW_PIN0      PIN4
00012 #define ROW_PIN1      PIN5
00013 #define ROW_PIN2      PIN6
00014 #define ROW_PIN3      PIN7
00015
00016 /*COL Port Options
00017 * 1- MDIO_PORTA
00018 * 2- MDIO_PORTB
00019 * 3- MDIO_PORTC
00020 * 4- MDIO_PORTD*/
00021 #define COL_PORT      MDIO_PORTD
00022 /*Row Pins Options: From PIN0 To PIN7*/
00023 #define COL_PIN0      PIN2
00024 #define COL_PIN1      PIN3
00025 #define COL_PIN2      PIN4
00026 #define COL_PIN3      PIN5
00027
00028 /*COL INIT is Start Col pin and COL End is last col pin +1 */
00029 #define COL_INIT      2
00030 #define COL_END       5
00031 /*row INIT is Start row pin and row End is last row pin +1 */
00032 #define ROW_INIT      4
00033 #define ROW_END       7
00034
00035 #endif

```

22.65 code/HAL/KEY_PAD/HKPD_Interface.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- `#define NOT_PRESSED 0`

Functions

- void [HKPD_VidInit](#) (void)
- u8 [HKPD_U8GetKeyPressed](#) (void)

22.65.1 Macro Definition Documentation

22.65.1.1 NOT_PRESSED #define NOT_PRESSED 0

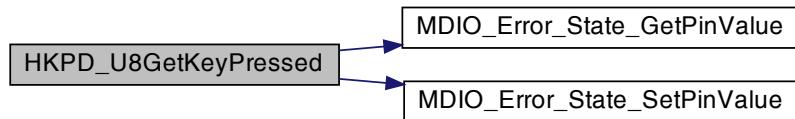
Definition at line 4 of file [HKPD_Interface.h](#).

22.65.2 Function Documentation

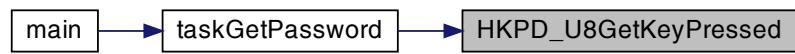
22.65.2.1 HKPD_U8GetKeyPressed() u8 HKPD_U8GetKeyPressed (void)

Definition at line 61 of file [HKPD_Program.c](#).

Here is the call graph for this function:



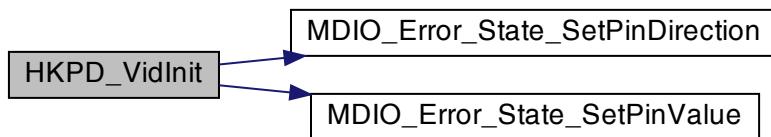
Here is the caller graph for this function:



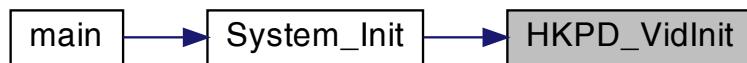
22.65.2.2 HKPD_VidInit() void HKPD_VidInit (void)

Definition at line 37 of file [HKPD_Program.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.66 HKPD_Interface.h

```

00001 #ifndef HKPD_INTERFACE_H_
00002 #define HKPD_INTERFACE_H_
00003
00004 #define NOT_PRESSED 0
00005
00006 /*KPD INIT*/
00007 void HKPD_VidInit(void);
00008 /*KPD Get pressed*/
00009 u8 HKPD_U8GetKeyPressed(void);
00010
00011 #endif
00012
  
```

22.67 code/HAL/KEY_PAD/HKPD_Private.h File Reference

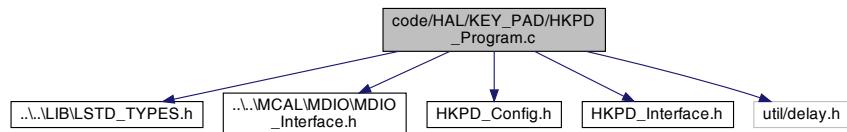
22.68 HKPD_Private.h

22.69 code/HAL/KEY_PAD/HKPD_Program.c File Reference

```

#include "..\..\LIB\LSTD_TYPES.h"
#include "..\..\MCAL\MDIO\MDIO_Interface.h"
#include "HKPD_Config.h"
#include "HKPD_Interface.h"
  
```

```
#include <util/delay.h>
Include dependency graph for HKPD_Program.c:
```



Functions

- void `HKPD_VidInit` (void)
- u8 `HKPD_U8GetKeyPressed` (void)

Variables

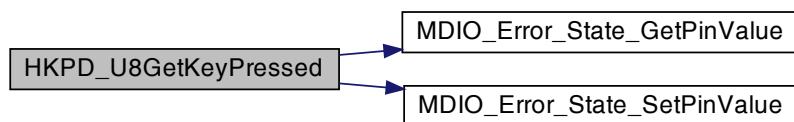
- const u8 `KPD_u8SwitchVal` [4][4]

22.69.1 Function Documentation

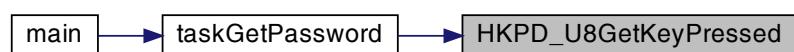
22.69.1.1 `HKPD_U8GetKeyPressed()` u8 `HKPD_U8GetKeyPressed` (void)

Definition at line 61 of file `HKPD_Program.c`.

Here is the call graph for this function:



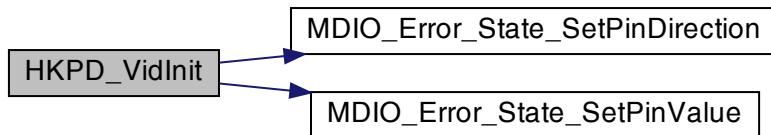
Here is the caller graph for this function:



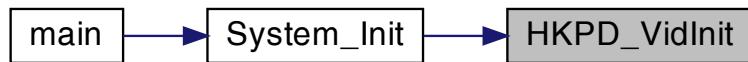
22.69.1.2 HKPD_VidInit() void HKPD_VidInit (void)

Definition at line 37 of file [HKPD_Program.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.69.2 Variable Documentation

22.69.2.1 KPD_u8SwitchVal const u8 KPD_u8SwitchVal[4][4]

Initial value:

```
=
{
    {
        '7',
        '4',
        '1',
        'C'
    },
    {
        '8',
        '5',
        '2',
        '0'
    },
    {
        '9',
        '6',
        '3',
        '='
    },
    {
        '/',
        '*',
        '-',
        '+'
    }
}
```

Definition at line 9 of file [HKPD_Program.c](#).

22.70 HKPD_Program.c

```

00001
00002
00003 #include"..\..\LIB\LSTD_TYPES.h"
00004 #include"..\..\MCAL\MDIO\MDIO_Interface.h"
00005 #include"HKPD_Config.h"
00006 #include"HKPD_Interface.h"
00007 #include<util/delay.h>
00008
00009 const u8 KPD_u8SwitchVal[4][4]=
00010 {
00011     {
00012         '7',
00013         '4',
00014         '1',
00015         'C'
00016     },
00017     {
00018         '8',
00019         '5',
00020         '2',
00021         '0'
00022     },
00023     {
00024         '9',
00025         '6',
00026         '3',
00027         '='
00028     },
00029     {
00030         '/',
00031         '*',
00032         '-',
00033         '+'
00034     },
00035 };
00036 /*KPD INIT*/
00037 void HKPD_VidInit(void)
00038 {
00039     /* Set Row Pins as an Input */
00040     MDIO_Error_State_SetPinDirection(ROW_PIN0, ROW_PORT, PIN_INPUT);
00041     MDIO_Error_State_SetPinDirection(ROW_PIN1, ROW_PORT, PIN_INPUT);
00042     MDIO_Error_State_SetPinDirection(ROW_PIN2, ROW_PORT, PIN_INPUT);
00043     MDIO_Error_State_SetPinDirection(ROW_PIN3, ROW_PORT, PIN_INPUT);
00044     /* Set Column Pins as an Output */
00045     MDIO_Error_State_SetPinDirection(COL_PIN0, COL_PORT, PIN_OUTPUT);
00046     MDIO_Error_State_SetPinDirection(COL_PIN1, COL_PORT, PIN_OUTPUT);
00047     MDIO_Error_State_SetPinDirection(COL_PIN2, COL_PORT, PIN_OUTPUT);
00048     MDIO_Error_State_SetPinDirection(COL_PIN3, COL_PORT, PIN_OUTPUT);
00049     /* Active Pull Up Resistor For Row Pins */
00050     MDIO_Error_State_SetPinValue(ROW_PIN0, ROW_PORT, PIN_HIGH);
00051     MDIO_Error_State_SetPinValue(ROW_PIN1, ROW_PORT, PIN_HIGH);
00052     MDIO_Error_State_SetPinValue(ROW_PIN2, ROW_PORT, PIN_HIGH);
00053     MDIO_Error_State_SetPinValue(ROW_PIN3, ROW_PORT, PIN_HIGH);
00054     /* Initialize 4 Column Pins By Ones */
00055     MDIO_Error_State_SetPinValue(COL_PIN0, COL_PORT, PIN_HIGH);
00056     MDIO_Error_State_SetPinValue(COL_PIN1, COL_PORT, PIN_HIGH);
00057     MDIO_Error_State_SetPinValue(COL_PIN2, COL_PORT, PIN_HIGH);
00058     MDIO_Error_State_SetPinValue(COL_PIN3, COL_PORT, PIN_HIGH);
00059 }
00060 /*KPD Get pressed*/
00061 u8 HKPD_U8GetKeyPressed(void)
00062 {
00063     u8 LOC_U8RowCount;
00064     u8 LOC_U8ColCount;
00065     /*Initialize the switch status to NOT PRESSED*/
00066     u8 LOC_U8ReturnValue=NOT_PRESSED;
00067     u8 LOC_U8PinState;
00068     /*Looping on columns of the keypad*/
00069     for(LOC_U8ColCount=COL_INIT;LOC_U8ColCount<=COL_END;LOC_U8ColCount++)
00070     {
00071         /*Active the Column */
00072         MDIO_Error_State_SetPinValue(LOC_U8ColCount, COL_PORT, PIN_LOW);
00073         /*Loop to read the all row pins*/
00074         for(LOC_U8RowCount=ROW_INIT;LOC_U8RowCount<=ROW_END;LOC_U8RowCount++)
00075         {
00076             /*check the status of the switch*/
00077             MDIO_Error_State_GetPinValue(LOC_U8RowCount, ROW_PORT, &LOC_U8PinState);
00078             if(LOC_U8PinState==0)
00079             {
00080                 /*Get the Value of the current pressed switch*/
00081                 LOC_U8ReturnValue=KPD_u8SwitchVal[LOC_U8ColCount-COL_INIT][LOC_U8RowCount-ROW_INIT];
00082                 /*wait until the switch is released(Single Press)*/
00083                 while(LOC_U8PinState==0)
00084                 {
00085                     MDIO_Error_State_GetPinValue(LOC_U8RowCount, ROW_PORT, &LOC_U8PinState);

```

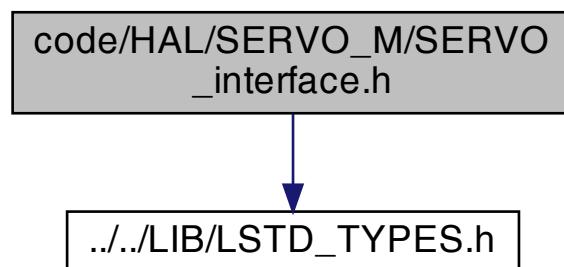
```
00086          }
00087          /*Delay To avoid Bouncing*/
00088          _delay_ms(10);
00089      }
00090      else
00091      {
00092          /*Do Nothing*/
00093      }
00094  }
00095  /*Deactivate the Column*/
00096  MDIO_Error_State_SetPinValue(LOC_U8ColCount,COL_PORT,PIN_HIGH);
00097 }
00098 return LOC_U8ReturnValue;
00099 }
```

22.71 code/HAL/SERVO_M/SERVO_Config.h File Reference

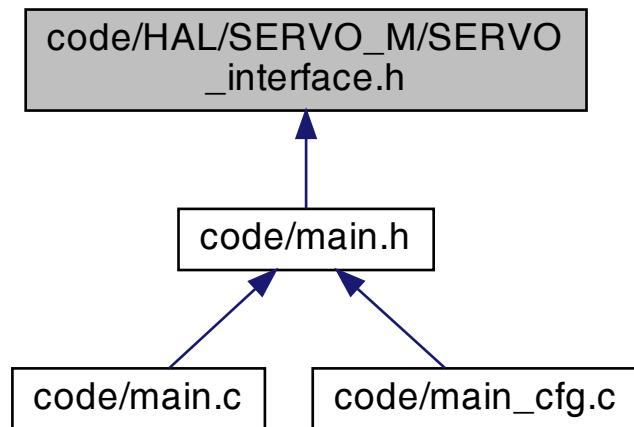
22.72 SERVO_Config.h

22.73 code/HAL/SERVO_M/SERVO_interface.h File Reference

```
#include "../../LIB/LSTD_TYPES.h"
Include dependency graph for SERVO_interface.h:
```



This graph shows which files directly or indirectly include this file:



Functions

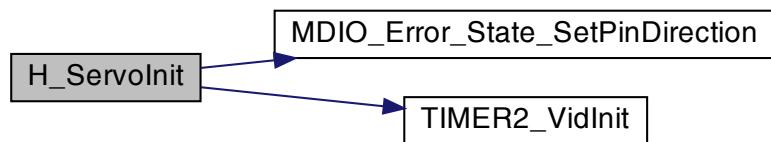
- void `H_ServoInit` (void)
- void `H_ServoSetAngle` (u16)
- void `H_ServoStart` (void)
- void `H_ServoStop` (void)

22.73.1 Function Documentation

22.73.1.1 H_ServoInit() `void H_ServoInit (`
 `void)`

Definition at line 7 of file `SERVO_Program.c`.

Here is the call graph for this function:



22.73.1.2 H_ServoSetAngle() void H_ServoSetAngle (
 u16_local_angle)

Definition at line 13 of file [SERVO_Program.c](#).

Here is the call graph for this function:



22.73.1.3 H_ServoStart() void H_ServoStart (
 void)

Definition at line 18 of file [SERVO_Program.c](#).

Here is the call graph for this function:



22.73.1.4 H_ServoStop() void H_ServoStop (
 void)

Definition at line 22 of file [SERVO_Program.c](#).

Here is the call graph for this function:



22.74 SERVO_interface.h

```

00001 #ifndef SERVO_H_
00002 #define SERVO_H_
00003
00004 #include"../../LIB/LSTD_TYPES.h"
00005
00006 void H_ServoInit(void);
00007 void H_ServoSetAngle(u16);
00008 void H_ServoStart(void);
00009 void H_ServoStop(void);
00010
00011 #endif /* SERVO_H_ */

```

22.75 code/HAL/SERVO_M/SERVO_Private.h File Reference

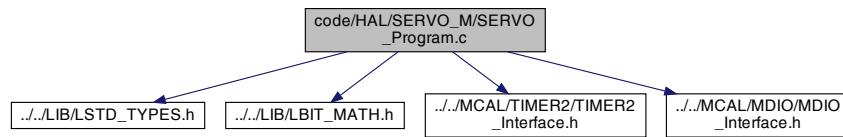
22.76 SERVO_Private.h

22.77 code/HAL/SERVO_M/SERVO_Program.c File Reference

```

#include "../../LIB/LSTD_TYPES.h"
#include "../../LIB/LBIT_MATH.h"
#include "../../MCAL/TIMER2/TIMER2_Interface.h"
#include "../../MCAL/MDIO/MDIO_Interface.h"
Include dependency graph for SERVO_Program.c:

```



Functions

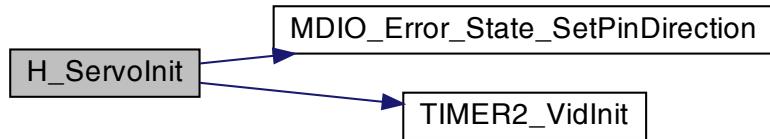
- void H_ServoInit (void)
- void H_ServoSetAngle (u16 u16_local_angle)
- void H_ServoStart (void)
- void H_ServoStop (void)

22.77.1 Function Documentation

```
22.77.1.1 H_ServoInit() void H_ServoInit (
    void )
```

Definition at line 7 of file [SERVO_Program.c](#).

Here is the call graph for this function:



```
22.77.1.2 H_ServoSetAngle() void H_ServoSetAngle (
    u16 u16_local_angle )
```

Definition at line 13 of file [SERVO_Program.c](#).

Here is the call graph for this function:



```
22.77.1.3 H_ServoStart() void H_ServoStart (
    void )
```

Definition at line 18 of file [SERVO_Program.c](#).

Here is the call graph for this function:



```
22.77.1.4 H_ServoStop() void H_ServoStop (
    void )
```

Definition at line 22 of file [SERVO_Program.c](#).

Here is the call graph for this function:



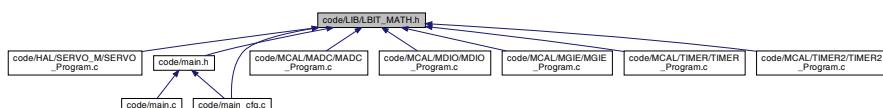
22.78 SERVO_Program.c

```

00001 #include"../../LIB/LSTD_TYPES.h"
00002 #include"../../LIB/LBIT_MATH.h"
00003 #include"../../MCAL/TIMER2/TIMER2_Interface.h"
00004 #include"../../MCAL/MDIO/MDIO_Interface.h"
00005
00006
00007 void H_ServoInit(void)
00008 {
00009     MDIO_Error_State_SetPinDirection(PIN7, MDIO_PORTD, PIN_OUTPUT);
00010     TIMER2_VidInit();
00011     //M_Pwm2Init();
00012 }
00013 void H_ServoSetAngle(u16 u16_local_angle)
00014 {
00015     f64 f64_local_DutyCycle = ( ( 5 * u16_local_angle ) / 180 ) + 5 );
00016     M_Pwm2SetDutyCycle(f64_local_DutyCycle);
00017 }
00018 void H_ServoStart(void)
00019 {
00020     M_Pwm2Start();
00021 }
00022 void H_ServoStop(void)
00023 {
00024     M_Pwm2Stop();
00025 }
```

22.79 code/LIB/LBIT_MATH.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- `#define SET_BIT(VAR, BIT_NUM) VAR|=(1<<BIT_NUM)`
- `#define CLR_BIT(VAR, BIT_NUM) VAR&=(~(1<<BIT_NUM))`
- `#define TOGGLE_BIT(VAR, BIT_NUM) VAR^=(1<<BIT_NUM)`
- `#define GET_BIT(VAR, BIT_NUM) ((VAR>>BIT_NUM)&1)`

22.79.1 Macro Definition Documentation

22.79.1.1 CLR_BIT #define CLR_BIT(

BIT_NUM) VAR&= ($\sim (1 << \text{BIT_NUM})$)

Definition at line 2 of file [LBIT_MATH.h](#).

22.79.1.2 GET_BIT #define GET_BIT(

VAR,
BIT_NUM) ((*VAR*>>*BIT_NUM*) & 1)

Definition at line 4 of file [LBIT_MATH.h](#).

22.79.1.3 SET_BIT #define SET_BIT(

VAR,
BIT_NUM) VAR|=(1<<*BIT_NUM*)

Definition at line 1 of file [LBIT_MATH.h](#).

22.79.1.4 TOGGLE BIT #define TOGGLE_BIT(

VAR,
BIT_NUM) VAR^=(1<<*BIT_NUM*)

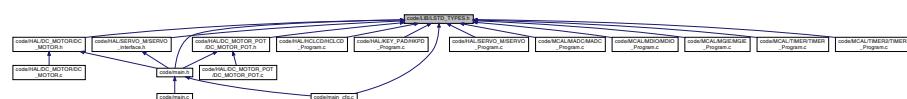
Definition at line 3 of file [LBIT_MATH.h](#).

22.80 LBIT MATH.h

```
00001 #define SET_BIT(VAR,BIT_NUM)           VAR|=(1<<BIT_NUM)//VAR=VAR|(1<<BIT_NUM)
00002 #define CLR_BIT(VAR,BIT_NUM)           VAR&=~(1<<BIT_NUM)
00003 #define TOGGLE_BIT(VAR,BIT_NUM)        VAR^=(1<<BIT_NUM)
00004 #define GET_BIT(VAR,BIT_NUM)            ((VAR>>BIT_NUM)&1)
00005
```

22.81 code/LIB/LSTD_TYPES.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- `#define NULL_POINTER 1`
- `#define NULL 0`

Typedefs

- `typedef unsigned char u8`
- `typedef signed char s8`
- `typedef unsigned short int u16`
- `typedef signed short int s16`
- `typedef unsigned long int u32`
- `typedef signed long int s32`
- `typedef unsigned long long int u64`
- `typedef signed long long int s64`
- `typedef float f32`
- `typedef double f64`

Enumerations

- `enum Error_State { NOK =0 , OK }`

22.81.1 Macro Definition Documentation

22.81.1.1 `NULL` `#define NULL 0`

Definition at line 11 of file [LSTD_TYPES.h](#).

22.81.1.2 `NULL_POINTER` `#define NULL_POINTER 1`

Definition at line 10 of file [LSTD_TYPES.h](#).

22.81.2 Typedef Documentation

22.81.2.1 `f32` `typedef float f32`

Definition at line 25 of file [LSTD_TYPES.h](#).

22.81.2.2 f64 `typedef double f64`

Definition at line 26 of file [LSTD_TYPES.h](#).

22.81.2.3 s16 `typedef signed short int s16`

Definition at line 17 of file [LSTD_TYPES.h](#).

22.81.2.4 s32 `typedef signed long int s32`

Definition at line 20 of file [LSTD_TYPES.h](#).

22.81.2.5 s64 `typedef signed long long int s64`

Definition at line 23 of file [LSTD_TYPES.h](#).

22.81.2.6 s8 `typedef signed char s8`

Definition at line 14 of file [LSTD_TYPES.h](#).

22.81.2.7 u16 `typedef unsigned short int u16`

Definition at line 16 of file [LSTD_TYPES.h](#).

22.81.2.8 u32 `typedef unsigned long int u32`

Definition at line 19 of file [LSTD_TYPES.h](#).

22.81.2.9 u64 `typedef unsigned long long int u64`

Definition at line 22 of file [LSTD_TYPES.h](#).

22.81.2.10 u8 `typedef unsigned char u8`

Definition at line 13 of file [LSTD_TYPES.h](#).

22.81.3 Enumeration Type Documentation

22.81.3.1 Error_State `enum Error_State`

Enumerator

NOK	
OK	

Definition at line 5 of file [LSTD_TYPES.h](#).

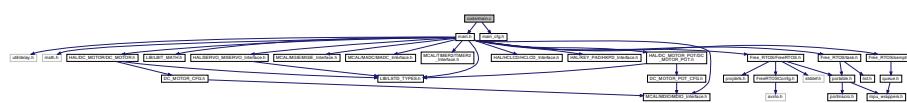
22.82 LSTD_TYPES.h

```
00001
00002 #ifndef STD_TYPES_H_
00003 #define STD_TYPES_H_
00004
00005 typedef enum{
00006     NOK=0,
00007     OK
00008 }Error_State;
00009
00010 #define NULL_POINTER 1
00011 #define NULL 0
00012
00013 typedef unsigned char u8;
00014 typedef signed char s8;
00015
00016 typedef unsigned short int u16;
00017 typedef signed short int s16;
00018
00019 typedef unsigned long int u32;
00020 typedef signed long int s32;
00021
00022 typedef unsigned long long int u64;
00023 typedef signed long long int s64;
00024
00025 typedef float f32;
00026 typedef double f64;
00027
00028
00029 #endif /* STD TYPES H */
```

22.83 code/main.c File Reference

Smart Home Security System.

```
#include "main.h"
#include "main_cfg.h"
Include dependency graph for main.c:
```



Functions

- int **main** (void)
 - void **taskGetPassword** (void *pv)
Task to get the password from the keypad.
 - void **taskDisplaySystemState** (void *pv)
Task to display the system's data.
 - void **taskReadSensors** (void *pv)
Task to read the sensors.

- void **taskControlGasLevel** (void *pv)
Task to control the gas level.
- void **taskControlTemperature** (void *pv)
Task to control the temperature.
- void **ADC_SetNotification** (void)
- void **System_Init** (u8 u8_AdcChannelToRead)
Initialize system peripherals.
- void **displayDoorState** (void)
Display the current state of the door: open or closed.
- void **displayTemperatureValue** (void)
Display the current temperature of the room.
- void **displayFireState** (void)
Checking if the fire alarm is on or not.
- void **displayEntryPasswordSecurity** (void)
Display the security of the house (Password).
- u8 **comparePasswords** (const u8 *const pass1, const u8 *const pass2, const u8 length)
Compare the password entered by the user with the default password. @retaur If the password is correct, return 1. Otherwise, return 0.
- void **controlDoor** (void)
It controls the door of the house. Open the door if the user enters the correct password. Otherwise, close the door.

Variables

- xSemaphoreHandle **ADC_Semaphore**
*Semaphor used by the ADC to indicate that the conversion is done. It is given by the ADC ISR, and taken inside the **taskReadSensors()**.*
- u8 u8_AdcChannelToRead = 0
*The ADC channel to be read. It toggles between the channels of the gas sensor and the temperature sensor. So, each time the **taskReadSensors()** starts a new conversion, it reads the next channel.*

22.83.1 Detailed Description

Smart Home Security System.

Author

Silence is the key #13 - Team:

- Mahmoud Karam Emara (ma.karam272@gmail.com)
- Ahmed Abdelgawad Kamal (ahmedabdelgawad234@gmail.com)
- Mina Ghobrial Abdulla (menaghobrial98@gmail.com)
- Hossam Mostafa Abd El-Aziz (hossam11015@gmail.com)
- Ahmed Mohamed Mannaa (ahmed.mg.manna3@gmail.com)

This is the main file of the system. It consists of the following functions:

- Gas Sensor to sense the gas level in the house.
- Temperature Sensor to sense the temperature in the house.
- Keypad to enter the password to open the door.
- Motor to open/close the door when the password is correct.
- Buzzer to alert the user when the system is in alarm mode:
 - The password is incorrect for 3 times.
 - The gas level is above the critical level.
- LCD to display the system's data (temperature, door status, gas level).

Version

1.0.0

Date

2022-06-28

Copyright

Copyright (c) 2022

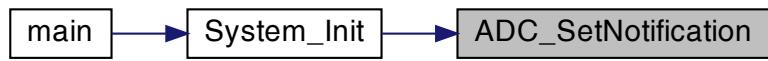
Definition in file [main.c](#).

22.83.2 Function Documentation

22.83.2.1 ADC_SetNotification() `void ADC_SetNotification (void)`

Definition at line [271](#) of file [main.c](#).

Here is the caller graph for this function:

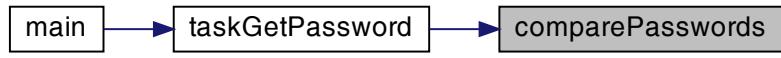


22.83.2.2 comparePasswords() `u8 comparePasswords (const u8 *const pass1, const u8 *const pass2, const u8 length)`

Compare the password entered by the user with the default password. @return If the password is correct, return 1. Otherwise, return 0.

Definition at line [449](#) of file [main.c](#).

Here is the caller graph for this function:



```
22.83.2.3 controlDoor() void controlDoor (
    void )
```

It controls the door of the house. Open the door if the user enters the correct password. Otherwise, close the door.

Definition at line [466](#) of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
22.83.2.4 displayDoorState() void displayDoorState (
    void )
```

Display the current state of the door: open or closed.

If the door is open, "D: Open" is displayed. If the door is closed, "D: Close" is displayed.

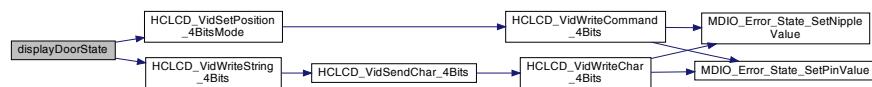
Note

This function writes to the LCD at the line 1, position 0, and uses 8 positions to write the string. So, avoid writing to the first 8 positions of the LCD in line 1.

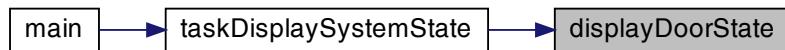
Checking if the door is open or not, and writing the string "Open" or "Close" to the LCD.

Definition at line [323](#) of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.83.2.5 displayEntryPasswordSecurity() `void displayEntryPasswordSecurity (void)`

Display the security of the house (Password).

If the password is:

- Correct: "AMAN(x)" is written to the LCD, where x is the number of attempts.
- Wrong: "7RAMY(0)" is written to the LCD, where 0 denotes the number of attempts left is 0. And the buzzer is on for 3 seconds. We assume that the police arrives during the 3 seconds.

Note

This function writes to the LCD at the line 2, position 6, and uses 4 positions to write the string. So, avoid writing to the last 4 positions of the LCD in line 2.

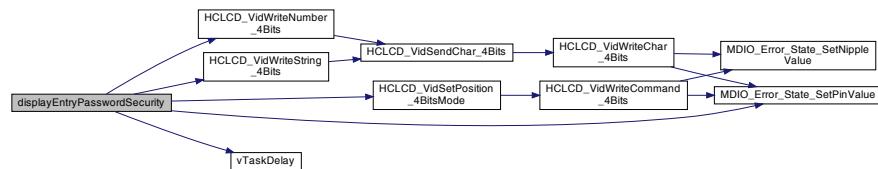
If the password is correct, it will display "AMAN(x)" where x is the number of attempts. If the password is wrong, it will display "7RAMY(0)" where 0 denotes the number of attempts left is 0. And the buzzer is on for 3 seconds. We assume that the police arrives during the 3 seconds.

< Turn off the buzzer only if the fire alarm is off. This is to prevent the buzzer from turning off when the fire alarm is on.

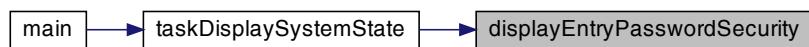
< buzzer flag is off.

Definition at line 410 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
22.83.2.6 displayFireState() void displayFireState (
    void )
```

Checking if the fire alarm is on or not.

If the fire alarm is on, the string "FIRE" is written to the LCD, and the buzzer is on. Otherwise, the string "NoFire" is written to the LCD.

Note

This function writes to the LCD at the line 2, position 0, and uses 6 positions to write the string. So, avoid writing to the first 6 positions of the LCD in line 2.

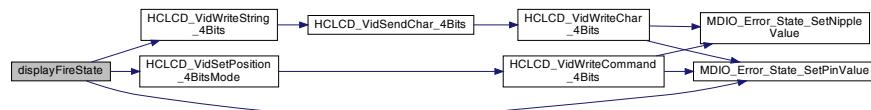
If the fire alarm is on, it will turn on the buzzer and display "FIRE" on the LCD. If it is off, it will turn off the buzzer and display "NoFire" on the LCD.

< Turn on the buzzer.

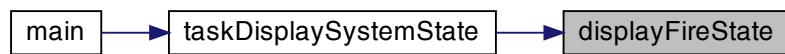
< Turn off the buzzer.

Definition at line 375 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
22.83.2.7 displayTemperatureValue() void displayTemperatureValue (
    void )
```

Display the current temperature of the room.

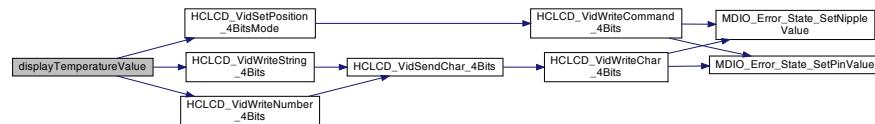
The current temperature is displayed in the LCD. "T: xx°C" is displayed.

Note

This function writes to the LCD at the line 1, position 10, and uses 7 positions to write the string. So, avoid writing to the last 7 positions of the LCD in line 1.

Definition at line 348 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.83.2.8 main()

```

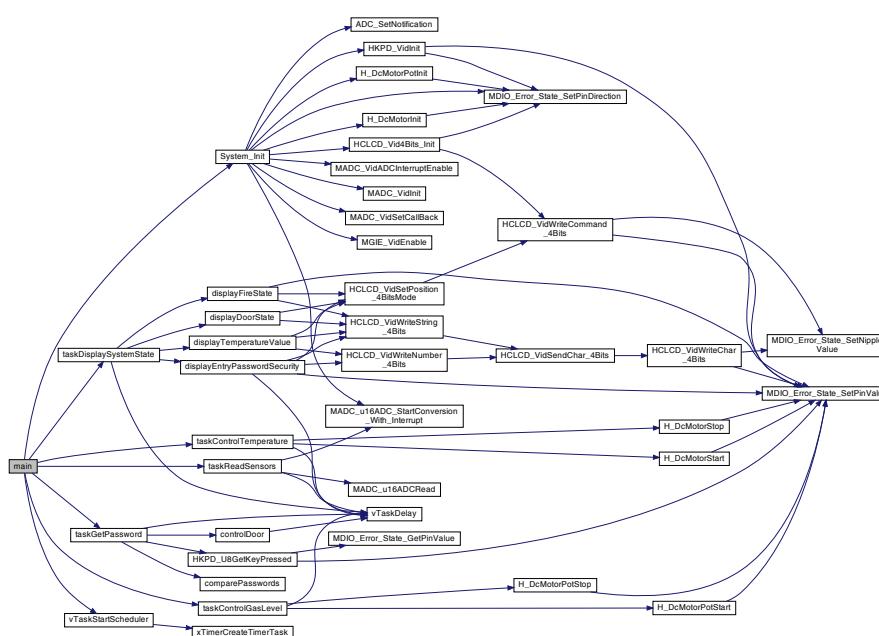
int main (
    void
)
  
```

< Initial ADC channel to be read

< Initialize the system

Definition at line 51 of file [main.c](#).

Here is the call graph for this function:



22.83.2.9 System_Init() void System_Init (
 8 u8_AdcChannelToRead)

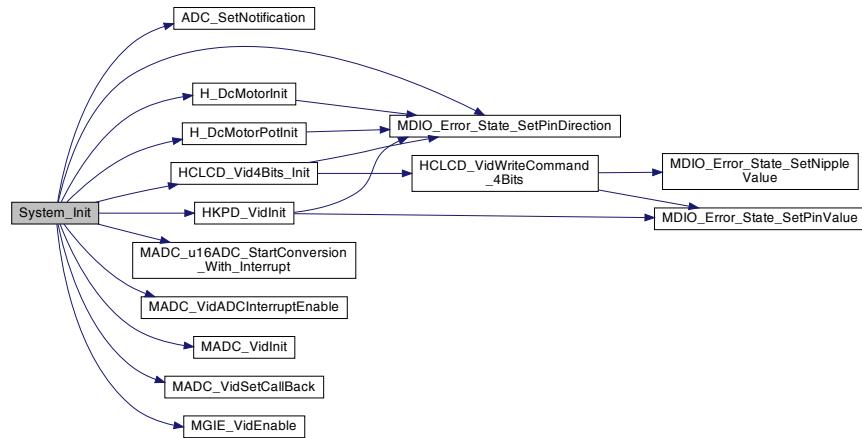
Initialize system peripherals.

Parameters

u8_AdcChannelToRead	The initial ADC channel to be read at the beginning.
---------------------	--

Definition at line 280 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.83.2.10 taskControlGasLevel() void taskControlGasLevel (
 void * pv)

Task to control the gas level.

This task is responsible for controlling the gas level. It checks if the gas level is above the threshold and if it is:

- Above: Open the window, and raise the flag to alert the user (buzzer).
- Below: Close the window, and downset the flag of the alert.

Note

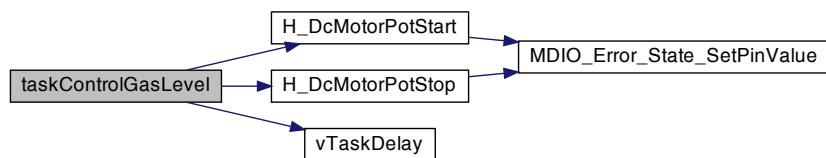
The gas level is read from the ADC using a potentiometer. It is converted to a percentage of the full scale.

Parameters

<i>pv</i>	RTOS task parameter. Not used.
-----------	--------------------------------

Definition at line 221 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.83.2.11 taskControlTemperature()

```
void taskControlTemperature (
    void * pv )
```

Task to control the temperature.

This task is responsible for controlling the temperature. It checks if the temperature is above the threshold and if it is:

- Above: Open the fan.
- Below: Stop the fan.

Note

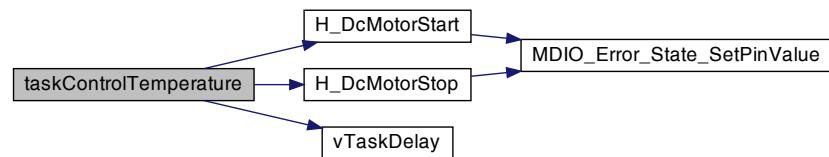
Temperature is measured using the LM35 sensor. The LM35 sensor is connected to the ADC. The sensor gives 10mV for every degree Celsius. So, you can measure the temperature in degrees Celsius by dividing the ADC voltage value by 10mV.

Parameters

<i>pv</i>	RTOS task parameter. Not used.
-----------	--------------------------------

Definition at line 249 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.83.2.12 taskDisplaySystemState() `void taskDisplaySystemState (void * pv)`

Task to display the system's data.

This task is responsible for displaying the system's data. It displays:

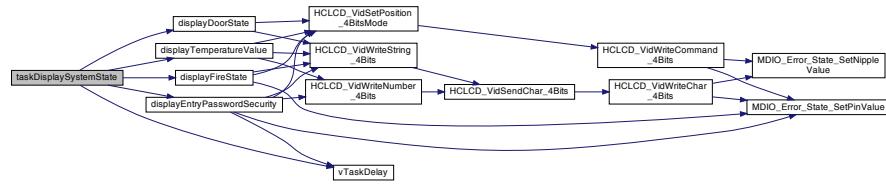
- The temperature.
- The door status.
- The fire alarm status.
- The door security status.

Parameters

<i>pv</i>	RTOS task parameter. Not used.
-----------	--------------------------------

Definition at line 156 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.83.2.13 taskGetPassword() `void taskGetPassword (void * pv)`

Task to get the password from the keypad.

This task is responsible for getting the password from the keypad. If the password is:

- Correct: raise the flag to open the door.
- Incorrect: decrease the number of attempts to open the door. If the number of attempts is 0, raise the flag to alert the user (buzzer).

Parameters

<code>pv</code>	RTOS task parameter. Not used.
-----------------	--------------------------------

< Getting the key pressed from the keyboard.
 < Check if the user pressed a key
 < If the user pressed a key, then save it to the enteredPassword array
 < Increment the index of the enteredPassword array

Checking if the password is fully entered.

If the password is:

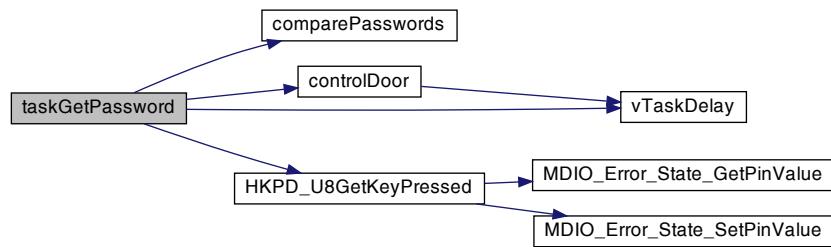
- Correct: system.flag.correctPassword is set to 1.
- Wrong: system.flag.correctPassword is set to 0 and system.password.attempts is decremented.

Note

If system.password.attempts is 0, the system.flag.buzzer is set to 1.

Definition at line 94 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.83.2.14 taskReadSensors() void taskReadSensors (void * *pv*)

Task to read the sensors.

This task is responsible for reading the sensors. It reads the gas level and the temperature. It uses the ADC in Non-Blocking mode, to enhances the responsiveness of the system.

Parameters

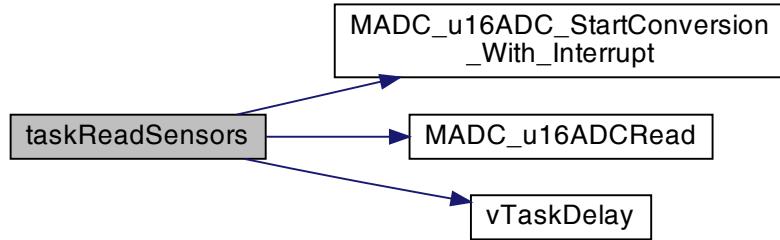
<i>pv</i>	RTOS task parameter. Not used.
-----------	--------------------------------

< Check if the ADC semaphore is available

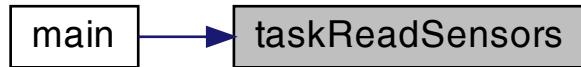
Read temperature sensor or gas sensor depending on the system.inputs.gas.u8_AdcChannel, then swap the ADC channel to read the next sensor in the next iteration.

Definition at line 175 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.83.3 Variable Documentation

22.83.3.1 ADC_Semaphore `xSemaphoreHandle` ADC_Semaphore

Semaphor used by the ADC to indicate that the conversion is done. It is given by the ADC ISR, and taken inside the [taskReadSensors\(\)](#).

Definition at line 37 of file [main.c](#).

22.83.3.2 u8_AdcChannelToRead `u8 u8_AdcChannelToRead = 0`

The ADC channel to be read. It toggles between the channels of the gas sensor and the temperature sensor. So, each time the [taskReadSensors\(\)](#) starts a new conversion, it reads the next channel.

Definition at line 44 of file [main.c](#).

22.84 main.c

```

00136         if(system.password.attempts == 0) {
00137             system.password.attempts = system.password.maxAttempts;
00138         }
00139         system.flag.correctPassword = 0;
00140         i = 0;
00141     }
00142 }
00143 vTaskDelay(20);
00144 }
00145 }
00146
00147
00156 void taskDisplaySystemState(void * pv) {
00157
00158     while(1) {
00159         displayDoorState();
00160         displayTemperatureValue();
00161         displayFireState();
00162         displayEntryPasswordSecurity();
00163
00164         vTaskDelay(200);
00165     }
00166 }
00167
00168
00175 void taskReadSensors(void * pv) {
00176     u8 LOC_u8SemState = 0;
00177
00178     while(1) {
00179         /* Trying to take the semaphore, with 5 ticks timeout */
00180         LOC_u8SemState = xSemaphoreTake(ADC_Semaphore, 5);
00181
00182         if(pdPASS == LOC_u8SemState) {
00183
00184             if(u8_AdcChannelToRead == system.inputs.gas.u8_AdcChannel) {
00185                 system.inputs.gas.ul6_CurrentValueInBinary = MADC_ul6ADCRead();
00186                 u8_AdcChannelToRead = system.inputs.temperature.u8_AdcChannel;
00187             } else if(u8_AdcChannelToRead == system.inputs.temperature.u8_AdcChannel) {
00188                 system.inputs.temperature.ul6_CurrentValueInBinary = MADC_ul6ADCRead();
00189                 u8_AdcChannelToRead = system.inputs.gas.u8_AdcChannel;
00190             } else {
00191                 /* MISRA C */
00192             }
00193
00194             MADC_ul6ADC_StartConversion_With_Interrupt(u8_AdcChannelToRead);
00195         } else {
00196             /*Do Nothing*/
00197         }
00198
00199         vTaskDelay(10);
00200     }
00201 }
00202
00221 void taskControlGasLevel(void * pv) {
00222     while(1) {
00223         system.inputs.gas.u8_CurrentValue = (u8)((u32)system.inputs.gas.ul6_CurrentValueInBinary *
00224             system.inputs.gas.u8_MaxValue) / 1024.0f);
00225
00226         if(system.inputs.gas.u8_CurrentValue >= system.inputs.gas.u8_CriticalValue) {
00227             H_DcMotorPotStart(CLK_W);
00228             system.flag.fire = 1;
00229         } else {
00230             system.flag.fire = 0;
00231             H_DcMotorPotStop();
00232         }
00233
00234         vTaskDelay(15);
00235     }
00236
00237
00249 void taskControlTemperature(void * pv) {
00250     while(1) {
00251         system.inputs.temperature.u8_CurrentValue =
00252             (u8)((system.inputs.temperature.ul6_CurrentValueInBinary * 500.0f) / 1024.0f);
00253
00254         system.inputs.temperature.u8_CurrentValue = round(system.inputs.temperature.u8_CurrentValue);
00255
00256         if(system.inputs.temperature.u8_CurrentValue > system.inputs.temperature.u8_CriticalValue) {
00257             H_DcMotorStart(CLK_W);
00258         } else {
00259             H_DcMotorStop();
00260         }
00261
00262     }
00263 }
```

```

00261     vTaskDelay(15);
00262 }
00263 }
00264
00265 /*-----*
00266 /*
00267 /*                               ORDINARY FUNCTIONS
00268 /*
00269 /*-----*/
00270
00271 void ADC_SetNotification(void) {
00272     /* This code is giving the ADC_Semaphore. */
00273     xSemaphoreGive(ADC_Semaphore);
00274 }
00275
00276
00278 void System_Init(u8 u8_AdcChannelToRead) {
00279     HCLCD_Vid4Bits_Init();
00280
00281     /*ADC Init*/
00282     MGIE_VidEnable();
00283     MADC_VidSetCallBack(ADC_SetNotification);
00284     MADC_VidADCInterruptEnable();
00285     MADC_VidInit();
00286     MADC_u16ADC_StartConversion_With_Interrupt(u8_AdcChannelToRead);
00287
00288     /* Init sensors*/
00289     MDIO_Error_State_SetPinDirection(PIN0, MDIO_PORTA, PIN_INPUT);
00290     MDIO_Error_State_SetPinDirection(PIN7, MDIO_PORTA, PIN_INPUT);
00291
00292     /* LEDS Init*/
00293     MDIO_Error_State_SetPinDirection(PIN0, MDIO_PORTC, PIN_OUTPUT);
00294     MDIO_Error_State_SetPinDirection(PIN1, MDIO_PORTC, PIN_OUTPUT);
00295     MDIO_Error_State_SetPinDirection(PIN2, MDIO_PORTC, PIN_OUTPUT);
00296
00297     /* BUZZER Init */
00298     MDIO_Error_State_SetPinDirection(PIN5, MDIO_PORTC, PIN_OUTPUT);
00299
00300     /* KEYPAD INIT */
00301     HKPD_VidInit();
00302     /* SERVO INIT */
00303     // H_ServoInit();
00304
00305     //DC Motor Init
00306     H_DcMotorInit();
00307
00308     /* Pot DC Motor Init */
00309     H_DcMotorPotInit();
00310
00311 }
00312
00313 }
00314
00315
00316
00317 void displayDoorState(void) {
00318     /* Writing the string "D: " to the LCD at the line 1, position 0 */
00319     HCLCD_VidSetPosition_4BitsMode(HCLCD_LINE1, 0);
00320     HCLCD_VidWriteString_4Bits((u8*)"D: ");
00321
00322     if(system.flag.openDoor) {
00323         HCLCD_VidSetPosition_4BitsMode(HCLCD_LINE1, 3);
00324         HCLCD_VidWriteString_4Bits((u8*)"Open ");
00325     } else {
00326         HCLCD_VidSetPosition_4BitsMode(HCLCD_LINE1, 3);
00327         HCLCD_VidWriteString_4Bits((u8*)"Close");
00328     }
00329 }
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339 }
00340
00341
00342 void displayTemperatureValue(void) {
00343
00344     /* Writing the string "T: " to the LCD at the line 1, position 10 */
00345     HCLCD_VidSetPosition_4BitsMode(HCLCD_LINE1, 10);
00346     HCLCD_VidWriteString_4Bits((u8*)"T: ");
00347
00348     /* Writing the temperature value to the LCD. */
00349     HCLCD_VidWriteNumber_4Bits(system.inputs.temperature.u8_CurrentValue);
00350     HCLCD_VidWriteString_4Bits((u8*)"C");
00351
00352     /* Adding spaces to the LCD display to clear the previous value. */
00353     if(system.inputs.temperature.u8_CurrentValue < 10) {
00354         HCLCD_VidWriteString_4Bits((u8*)"   ");
00355     } else if(system.inputs.temperature.u8_CurrentValue < 100) {
00356         HCLCD_VidWriteString_4Bits((u8*)" ");
00357     }
00358 }
```

```
00375 void displayFireState(void) {
00376
00377     /* Setting the position of the cursor to the second line and the first column. */
00378     HCLCD_VidSetPosition_4BitsMode(HCLCD_LINE2, 0);
00379
00380
00385     if(system.flag.fire) {
00386         MDIO_Error_State_SetPinValue(PIN5, MDIO_PORTC, PIN_HIGH);
00387         HCLCD_VidWriteString_4Bits((u8*)"FIRE  ");
00388     } else {
00389         /* Setting the pin value to low, only if the buzzer flag is off. */
00390         if(!system.flag.buzzer) {
00391             MDIO_Error_State_SetPinValue(PIN5, MDIO_PORTC, PIN_LOW);
00392         }
00393
00394         HCLCD_VidWriteString_4Bits((u8*)"NoFIRE");
00395     }
00396 }
00397
00398
00410 void displayEntryPasswordSecurity(void) {
00411
00412     /* Setting the position of the cursor to the second line and the 8th column. */
00413     HCLCD_VidSetPosition_4BitsMode(HCLCD_LINE2, 8);
00414
00415
00423     if(system.flag.buzzer) {
00424         HCLCD_VidWriteString_4Bits((u8*)"7RAMY(0)");
00425
00426         /* Turning on the buzzer for 3 seconds. Police arrives during the 3 seconds.*/
00427         MDIO_Error_State_SetPinValue(PIN5, MDIO_PORTC, PIN_HIGH);
00428         vTaskDelay(3000);
00429
00432         if(!system.flag.fire) {
00433             MDIO_Error_State_SetPinValue(PIN5, MDIO_PORTC, PIN_LOW);
00434         }
00435
00436         /* Setting the flag to 0 and the password attempts to the max attempts. */
00437         system.flag.buzzer = 0;
00438     } else {
00439         HCLCD_VidWriteString_4Bits((u8*)"AMAN(");
00440         HCLCD_VidWriteNumber_4Bits(system.password.attempts);
00441         HCLCD_VidWriteString_4Bits((u8*)" )");
00442     }
00443 }
00444
00445
00449 u8 comparePasswords(const u8 * const pass1, const u8 * const pass2, const u8 length) {
00450     u8 i = 0, state = 1;
00451
00452     for(i = 0; i < length; ++i) {
00453         if(pass1[i] != pass2[i]) {
00454             state = 0;
00455             break;
00456         }
00457     }
00458
00459     return state;
00460 }
00461
00462
00466 void controlDoor(void) {
00467     if(system.flag.openDoor) {
00468         SET_BIT(PORTC, 0); //RED
00469         CLR_BIT(PORTC, 1); //Green
00470         vTaskDelay(2000);
00471
00472         CLR_BIT(PORTC, 0); //RED
00473         SET_BIT(PORTC, 1); //Green
00474         vTaskDelay(2000);
00475
00476         CLR_BIT(PORTC, 1); //Green
00477         SET_BIT(PORTC, 2); //Blue
00478         vTaskDelay(2000);
00479
00480         SET_BIT(PORTC, 1); //Green
00481         CLR_BIT(PORTC, 2); //Blue
00482
00483         system.flag.openDoor = 0;
00484     } else {
00485         SET_BIT(PORTC, 1);
00486         CLR_BIT(PORTC, 0);
00487         CLR_BIT(PORTC, 2);
00488     }
00489 }
```

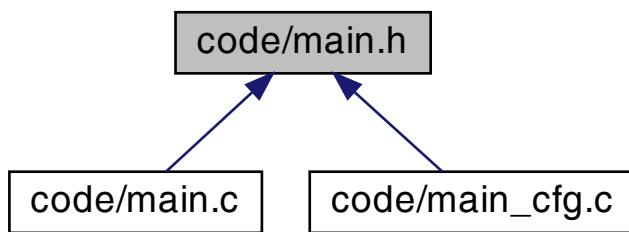
22.85 code/main.h File Reference

```
#include <util/delay.h>
#include <math.h>
#include "LIB/LSTD_TYPES.h"
#include "LIB/LBIT_MATH.h"
#include "MCAL/MDIO/MDIO_Interface.h"
#include "MCAL/MGIE/MGIE_Interface.h"
#include "MCAL/MADC/MADC_Interface.h"
#include "MCAL/TIMER2/TIMER2_Interface.h"
#include "HAL/SERVO_M/SERVO_Interface.h"
#include "HAL/HCLCD/HCLCD_Interface.h"
#include "HAL/KEY_PAD/HKPD_Interface.h"
#include "HAL/DC_MOTOR/DC_MOTOR.h"
#include "HAL/DC_MOTOR_POT/DC_MOTOR_POT.h"
#include "FreeRTOS/FreeRTOS.h"
#include "FreeRTOS/task.h"
#include "FreeRTOS/semphr.h"
```

Include dependency graph for main.h:



This graph shows which files directly or indirectly include this file:



Functions

- void **taskGetPassword** (void *pv)

Task to get the password from the keypad.
- void **taskDisplaySystemState** (void *pv)

Task to display the system's data.
- void **taskControlGasLevel** (void *pv)

Task to control the gas level.
- void **taskControlTemperature** (void *pv)

Task to control the temperature.
- void **taskReadSensors** (void *pv)

- Task to read the sensors.*
- void `ADC_SetNotification` (void)
 - void `System_Init` (u8 u8_AdcChannelToRead)
- Initialize system peripherals.*
- void `displayDoorState` (void)
- Display the current state of the door: open or closed.*
- void `displayTemperatureValue` (void)
- Display the current temperature of the room.*
- void `displayFireState` (void)
- Checking if the fire alarm is on or not.*
- void `displayEntryPasswordSecurity` (void)
- Display the security of the house (Password).*
- void `controlDoor` (void)
- It controls the door of the house. Open the door if the user enters the correct password. Otherwise, close the door.*
- u8 `comparePasswords` (const u8 *const pass1, const u8 *const pass2, const u8 length)
- Compare the password entered by the user with the default password. @retaur If the password is correct, return 1. Otherwise, return 0.*

22.85.1 Function Documentation

22.85.1.1 ADC_SetNotification() void `ADC_SetNotification` (void)

Definition at line 271 of file `main.c`.

Here is the caller graph for this function:

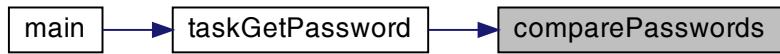


22.85.1.2 comparePasswords() u8 `comparePasswords` (const u8 *const pass1, const u8 *const pass2, const u8 length)

Compare the password entered by the user with the default password. @retaur If the password is correct, return 1. Otherwise, return 0.

Definition at line 449 of file `main.c`.

Here is the caller graph for this function:



22.85.1.3 controlDoor() void controlDoor (
void)

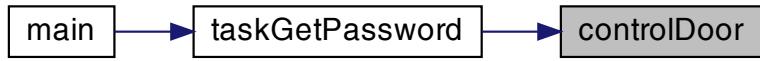
It controls the door of the house. Open the door if the user enters the correct password. Otherwise, close the door.

Definition at line 466 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.85.1.4 displayDoorState() void displayDoorState (
void)

Display the current state of the door: open or closed.

If the door is open, "D: Open" is displayed. If the door is closed, "D: Close" is displayed.

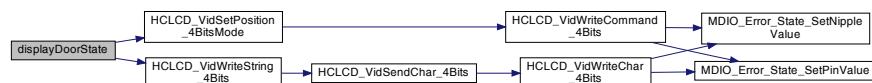
Note

This function writes to the LCD at the line 1, position 0, and uses 8 positions to write the string. So, avoid writing to the first 8 positions of the LCD in line 1.

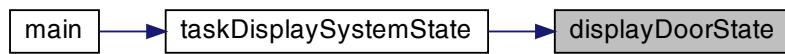
Checking if the door is open or not, and writing the string "Open" or "Close" to the LCD.

Definition at line 323 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.85.1.5 displayEntryPasswordSecurity() `void displayEntryPasswordSecurity (void)`

Display the security of the house (Password).

If the password is:

- Correct: "AMAN(x)" is written to the LCD, where x is the number of attempts.
- Wrong: "7RAMY(0)" is written to the LCD, where 0 denotes the number of attempts left is 0. And the buzzer is on for 3 seconds. We assume that the police arrives during the 3 seconds.

Note

This function writes to the LCD at the line 2, position 6, and uses 4 positions to write the string. So, avoid writing to the last 4 positions of the LCD in line 2.

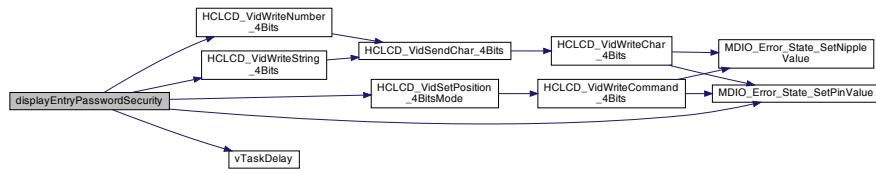
If the password is correct, it will display "AMAN(x)" where x is the number of attempts. If the password is wrong, it will display "7RAMY(0)" where 0 denotes the number of attempts left is 0. And the buzzer is on for 3 seconds. We assume that the police arrives during the 3 seconds.

< Turn off the buzzer only if the fire alarm is off. This is to prevent the buzzer from turning off when the fire alarm is on.

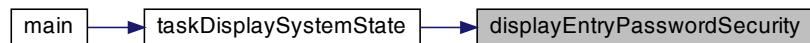
< buzzer flag is off.

Definition at line 410 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.85.1.6 displayFireState()

```

void displayFireState (
    void
)
  
```

Checking if the fire alarm is on or not.

If the fire alarm is on, the string "FIRE" is written to the LCD, and the buzzer is on. Otherwise, the string "NoFire" is written to the LCD.

Note

This function writes to the LCD at the line 2, position 0, and uses 6 positions to write the string. So, avoid writing to the first 6 positions of the LCD in line 2.

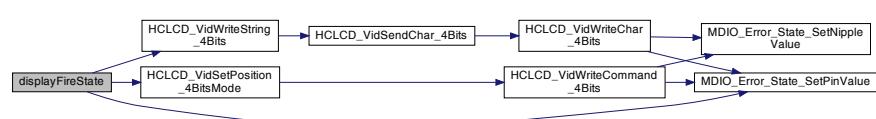
If the fire alarm is on, it will turn on the buzzer and display "FIRE" on the LCD. If it is off, it will turn off the buzzer and display "NoFire" on the LCD.

< Turn on the buzzer.

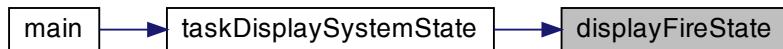
< Turn off the buzzer.

Definition at line 375 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.85.1.7 displayTemperatureValue() `void displayTemperatureValue (void)`

Display the current temperature of the room.

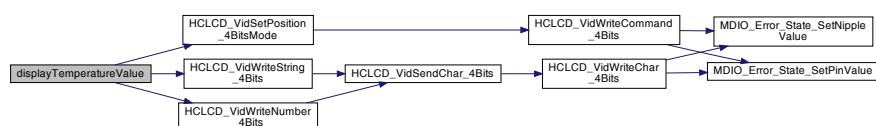
The current temperature is displayed in the LCD. "T: xx°C" is displayed.

Note

This function writes to the LCD at the line 1, position 10, and uses 7 positions to write the string. So, avoid writing to the last 7 positions of the LCD in line 1.

Definition at line 348 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.85.1.8 System_Init() `void System_Init (u8 u8_AdcChannelToRead)`

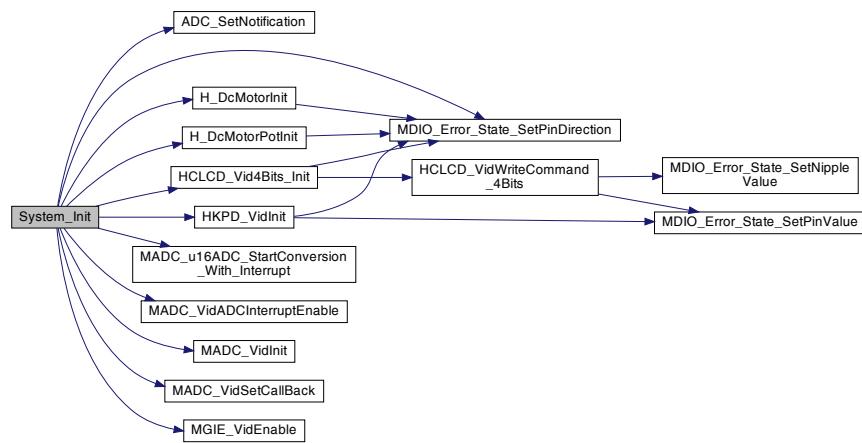
Initialize system peripherals.

Parameters

<code>u8_AdcChannelToRead</code>	The initial ADC channel to be read at the beginning.
----------------------------------	--

Definition at line 280 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```

22.85.1.9 taskControlGasLevel() void taskControlGasLevel (
    void * pv )
  
```

Task to control the gas level.

This task is responsible for controlling the gas level. It checks if the gas level is above the threshold and if it is:

- Above: Open the window, and raise the flag to alert the user (buzzer).
- Below: Close the window, and downset the flag of the alert.

Note

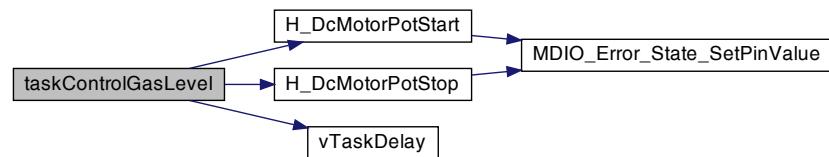
The gas level is read from the ADC using a potentiometer. It is converted to a percentage of the full scale.

Parameters

<i>pv</i>	RTOS task parameter. Not used.
-----------	--------------------------------

Definition at line 221 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.85.1.10 taskControlTemperature() `void taskControlTemperature (void * pv)`

Task to control the temperature.

This task is responsible for controlling the temperature. It checks if the temperature is above the threshold and if it is:

- Above: Open the fan.
- Below: Stop the fan.

Note

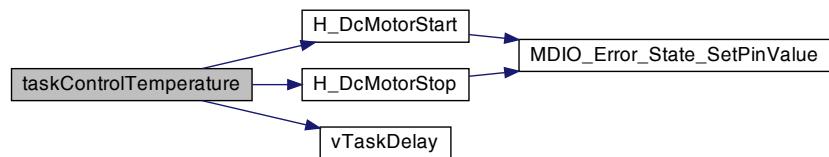
Temperature is measured using the LM35 sensor. The LM35 sensor is connected to the ADC. The sensor gives 10mV for every degree Celsius. So, you can measure the temperature in degrees Celsius by dividing the ADC voltage value by 10mV.

Parameters

<i>pv</i>	RTOS task parameter. Not used.
-----------	--------------------------------

Definition at line 249 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.85.1.11 taskDisplaySystemState() `void taskDisplaySystemState (void * pv)`

Task to display the system's data.

This task is responsible for displaying the system's data. It displays:

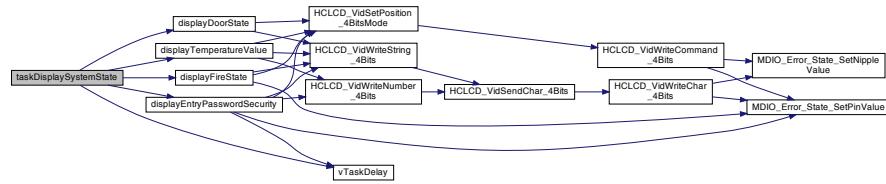
- The temperature.
- The door status.
- The fire alarm status.
- The door security status.

Parameters

<i>pv</i>	RTOS task parameter. Not used.
-----------	--------------------------------

Definition at line 156 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.85.1.12 taskGetPassword() `void taskGetPassword (void * pv)`

Task to get the password from the keypad.

This task is responsible for getting the password from the keypad. If the password is:

- Correct: raise the flag to open the door.
- Incorrect: decrease the number of attempts to open the door. If the number of attempts is 0, raise the flag to alert the user (buzzer).

Parameters

<code>pv</code>	RTOS task parameter. Not used.
-----------------	--------------------------------

< Getting the key pressed from the keyboard.
 < Check if the user pressed a key
 < If the user pressed a key, then save it to the enteredPassword array
 < Increment the index of the enteredPassword array

Checking if the password is fully entered.

If the password is:

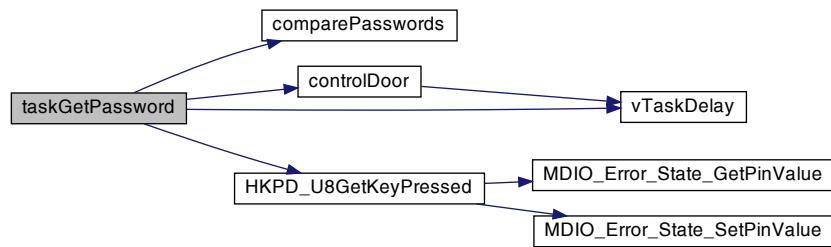
- Correct: system.flag.correctPassword is set to 1.
- Wrong: system.flag.correctPassword is set to 0 and system.password.attempts is decremented.

Note

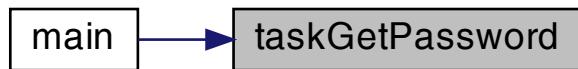
If system.password.attempts is 0, the system.flag.buzzer is set to 1.

Definition at line [94](#) of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.85.1.13 taskReadSensors() void taskReadSensors (void * *pv*)

Task to read the sensors.

This task is responsible for reading the sensors. It reads the gas level and the temperature. It uses the ADC in Non-Blocking mode, to enhances the responsiveness of the system.

Parameters

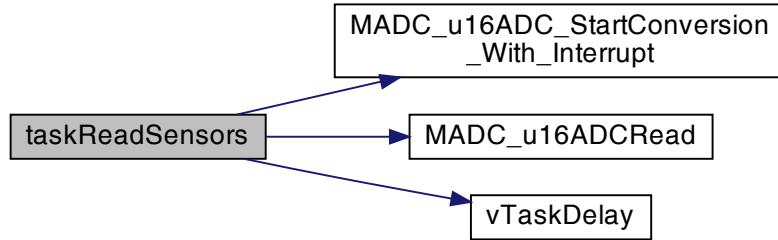
<i>pv</i>	RTOS task parameter. Not used.
-----------	--------------------------------

< Check if the ADC semaphore is available

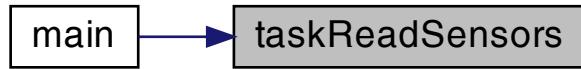
Read temperature sensor or gas sensor depending on the system.inputs.gas.u8_AdcChannel, then swap the ADC channel to read the next sensor in the next iteration.

Definition at line 175 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



22.86 main.h

```

00001 #ifndef MAIN_H
00002 #define MAIN_H
00003
00004 /-----*/
00005 /*
00006 /* Includes */
00007 */
00008 /-----*/
00009 #include <util/delay.h>
00010 #include <math.h>
00011
00012 #include "LIB/LSTD_TYPES.h"
00013 #include "LIB/LBIT_MATH.h"
00014
00015 #include "MCAL/MDIO/MDIO_Interface.h"
00016 #include "MCAL/MGIE/MGIE_Interface.h"
00017 #include "MCAL/MADC/MADC_Interface.h"
00018 #include "MCAL/TIMER2/TIMER2_Interface.h"
00019
00020 #include "HAL/SERVO_M/SERVO_Interface.h"
00021 #include "HAL/HCLCD/HCLCD_Interface.h"
00022 #include "HAL/KEY_PAD/HKPD_Interface.h"
00023 #include "HAL/DC_MOTOR/DC_MOTOR.h"
00024 #include "HAL/DC_MOTOR_POT/DC_MOTOR_POT.h"
00025
00026 #include "FreeRTOS/FreeRTOS.h"
00027 #include "FreeRTOS/task.h"
00028 #include "FreeRTOS/semphr.h"
00029
00030
00031 /-----*/
00032 /*
00033 /* TASKS PROTOTYPES */
00034 */
00035 /-----*/

```

```

00036 void taskGetPassword(void * pv);
00037 void taskDisplaySystemState(void * pv );
00038 void taskControlGasLevel(void * pv);
00039 void taskControlTemperature(void * pv);
00040 void taskReadSensors(void * pv);
00041 void ADC_SetNotification(void);
00042
00043 /*-----*/
00044 /*
00045  *          ORDINARY FUNCTIONS PROTOTYPES
00046  */
00047 /*-----*/
00048
00049 void System_Init(u8 u8_AdcChannelToRead);
00050 void displayDoorState(void);
00051 void displayTemperatureValue(void);
00052 void displayFireState(void);
00053 void displayEntryPasswordSecurity(void);
00054 void controlDoor(void);
00055 u8 comparePasswords(const u8 * const pass1, const u8 * const pass2, const u8 length);
00056
00057 #endif /* MAIN_H */

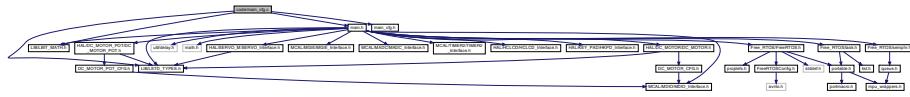
```

22.87 code/main_cfg.c File Reference

```

#include "LIB/LSTD_TYPES.h"
#include "LIB/LBIT_MATH.h"
#include "main.h"
#include "main_cfg.h"
Include dependency graph for main_cfg.c:

```



Variables

- SYSTEM_t system

22.87.1 Variable Documentation

22.87.1.1 system SYSTEM_t system

Initial value:

```

= {
    .inputs.temperature = {
        .u8_MaxValue      = 150,
        .u8_CriticalValue = 40,
        .u8_CurrentValue  = 0,
        .u16_CurrentValueInBinary = 0,
        .u8_AdcChannel   = 7,
    },
    .inputs.gas = {
        .u8_MaxValue      = 100,
        .u8_CriticalValue = 50,
        .u8_CurrentValue  = 0,
        .u16_CurrentValueInBinary = 0,
        .u8_AdcChannel   = 0,
    },
    .password = {
        .value      = {'1', '2', '3', '4'},
        .attempts   = 3,
    }
}

```

```

        .maxAttemps = 3
    },
    .flag = {
        .correctPassword = 0,
        .buzzer = 0,
        .openDoor = 0,
        .fire = 0,
    }
}
}

```

Definition at line 20 of file [main_cfg.c](#).

22.88 main_cfg.c

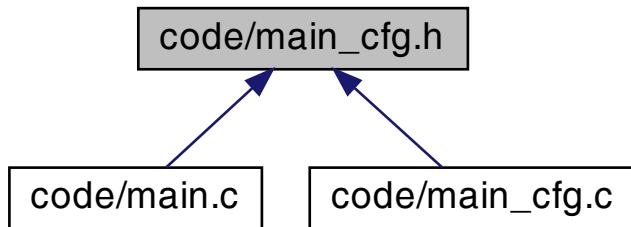
```

00001 /*-----*/
00002 /*
00003 *           Includes
00004 */
00005 /*-----*/
00006 #include "LIB/LSTD_TYPES.h"
00007 #include "LIB/LBIT_MATH.h"
00008
00009 #include "main.h"
00010 #include "main_cfg.h"
00011
00012
00013
00014 /*-----*/
00015 /*
00016 *           CHANGE THE FOLLOWING TO YOUR NEEDS
00017 */
00018 /*-----*/
00019
00020 SYSTEM_t system = {
00021     .inputs.temperature = {
00022         .u8_MaxValue = 150,      /* Maximum temperature: 150 C */
00023         .u8_CriticalValue = 40,  /* Critical temperature: 40 C */
00024         .u8_CurrentValue = 0,    /* Current temperature reading */
00025         .u16_CurrentValueInBinary = 0, /* Current temperature reading in binary */
00026         .u8_AdcChannel = 7,     /* ADC channel for temperature sensor */
00027     },
00028     .inputs.gas = {
00029         .u8_MaxValue = 100,      /* Maximum gas: 100 % */
00030         .u8_CriticalValue = 50,  /* Critical gas: 50 % */
00031         .u8_CurrentValue = 0,    /* Current gas level reading */
00032         .u16_CurrentValueInBinary = 0, /* Current gas level reading in binary */
00033         .u8_AdcChannel = 0,     /* ADC channel for gas sensor */
00034     },
00035     .password = {
00036         .value = {'1', '2', '3', '4'}, /* Password: 1234 */
00037         .attempts = 3,             /* Number of attempts to enter password */
00038         .maxAttemps = 3            /* Maximum number of attempts to enter password */
00039     },
00040     .flag = {
00041         .correctPassword = 0,      /* Flag to check if the password is correct */
00042         .buzzer = 0,              /* Flag to enable the buzzer */
00043         .openDoor = 0,             /* Flag to open the door */
00044         .fire = 0,                /* Flag to fire the alarm */
00045     }
00046 };

```

22.89 code/main_cfg.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct [FLAG_t](#)
This is the structure that holds the system's state flags.
- struct [PASSWORD_t](#)
This is the structure that holds the system's password data.
- struct [SENSOR_t](#)
This is the structure that holds the system's sensors data.
- struct [SYSTEM_INPUTS_t](#)
- struct [SYSTEM_t](#)
*This is the structure that holds the system's data that is used to control the system and display it on the LCD screen.
The system's data is:*

Macros

- #define [PASSWORD_LENGTH](#) (4U)

Variables

- [SYSTEM_t](#) system

22.89.1 Macro Definition Documentation

22.89.1.1 [PASSWORD_LENGTH](#) #define PASSWORD_LENGTH (4U)

Definition at line 9 of file [main_cfg.h](#).

22.89.2 Variable Documentation

22.89.2.1 system SYSTEM_t system [extern]

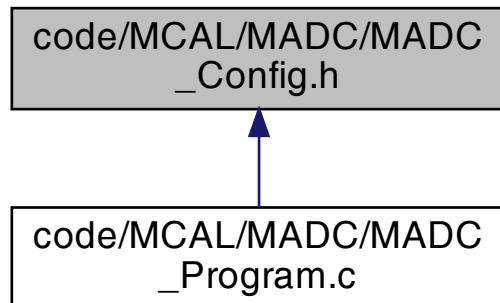
Definition at line 20 of file [main_cfg.c](#).

22.90 main_cfg.h

```
00001 #ifndef MAIN_CFG_H
00002 #define MAIN_CFG_H
00003
00004 /-----*/
00005 /*
00006 /*           CHANGE THESE VALUES TO YOUR NEEDS
00007 */
00008 /-----*/
00009 #define PASSWORD_LENGTH      (4U)
00010
00011 /-----*/
00012 /*
00013 /*           DO NOT CHANGE THE FOLLOWING
00014 */
00015 /-----*/
00016
00017 /-----*/
00018 /*
00019 /*           TYPEDEFS
00020 */
00021 /-----*/
00022
00023
00024 typedef struct {
00025     u8 correctPassword;
00026     u8 buzzer;
00027     u8 openDoor;
00028     u8 fire;
00029 } FLAG_t;
00030
00031
00032
00033
00034 typedef struct {
00035     u8 value[PASSWORD_LENGTH];
00036     u8 attempt;
00037     const u8 maxAttempts;
00038 } PASSWORD_t;
00039
00040
00041
00042
00043 typedef struct {
00044     u16 u16_CurrentValueInBinary;
00045     u8 u8_CriticalValue;
00046     u8 u8_MaxValue;
00047     u8 u8_CurrentValue;
00048     u8 u8_AdcChannel;
00049 } SENSOR_t;
00050
00051
00052
00053 typedef struct {
00054     SENSOR_t temperature;
00055     SENSOR_t gas;
00056 } SYSTEM_INPUTS_t;
00057
00058
00059 typedef struct {
00060     SYSTEM_INPUTS_t inputs;
00061     PASSWORD_t password;
00062     FLAG_t flag;
00063 } SYSTEM_t;
00064
00065
00066
00067 extern SYSTEM_t system;
00068
00069
00070 #endif /* MAIN_CFG_H */
```

22.91 code/MCAL/MADC/MADC_Config.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- `#define MADC_SET_REFERENCEVOLATGE MADC_AVCC_REFERENCEVOLATGE`
- `#define MADC_SET_PRESCALER MADC_128_PRESCALER`
- `#define MADC_SET_ADJUST MADC_RIGHT_ADJUST`

22.91.1 Macro Definition Documentation

22.91.1.1 **MADC_SET_ADJUST** `#define MADC_SET_ADJUST MADC_RIGHT_ADJUST`

Definition at line 29 of file [MADC_Config.h](#).

22.91.1.2 **MADC_SET_PRESCALER** `#define MADC_SET_PRESCALER MADC_128_PRESCALER`

Definition at line 22 of file [MADC_Config.h](#).

22.91.1.3 **MADC_SET_REFERENCEVOLATGE** `#define MADC_SET_REFERENCEVOLATGE MADC_AVCC_REFERENCEVOLATGE`

Definition at line 11 of file [MADC_Config.h](#).

22.92 MADC_Config.h

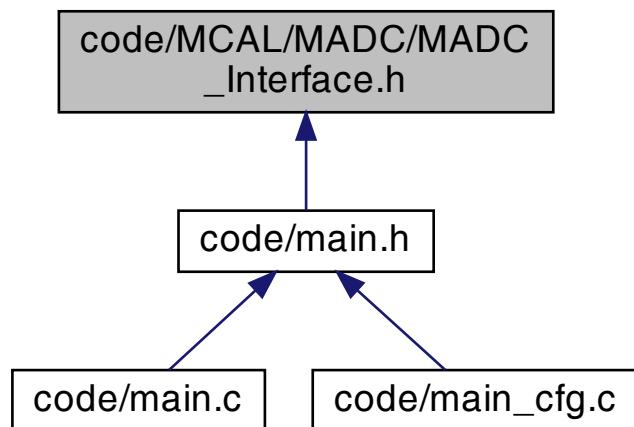
```

00001
00002
00003 #ifndef MCAL_MADC_MADC_CONFIG_H_
00004 #define MCAL_MADC_MADC_CONFIG_H_
00005
00006 /*MADC REFERENCE VOLATGE Options:
00007 *1- MADC_AVCC_REFERENCEVOLATGE
00008 *2- MADC_INTERNAL_REFERENCEVOLATGE
00009 */
00010
00011 #define MADC_SET_REFERENCEVOLATGE MADC_AVCC_REFERENCEVOLATGE
00012
00013 /*MADC Prescaler options:
00014 * 1- MADC_2_PRESCALER
00015 * 2- MADC_4_PRESCALER
00016 * 3- MADC_8_PRESCALER
00017 * 4- MADC_16_PRESCALER
00018 * 5- MADC_32_PRESCALER
00019 * 6- MADC_64_PRESCALER
00020 * 7- MADC_128_PRESCALER
00021 */
00022 #define MADC_SET_PRESCALER MADC_128_PRESCALER
00023
00024 /*MADC Adjust Options:
00025 *1- MADC_LEFT_ADJUST
00026 *2- MADC_RIGHT_ADJUST
00027 */
00028
00029 #define MADC_SET_ADJUST MADC_RIGHT_ADJUST
00030
00031#endif /* MCAL_MADC_MADC_CONFIG_H_ */

```

22.93 code/MCAL/MADC/MADC_Interface.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define CHANNEL_0 0
- #define CHANNEL_1 1
- #define CHANNEL_2 2
- #define CHANNEL_3 3

- #define CHANNEL_4 4
- #define CHANNEL_5 5
- #define CHANNEL_6 6
- #define CHANNEL_7 7

Functions

- void MADC_VidInit (void)
- void MADC_VidADCInterruptEnable (void)
- u16 MADC_u16ADC_StartConversion (u8 Copy_u8Channel)
- void MADC_u16ADC_StartConversion_With_Interrupt (u8 Copy_u8Channel)
- u16 MADC_u16ADCRead (void)
- void MADC_VidSetCallBack (void(*Copy_pFun)(void))

22.93.1 Macro Definition Documentation

22.93.1.1 CHANNEL_0 #define CHANNEL_0 0

Definition at line 6 of file [MADC_Interface.h](#).

22.93.1.2 CHANNEL_1 #define CHANNEL_1 1

Definition at line 7 of file [MADC_Interface.h](#).

22.93.1.3 CHANNEL_2 #define CHANNEL_2 2

Definition at line 8 of file [MADC_Interface.h](#).

22.93.1.4 CHANNEL_3 #define CHANNEL_3 3

Definition at line 9 of file [MADC_Interface.h](#).

22.93.1.5 CHANNEL_4 #define CHANNEL_4 4

Definition at line 10 of file [MADC_Interface.h](#).

22.93.1.6 CHANNEL_5 #define CHANNEL_5 5

Definition at line 11 of file [MADC_Interface.h](#).

22.93.1.7 CHANNEL_6 #define CHANNEL_6 6

Definition at line 12 of file [MADC_Interface.h](#).

22.93.1.8 CHANNEL_7 #define CHANNEL_7 7

Definition at line 13 of file [MADC_Interface.h](#).

22.93.2 Function Documentation

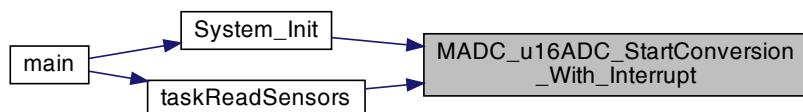
22.93.2.1 MADC_u16ADC_StartConversion() u16 MADC_u16ADC_StartConversion (
 u8 Copy_u8Channel)

Definition at line 42 of file [MADC_Program.c](#).

22.93.2.2 MADC_u16ADC_StartConversion_With_Interrupt() void MADC_u16ADC_StartConversion_With_Interrupt (
 u8 Copy_u8Channel)

Definition at line 55 of file [MADC_Program.c](#).

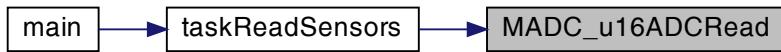
Here is the caller graph for this function:



22.93.2.3 MADC_u16ADCRead() `u16 MADC_u16ADCRead (`
 `void)`

Definition at line 64 of file [MADC_Program.c](#).

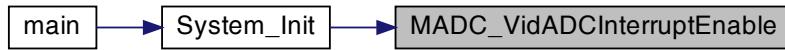
Here is the caller graph for this function:



22.93.2.4 MADC_VidADCInterruptEnable() `void MADC_VidADCInterruptEnable (`
 `void)`

Definition at line 37 of file [MADC_Program.c](#).

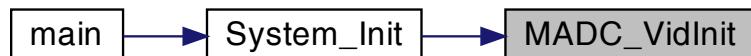
Here is the caller graph for this function:



22.93.2.5 MADC_VidInit() `void MADC_VidInit (`
 `void)`

Definition at line 11 of file [MADC_Program.c](#).

Here is the caller graph for this function:



```
22.93.2.6 MADC_VidSetCallBack() void MADC_VidSetCallBack (
    void(*)(void) Copy_pFun )
```

Definition at line 68 of file [MADC_Program.c](#).

Here is the caller graph for this function:

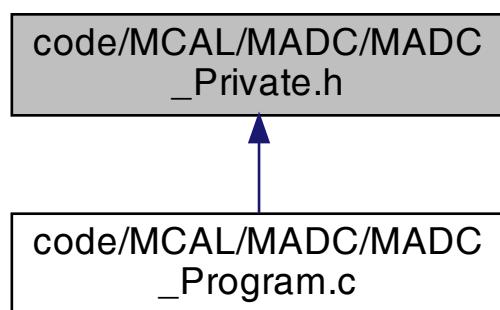


22.94 MADC_Interface.h

```
00001
00002
00003 #ifndef MCAL_MADC_MADC_INTERFACE_H_
00004 #define MCAL_MADC_MADC_INTERFACE_H_
00005
00006 #define CHANNEL_0      0
00007 #define CHANNEL_1      1
00008 #define CHANNEL_2      2
00009 #define CHANNEL_3      3
00010 #define CHANNEL_4      4
00011 #define CHANNEL_5      5
00012 #define CHANNEL_6      6
00013 #define CHANNEL_7      7
00014
00015 /*ADC Initialization*/
00016 void MADC_VidInit(void);
00017 void MADC_VidADCInterruptEnable(void);
00018 /*ADC Start Conversion-->Polling , Return ADC Value*/
00019 u16 MADC_u16ADC_StartConversion(u8 Copy_u8Channel);
00020 void MADC_u16ADC_StartConversion_With Interrupt(u8 Copy_u8Channel);
00021 u16 MADC_u16ADCRead(void);
00022 void MADC_VidSetCallBack(void (*Copy_pFun)(void));
00023
00024
00025
00026
00027 #endif /* MCAL_MADC_MADC_INTERFACE_H_ */
```

22.95 code/MCAL/MADC/MADC_Private.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define ADMUX *((volatile u8*)0x27)
- #define ADCSRA *((volatile u8*)0x26)
- #define ADC *((volatile u16*)0x24)
- #define SFIOR *((volatile u16*)0x50)
- #define MADC_AVCC_REFERENCEVOLATGE 1
- #define MADC_INTERNAL_REFERENCEVOLATGE 2
- #define MADC_BIT_MASKING_PRESCALER 0xF8
- #define MADC_BIT_MASKING_CHANNEL 0x07
- #define MADC_BIT_MASKING_REG_CHANNEL 0xE0
- #define MADC_2_PRESCALER 1
- #define MADC_4_PRESCALER 2
- #define MADC_8_PRESCALER 3
- #define MADC_16_PRESCALER 4
- #define MADC_32_PRESCALER 5
- #define MADC_64_PRESCALER 6
- #define MADC_128_PRESCALER 7
- #define MADC_RIGHT_ADJUST 0
- #define MADC_LEFT_ADJUST 1

Functions

- void __vector_16 (void) __attribute__((signal))

22.95.1 Macro Definition Documentation

22.95.1.1 ADC #define ADC *((volatile u16*)0x24)

Definition at line 8 of file [MADC_Private.h](#).

22.95.1.2 ADCSRA #define ADCSRA *((volatile u8*)0x26)

Definition at line 7 of file [MADC_Private.h](#).

22.95.1.3 ADMUX #define ADMUX *((volatile u8*)0x27)

Definition at line 6 of file [MADC_Private.h](#).

22.95.1.4 MADC_128_PRESCALER #define MADC_128_PRESCALER 7

Definition at line 24 of file [MADC_Private.h](#).

22.95.1.5 MADC_16_PRESCALER #define MADC_16_PRESCALER 4

Definition at line 21 of file [MADC_Private.h](#).

22.95.1.6 MADC_2_PRESCALER #define MADC_2_PRESCALER 1

Definition at line 18 of file [MADC_Private.h](#).

22.95.1.7 MADC_32_PRESCALER #define MADC_32_PRESCALER 5

Definition at line 22 of file [MADC_Private.h](#).

22.95.1.8 MADC_4_PRESCALER #define MADC_4_PRESCALER 2

Definition at line 19 of file [MADC_Private.h](#).

22.95.1.9 MADC_64_PRESCALER #define MADC_64_PRESCALER 6

Definition at line 23 of file [MADC_Private.h](#).

22.95.1.10 MADC_8_PRESCALER #define MADC_8_PRESCALER 3

Definition at line 20 of file [MADC_Private.h](#).

22.95.1.11 MADC_AVCC_REFERENCEVOLATGE #define MADC_AVCC_REFERENCEVOLATGE 1

Definition at line 11 of file [MADC_Private.h](#).

22.95.1.12 MADC_BIT_MASKING_CHANNEL #define MADC_BIT_MASKING_CHANNEL 0x07

Definition at line 15 of file [MADC_Private.h](#).

22.95.1.13 MADC_BIT_MASKING_PRESCALER #define MADC_BIT_MASKING_PRESCALER 0xF8

Definition at line 14 of file [MADC_Private.h](#).

22.95.1.14 MADC_BIT_MASKING_REG_CHANNEL #define MADC_BIT_MASKING_REG_CHANNEL 0xE0

Definition at line 16 of file [MADC_Private.h](#).

22.95.1.15 MADC_INTERNAL_REFERENCEVOLATGE #define MADC_INTERNAL_REFERENCEVOLATGE 2

Definition at line 12 of file [MADC_Private.h](#).

22.95.1.16 MADC_LEFT_ADJUST #define MADC_LEFT_ADJUST 1

Definition at line 27 of file [MADC_Private.h](#).

22.95.1.17 MADC_RIGHT_ADJUST #define MADC_RIGHT_ADJUST 0

Definition at line 26 of file [MADC_Private.h](#).

22.95.1.18 SFIOR #define SFIOR *((volatile u16*) 0x50)

Definition at line 9 of file [MADC_Private.h](#).

22.95.2 Function Documentation

22.95.2.1 __vector_16() void __vector_16 (void)

Definition at line 73 of file [MADC_Program.c](#).

22.96 MADC_Private.h

```

00001
00002
00003 #ifndef MCAL_MADC_MADC_PRIVATE_H_
00004 #define MCAL_MADC_MADC_PRIVATE_H_
00005
00006 #define ADMUX      *((volatile u8*)0x27)
00007 #define ADCSRA     *((volatile u8*)0x26)
00008 #define ADC        *((volatile u16*)0x24)
00009 #define SFIOR      *((volatile u16*)0x50)
00010
00011 #define MADC_AVCC_REFERENCEVOLATGE      1
00012 #define MADC_INTERNAL_REFERENCEVOLATGE    2
00013
00014 #define MADC_BIT_MASKING_PRESCALER      0xF8
00015 #define MADC_BIT_MASKING_CHANNEL        0x07
00016 #define MADC_BIT_MASKING_REG_CHANNEL    0xE0
00017
00018 #define MADC_2_PRESCALER      1
00019 #define MADC_4_PRESCALER      2
00020 #define MADC_8_PRESCALER      3
00021 #define MADC_16_PRESCALER     4
00022 #define MADC_32_PRESCALER     5
00023 #define MADC_64_PRESCALER     6
00024 #define MADC_128_PRESCALER    7
00025
00026 #define MADC_RIGHT_ADJUST      0
00027 #define MADC_LEFT_ADJUST       1
00028
00029 void __vector_16(void)     __attribute__((signal));
00030
00031 #endif /* MCAL_MADC_MADC_PRIVATE_H_ */

```

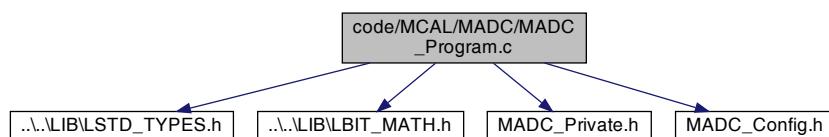
22.97 code/MCAL/MADC/MADC_Program.c File Reference

```

#include "..\..\LIB\LSTD_TYPES.h"
#include "..\..\LIB\LBIT_MATH.h"
#include "MADC_Private.h"
#include "MADC_Config.h"

```

Include dependency graph for MADC_Program.c:



Functions

- void **MADC_VidInit** (void)
- void **MADC_VidADCInterruptEnable** (void)
- **u16 MADC_u16ADC_StartConversion (u8 Copy_u8Channel)**
- void **MADC_u16ADC_StartConversion_With_Interrupt (u8 Copy_u8Channel)**
- **u16 MADC_u16ADCRead (void)**
- void **MADC_VidSetCallBack (void(*Copy_pFun)(void))**
- void **__vector_16 (void)**

Variables

- void(* **MADC_CallBack**)(void)

22.97.1 Function Documentation

22.97.1.1 __vector_16() `void __vector_16 (void)`

Definition at line 73 of file [MADC_Program.c](#).

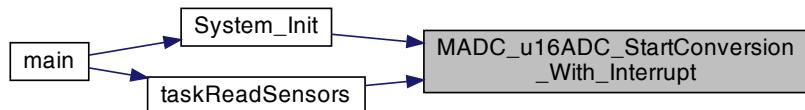
22.97.1.2 MADC_u16ADC_StartConversion() `u16 MADC_u16ADC_StartConversion (u8 Copy_u8Channel)`

Definition at line 42 of file [MADC_Program.c](#).

22.97.1.3 MADC_u16ADC_StartConversion_With_Interrupt() `void MADC_u16ADC_StartConversion_With_Interrupt (u8 Copy_u8Channel)`

Definition at line 55 of file [MADC_Program.c](#).

Here is the caller graph for this function:



22.97.1.4 MADC_u16ADCRead() `u16 MADC_u16ADCRead (void)`

Definition at line 64 of file [MADC_Program.c](#).

Here is the caller graph for this function:



```
22.97.1.5 MADC_VidADCInterruptEnable() void MADC_VidADCInterruptEnable (
    void )
```

Definition at line 37 of file [MADC_Program.c](#).

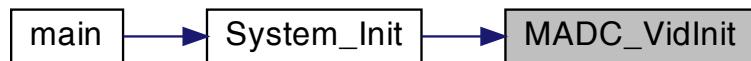
Here is the caller graph for this function:



```
22.97.1.6 MADC_VidInit() void MADC_VidInit (
    void )
```

Definition at line 11 of file [MADC_Program.c](#).

Here is the caller graph for this function:



```
22.97.1.7 MADC_VidSetCallBack() void MADC_VidSetCallBack (
    void(*)(void) Copy_pFun )
```

Definition at line 68 of file [MADC_Program.c](#).

Here is the caller graph for this function:



22.97.2 Variable Documentation

22.97.2.1 MADC_CallBack void(* MADC_CallBack) (void) (

```
void )
```

Definition at line 8 of file [MADC_Program.c](#).

22.98 MADC_Program.c

```

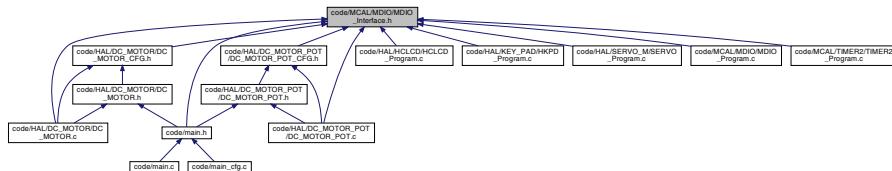
00001
00002 #include"..\..\LIB\LSTD_TYPES.h"
00003 #include"..\..\LIB\LBIT_MATH.h"
00004
00005 #include"MADC_Private.h"
00006 #include"MADC_Config.h"
00007
00008 void (*MADC_CallBack) (void);
00009
0010 /*ADC Initialization*/
0011 void MADC_VidInit(void)
0012 {
0013     /*Select Reference Voltage*/
0014 #if MADC_SET_REFERENCEVOLATGE == MADC_AVCC_REFERENCEVOLATGE
0015     SET_BIT(ADMUX,6);
0016     CLR_BIT(ADMUX,7);
0017 #elif MADC_SET_REFERENCEVOLATGE == MADC_2.56V_REFERENCEVOLATGE
0018     SET_BIT(ADMUX,6);
0019     SET_BIT(ADMUX,7);
0020 #else
0021 #error"ADC Reference Voltage Option I s not valid.... "
0022 #endif
0023 /*Set Prescaler*/
0024 ADCSRA&=MADC_BIT_MASKING_PRESCALER;
0025 ADCSRA|=MADC_SET_PRESCALER;
0026
0027 #if MADC_SET_ADJUST == MADC_LEFT_ADJUST
0028     SET_BIT(ADMUX,5);
0029 #elif MADC_SET_ADJUST == MADC_RIGHT_ADJUST
0030     CLR_BIT(ADMUX,5);
0031 #else
0032 #error"ADC Left adjust option is not valid...."
0033 #endif
0034 /*Enable To ADC*/
0035 SET_BIT(ADCSRA,7);
0036 }
0037 void MADC_VidADCInterruptEnable(void)
0038 {
0039     SET_BIT(ADCSRA,3);
0040 }
0041 /*ADC Start Conversion-->Polling , Return ADC Value*/
0042 u16 MADC_u16ADC_StartConversion(u8 Copy_u8Channel)
0043 {
0044     /*Select Channel*/
0045     Copy_u8Channel&=MADC_BIT_MASKING_CHANNEL;
0046     ADMUX&=MADC_BIT_MASKING_REG_CHANNEL;
0047     ADMUX|=Copy_u8Channel;
0048     /*send Start Conversion*/
0049     SET_BIT(ADCSRA,6);
0050     /*Wait On ADC Conversion Completed Flag is set to one */
0051     while(GET_BIT(ADCSRA,4)==0);
0052
0053     return ADC;
0054 }
0055 void MADC_u16ADC_StartConversion_With_Interrupt(u8 Copy_u8Channel)
0056 {
0057     /*Select Channel*/
0058     Copy_u8Channel&=MADC_BIT_MASKING_CHANNEL;
0059     ADMUX&=MADC_BIT_MASKING_REG_CHANNEL;
0060     ADMUX|=Copy_u8Channel;
0061     /*send Start Conversion*/
0062     SET_BIT(ADCSRA,6);
0063 }
0064 u16 MADC_u16ADCRead(void)
0065 {
0066     return ADC;
0067 }
0068 void MADC_VidSetCallBack(void (*Copy_pFun) (void))
0069 {
0070     MADC_CallBack=Copy_pFun;
0071 }
0072
0073 void __vector_16(void)
0074 {
0075     MADC_CallBack();
0076 }
```

22.99 code/MCAL/MDIO/MDIO_Config.h File Reference

22.100 MDIO_Config.h

22.101 code/MCAL/MDIO/MDIO_Interface.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- `#define MDIO_PORTA 0`
- `#define MDIO_PORTB 1`
- `#define MDIO_PORTC 2`
- `#define MDIO_PORTD 3`
- `#define PIN_OUTPUT 1`
- `#define PIN_INPUT 0`
- `#define PORT_OUTPUT 255`
- `#define PORT_INPUT 0`
- `#define PIN_HIGH 1`
- `#define PIN_LOW 0`

Enumerations

- enum `Pin_t` {
 `PIN0 =0 , PIN1 , PIN2 , PIN3 ,`
`PIN4 , PIN5 , PIN6 , PIN7 }`

Functions

- `Error_State MDIO_Error_State_SetPinDirection (u8 Copy_u8PinNumber, u8 Copy_u8PortNumber, u8 Copy_u8PinDirection)`
- `Error_State MDIO_Error_State_SetPortDirection (u8 Copy_u8PortNumber, u8 Copy_u8PortDirection)`
- `Error_State MDIO_Error_State_SetPinValue (u8 Copy_u8PinNumber, u8 Copy_u8PortNumber, u8 Copy_u8PinValue)`
- `Error_State MDIO_Error_State_SetPortValue (u8 Copy_u8PortNumber, u8 Copy_u8PortValue)`
- `Error_State MDIO_Error_State_GetPinValue (u8 Copy_u8PinNumber, u8 Copy_u8PortNumber, u8 *P_u8PinValue)`
- `Error_State MDIO_Error_State_SetNippleValue (u8 Copy_u8PinStart, u8 Copy_u8PortNumber, u8 Copy_u8Value)`

22.101.1 Macro Definition Documentation

22.101.1.1 MDIO_PORTA #define MDIO_PORTA 0

Definition at line 4 of file [MDIO_Interface.h](#).

22.101.1.2 MDIO_PORTB #define MDIO_PORTB 1

Definition at line 5 of file [MDIO_Interface.h](#).

22.101.1.3 MDIO_PORTC #define MDIO_PORTC 2

Definition at line 6 of file [MDIO_Interface.h](#).

22.101.1.4 MDIO_PORTD #define MDIO_PORTD 3

Definition at line 7 of file [MDIO_Interface.h](#).

22.101.1.5 PIN_HIGH #define PIN_HIGH 1

Definition at line 15 of file [MDIO_Interface.h](#).

22.101.1.6 PIN_INPUT #define PIN_INPUT 0

Definition at line 10 of file [MDIO_Interface.h](#).

22.101.1.7 PIN_LOW #define PIN_LOW 0

Definition at line 16 of file [MDIO_Interface.h](#).

22.101.1.8 PIN_OUTPUT #define PIN_OUTPUT 1

Definition at line 9 of file [MDIO_Interface.h](#).

22.101.1.9 PORT_INPUT #define PORT_INPUT 0

Definition at line 13 of file [MDIO_Interface.h](#).

22.101.1.10 PORT_OUTPUT #define PORT_OUTPUT 255

Definition at line 12 of file [MDIO_Interface.h](#).

22.101.2 Enumeration Type Documentation

22.101.2.1 Pin_t enum Pin_t

Enumerator

PIN0	
PIN1	
PIN2	
PIN3	
PIN4	
PIN5	
PIN6	
PIN7	

Definition at line 18 of file [MDIO_Interface.h](#).

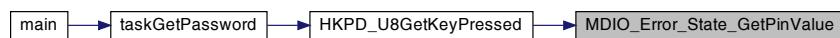
22.101.3 Function Documentation

22.101.3.1 MDIO_Error_State_GetPinValue() [Error_State](#) MDIO_Error_State_GetPinValue (

```
    u8 Copy_u8PinNumber,
    u8 Copy_u8PortNumber,
    u8 * P_u8PinValue )
```

Definition at line 220 of file [MDIO_Program.c](#).

Here is the caller graph for this function:

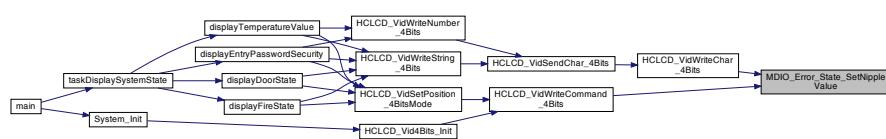


22.101.3.2 MDIO_Error_State_SetNippleValue() [Error_State](#) MDIO_Error_State_SetNippleValue (

```
    u8 Copy_u8PinStart,
    u8 Copy_u8PortNumber,
    u8 Copy_u8Value )
```

Definition at line 255 of file [MDIO_Program.c](#).

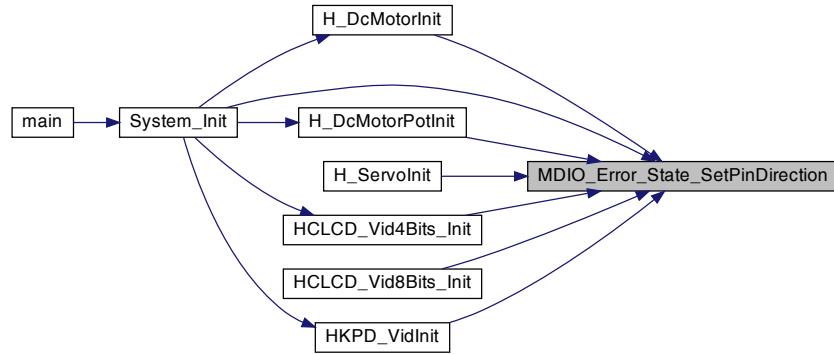
Here is the caller graph for this function:



22.101.3.3 MDIO_Error_State_SetPinDirection() `Error_State MDIO_Error_State_SetPinDirection (`

Definition at line 9 of file [MDIO_Program.c](#).

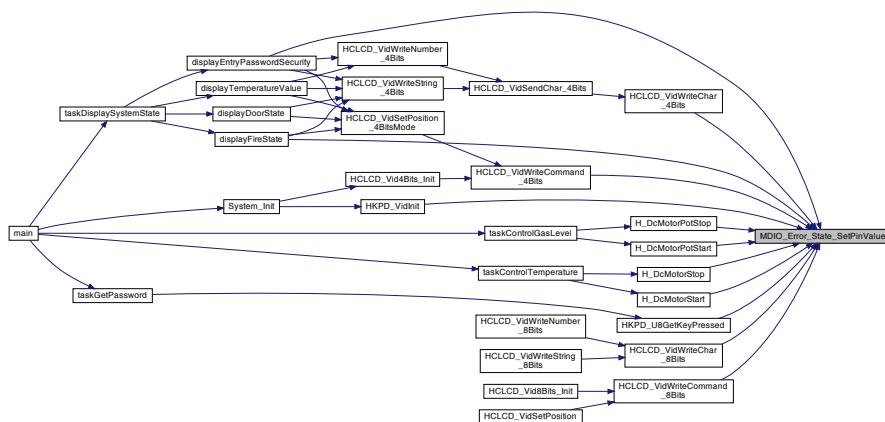
Here is the caller graph for this function:



22.101.3.4 MDIO_Error_State_SetPinValue() `Error_State MDIO_Error_State_SetPinValue (`

Definition at line 117 of file [MDIO_Program.c](#).

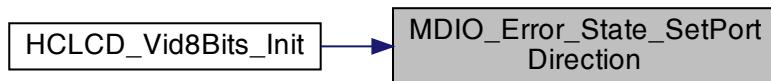
Here is the caller graph for this function:



22.101.3.5 MDIO_Error_State_SetPortDirection() `Error_State MDIO_Error_State_SetPortDirection (u8 Copy_u8PortNumber,
u8 Copy_u8PortDirection)`

Definition at line 85 of file [MDIO_Program.c](#).

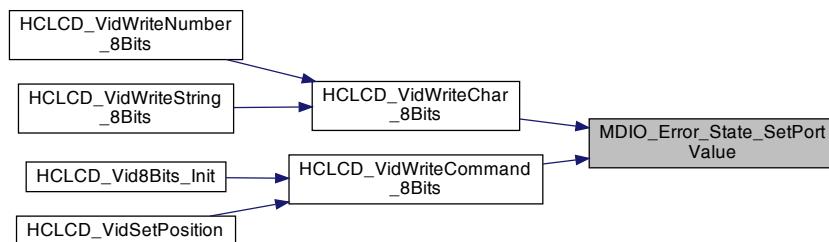
Here is the caller graph for this function:



22.101.3.6 MDIO_Error_State_SetPortValue() `Error_State MDIO_Error_State_SetPortValue (u8 Copy_u8PortNumber,
u8 Copy_u8PortValue)`

Definition at line 195 of file [MDIO_Program.c](#).

Here is the caller graph for this function:



22.102 MDIO_Interface.h

```

00001 #ifndef MDIO_INTERFACE_H_
00002 #define MDIO_INTERFACE_H_
00003
00004 #define MDIO_PORTA 0
00005 #define MDIO_PORTB 1
00006 #define MDIO_PORTC 2
00007 #define MDIO_PORTD 3
00008
00009 #define PIN_OUTPUT 1
00010 #define PIN_INPUT 0
00011
00012 #define PORT_OUTPUT 255
00013 #define PORT_INPUT 0
00014
00015 #define PIN_HIGH 1
00016 #define PIN_LOW 0
00017
00018 typedef enum
00019 {
00020     PIN0=0,

```

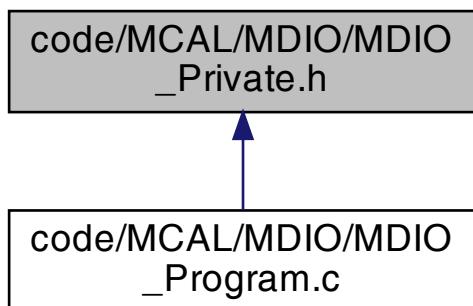
```

00021     PIN1,
00022     PIN2,
00023     PIN3,
00024     PIN4,
00025     PIN5,
00026     PIN6,
00027     PIN7
00028 }Pin_t;
00029
00030 /*Set Pin Direction Function */
00031 Error_State MDIO_Error_State_SetPinDirection(u8 Copy_u8PinNumber,u8 Copy_u8PortNumber,u8
Copy_u8PinDirection);
00032 /*Set Port Direction Function */
00033 Error_State MDIO_Error_State_SetPortDirection(u8 Copy_u8PortNumber,u8 Copy_u8PortDirection);
00034 /*Set Pin Value Function */
00035 Error_State MDIO_Error_State_SetPinValue(u8 Copy_u8PinNumber,u8 Copy_u8PortNumber,u8 Copy_u8PinValue);
00036 /*Set Port Value Function */
00037 Error_State MDIO_Error_State_SetPortValue(u8 Copy_u8PortNumber,u8 Copy_u8PortValue);
00038 /*Get Pin Value Function */
00039 Error_State MDIO_Error_State_GetPinValue(u8 Copy_u8PinNumber,u8 Copy_u8PortNumber,u8* P_u8PinValue);
00040
00041 /*Toggle Pin Value Function */
00042
00043 /*Active Pull Up Resistor Function*/
00044
00045 /*Set Nipple Direction Function */
00046
00047 /*Set Nipple Values Function */
00048 Error_State MDIO_Error_State_SetNippleValue(u8 Copy_u8PinStart,u8 Copy_u8PortNumber,u8 Copy_u8Value);
00049
00050 #endif
00051

```

22.103 code/MCAL/MDIO/MDIO_Private.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define PORTA *((volatile u8*)0x3B)
- #define DDRA *((volatile u8*)0x3A)
- #define PINA *((volatile u8*)0x39)
- #define PORTB *((volatile u8*)0x38)
- #define DDRB *((volatile u8*)0x37)
- #define PINB *((volatile u8*)0x36)
- #define PORTC *((volatile u8*)0x35)
- #define DDRC *((volatile u8*)0x34)

- #define **PINC** *((volatile u8*)0x33)
- #define **PORTD** *((volatile u8*)0x32)
- #define **DDRD** *((volatile u8*)0x31)
- #define **PIND** *((volatile u8*)0x30)

22.103.1 Macro Definition Documentation

22.103.1.1 **DDRA** #define DDRA *((volatile u8*)0x3A)

Definition at line 5 of file [MDIO_Private.h](#).

22.103.1.2 **DDRB** #define DDRB *((volatile u8*)0x37)

Definition at line 9 of file [MDIO_Private.h](#).

22.103.1.3 **DDRC** #define DDRC *((volatile u8*)0x34)

Definition at line 13 of file [MDIO_Private.h](#).

22.103.1.4 **DDRD** #define DDRD *((volatile u8*)0x31)

Definition at line 17 of file [MDIO_Private.h](#).

22.103.1.5 **PINA** #define PINA *((volatile u8*)0x39)

Definition at line 6 of file [MDIO_Private.h](#).

22.103.1.6 **PINB** #define PINB *((volatile u8*)0x36)

Definition at line 10 of file [MDIO_Private.h](#).

22.103.1.7 **PINC** #define PINC *((volatile u8*)0x33)

Definition at line 14 of file [MDIO_Private.h](#).

22.103.1.8 PIND #define PIND *((volatile u8*)0x30)

Definition at line 18 of file [MDIO_Private.h](#).

22.103.1.9 PORTA #define PORTA *((volatile u8*)0x3B)

Definition at line 4 of file [MDIO_Private.h](#).

22.103.1.10 PORTB #define PORTB *((volatile u8*)0x38)

Definition at line 8 of file [MDIO_Private.h](#).

22.103.1.11 PORTC #define PORTC *((volatile u8*)0x35)

Definition at line 12 of file [MDIO_Private.h](#).

22.103.1.12 PORTD #define PORTD *((volatile u8*)0x32)

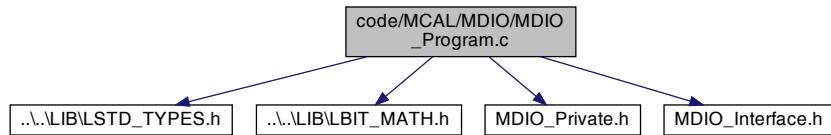
Definition at line 16 of file [MDIO_Private.h](#).

22.104 MDIO_Private.h

```
00001 #ifndef MDIO_PRIVATE_H_
00002 #define MDIO_PRIVATE_H_
00003
00004 #define PORTA      *((volatile u8*)0x3B)
00005 #define DDRA      *((volatile u8*)0x3A)
00006 #define PINA      *((volatile u8*)0x39)
00007
00008 #define PORTB      *((volatile u8*)0x38)
00009 #define DDRB      *((volatile u8*)0x37)
00010 #define PINB      *((volatile u8*)0x36)
00011
00012 #define PORTC      *((volatile u8*)0x35)
00013 #define DDRC      *((volatile u8*)0x34)
00014 #define PINC      *((volatile u8*)0x33)
00015
00016 #define PORTD      *((volatile u8*)0x32)
00017 #define DDRD      *((volatile u8*)0x31)
00018 #define PIND      *((volatile u8*)0x30)
00019
00020#endif
```

22.105 code/MCAL/MDIO/MDIO_Program.c File Reference

```
#include "..\..\LIB\LSTD_TYPES.h"
#include "..\..\LIB\LBIT_MATH.h"
#include "MDIO_Private.h"
#include "MDIO_Interface.h"
Include dependency graph for MDIO_Program.c:
```



Functions

- `Error_State MDIO_Error_State_SetPinDirection (u8 Copy_u8PinNumber, u8 Copy_u8PortNumber, u8 Copy_u8PinDirection)`
- `Error_State MDIO_Error_State_SetPortDirection (u8 Copy_u8PortNumber, u8 Copy_u8PortDirection)`
- `Error_State MDIO_Error_State_SetPinValue (u8 Copy_u8PinNumber, u8 Copy_u8PortNumber, u8 Copy_u8PinValue)`
- `Error_State MDIO_Error_State_SetPortValue (u8 Copy_u8PortNumber, u8 Copy_u8PortValue)`
- `Error_State MDIO_Error_State_GetPinValue (u8 Copy_u8PinNumber, u8 Copy_u8PortNumber, u8 *P_u8PinValue)`
- `Error_State MDIO_Error_State_SetNippleValue (u8 Copy_u8PinStart, u8 Copy_u8PortNumber, u8 Copy_u8Value)`

22.105.1 Function Documentation

22.105.1.1 MDIO_Error_State_GetPinValue() `Error_State MDIO_Error_State_GetPinValue (`
`u8 Copy_u8PinNumber,`
`u8 Copy_u8PortNumber,`
`u8 * P_u8PinValue)`

Definition at line 220 of file [MDIO_Program.c](#).

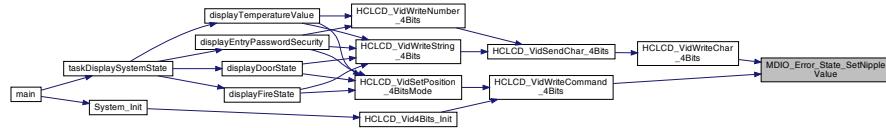
Here is the caller graph for this function:



22.105.1.2 MDIO_Error_State_SetNippleValue() `Error_State MDIO_Error_State_SetNippleValue (u8 Copy_u8PinStart,
u8 Copy_u8PortNumber,
u8 Copy_u8Value)`

Definition at line 255 of file [MDIO_Program.c](#).

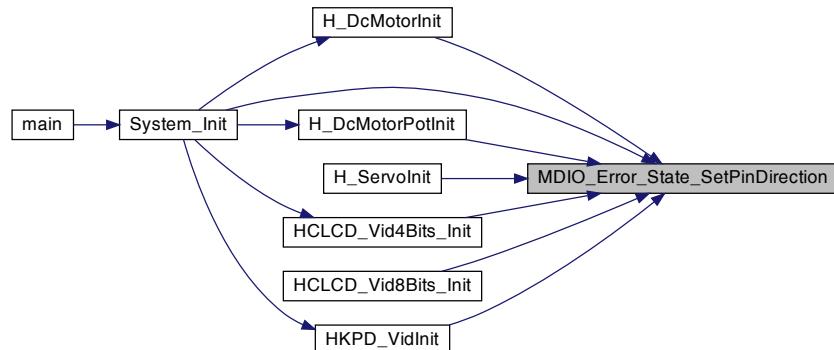
Here is the caller graph for this function:



22.105.1.3 MDIO_Error_State_SetPinDirection() `Error_State MDIO_Error_State_SetPinDirection (u8 Copy_u8PinNumber,
u8 Copy_u8PortNumber,
u8 Copy_u8PinDirection)`

Definition at line 9 of file [MDIO_Program.c](#).

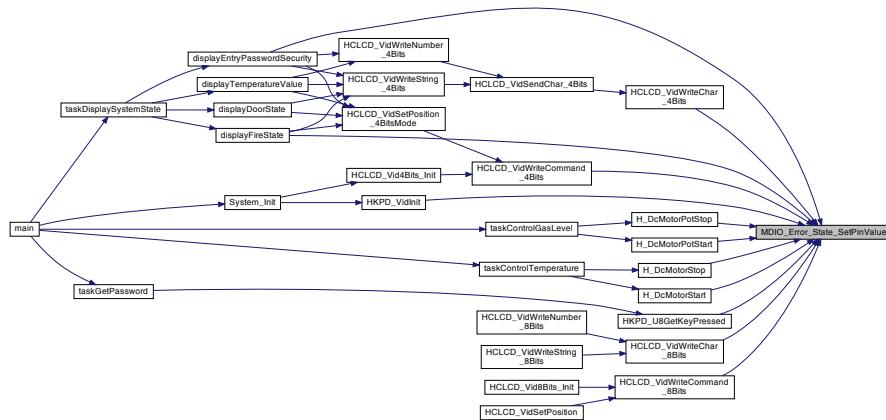
Here is the caller graph for this function:



22.105.1.4 MDIO_Error_State_SetPinValue() `Error_State MDIO_Error_State_SetPinValue (u8 Copy_u8PinNumber,
u8 Copy_u8PortNumber,
u8 Copy_u8PinValue)`

Definition at line 117 of file [MDIO_Program.c](#).

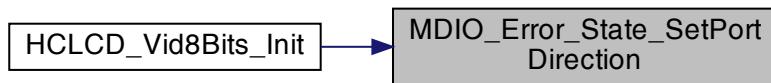
Here is the caller graph for this function:



22.105.1.5 MDIO_Error_State_SetPortDirection() `Error_State` MDIO_Error_State_SetPortDirection (
`u8 Copy_u8PortNumber,`
`u8 Copy_u8PortDirection)`

Definition at line 85 of file [MDIO_Program.c](#).

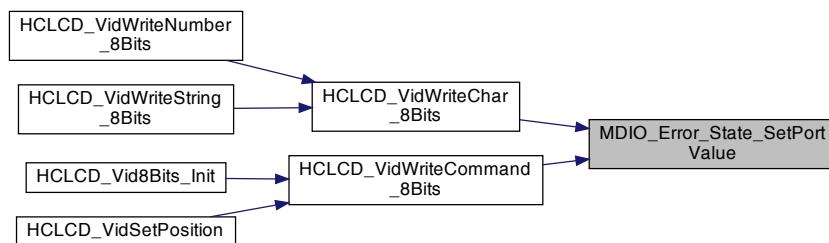
Here is the caller graph for this function:



22.105.1.6 MDIO_Error_State_SetPortValue() `Error_State` MDIO_Error_State_SetPortValue (
`u8 Copy_u8PortNumber,`
`u8 Copy_u8PortValue)`

Definition at line 195 of file [MDIO_Program.c](#).

Here is the caller graph for this function:



22.106 MDIO_Program.c

```

00001
00002 #include"..\..\LIB\LSTD_TYPES.h"
00003 #include"..\..\LIB\LBIT_MATH.h"
00004
00005 #include"MDIO_Private.h"
00006 #include"MDIO_Interface.h"
00007
00008 /*Set Pin Direction Function */
00009 Error_State MDIO_Error_State_SetPinDirection(u8 Copy_u8PinNumber,u8 Copy_u8PortNumber,u8
00010 Copy_u8PinDirection)
00010 {
00011     Error_State LOC_Error_State_ReturnState=OK;
00012     if((Copy_u8PinNumber>=0)&&(Copy_u8PinNumber<8))
00013     {
00014         switch(Copy_u8PortNumber)
00015         {
00016             case MDIO_PORTA:
00017                 if(Copy_u8PinDirection==PIN_OUTPUT)
00018                 {
00019                     SET_BIT(DDRA,Copy_u8PinNumber);
00020                 }
00021                 else if(Copy_u8PinDirection==PIN_INPUT)
00022                 {
00023                     CLR_BIT(DDRA,Copy_u8PinNumber);
00024                 }
00025             else
00026             {
00027                 LOC_Error_State_ReturnState=NOK;
00028             }
00029             break;
00030             case MDIO_PORTB:
00031                 if(Copy_u8PinDirection==PIN_OUTPUT)
00032                 {
00033                     SET_BIT(DDRB,Copy_u8PinNumber);
00034                 }
00035                 else if(Copy_u8PinDirection==PIN_INPUT)
00036                 {
00037                     CLR_BIT(DDRB,Copy_u8PinNumber);
00038                 }
00039             else
00040             {
00041                 LOC_Error_State_ReturnState=NOK;
00042             }
00043             break;
00044             case MDIO_PORTC:
00045                 if(Copy_u8PinDirection==PIN_OUTPUT)
00046                 {
00047                     SET_BIT(DDRC,Copy_u8PinNumber);
00048                 }
00049                 else if(Copy_u8PinDirection==PIN_INPUT)
00050                 {
00051                     CLR_BIT(DDRC,Copy_u8PinNumber);
00052                 }
00053             else
00054             {
00055                 LOC_Error_State_ReturnState=NOK;
00056             }
00057             break;
00058             case MDIO_PORTD:
00059                 if(Copy_u8PinDirection==PIN_OUTPUT)
00060                 {
00061                     SET_BIT(DDRD,Copy_u8PinNumber);
00062                 }
00063                 else if(Copy_u8PinDirection==PIN_INPUT)
00064                 {
00065                     CLR_BIT(DDRD,Copy_u8PinNumber);
00066                 }
00067             else
00068             {
00069                 LOC_Error_State_ReturnState=NOK;
00070             }
00071             break;
00072         default:
00073             LOC_Error_State_ReturnState=NOK;
00074             break;
00075     }
00076 }
00077 else
00078 {
00079     LOC_Error_State_ReturnState=NOK;
00080 }
00081
00082 return LOC_Error_State_ReturnState;
00083 }
00084 /*Set Port Direction Function */

```

```
00085 Error_State MDIO_Error_State_SetPortDirection(u8 Copy_u8PortNumber,u8 Copy_u8PortDirection)
00086 {
00087     Error_State LOC_Error_State_ReturnState=OK;
00088     if((Copy_u8PortDirection==PORT_OUTPUT) || (Copy_u8PortDirection==PORT_INPUT))
00089     {
00090         switch(Copy_u8PortNumber)
00091         {
00092             case MDIO_PORTA:
00093                 DDRA=Copy_u8PortDirection;
00094                 break;
00095             case MDIO_PORTB:
00096                 DDRB=Copy_u8PortDirection;
00097                 break;
00098             case MDIO_PORTC:
00099                 DDRC=Copy_u8PortDirection;
00100                 break;
00101             case MDIO_PORTD:
00102                 DDRD=Copy_u8PortDirection;
00103                 break;
00104             default:
00105                 LOC_Error_State_ReturnState=NOK;
00106                 break;
00107         }
00108     }
00109     else
00110     {
00111         LOC_Error_State_ReturnState=NOK;
00112     }
00113     return LOC_Error_State_ReturnState;
00114 }
00115
00116 /*Set Pin Value Function */
00117 Error_State MDIO_Error_State_SetPinValue(u8 Copy_u8PinNumber,u8 Copy_u8PortNumber,u8 Copy_u8PinValue)
00118 {
00119     Error_State LOC_Error_State_ReturnState=OK;
00120
00121     if((Copy_u8PinNumber>=0) && (Copy_u8PinNumber<8) )
00122     {
00123         switch(Copy_u8PortNumber)
00124         {
00125             case MDIO_PORTA:
00126                 if(Copy_u8PinValue==PIN_HIGH)
00127                 {
00128                     SET_BIT(PORTA,Copy_u8PinNumber);
00129                 }
00130                 else if(Copy_u8PinValue==PIN_LOW)
00131                 {
00132                     CLR_BIT(PORTA,Copy_u8PinNumber);
00133                 }
00134                 else
00135                 {
00136                     LOC_Error_State_ReturnState=NOK;
00137                 }
00138                 break;
00139             case MDIO_PORTB:
00140                 if(Copy_u8PinValue==PIN_HIGH)
00141                 {
00142                     SET_BIT(PORTB,Copy_u8PinNumber);
00143                 }
00144                 else if(Copy_u8PinValue==PIN_LOW)
00145                 {
00146                     CLR_BIT(PORTB,Copy_u8PinNumber);
00147                 }
00148                 else
00149                 {
00150                     LOC_Error_State_ReturnState=NOK;
00151                 }
00152                 break;
00153             case MDIO_PORTC:
00154                 if(Copy_u8PinValue==PIN_HIGH)
00155                 {
00156                     SET_BIT(PORTC,Copy_u8PinNumber);
00157                 }
00158                 else if(Copy_u8PinValue==PIN_LOW)
00159                 {
00160                     CLR_BIT(PORTC,Copy_u8PinNumber);
00161                 }
00162                 else
00163                 {
00164                     LOC_Error_State_ReturnState=NOK;
00165                 }
00166                 break;
00167             case MDIO_PORTD:
00168                 if(Copy_u8PinValue==PIN_HIGH)
00169                 {
00170                     SET_BIT(PORTD,Copy_u8PinNumber);
00171                 }
00172         }
00173     }
00174 }
```

```

00172         else if(Copy_u8PinValue==PIN_LOW)
00173     {
00174         CLR_BIT(PORTD,Copy_u8PinNumber);
00175     }
00176     else
00177     {
00178         LOC_Error_State_ReturnState=NOK;
00179     }
00180     break;
00181 default:
00182     LOC_Error_State_ReturnState=NOK;
00183     break;
00184 }
00185 }
00186 else
00187 {
00188     LOC_Error_State_ReturnState=NOK;
00189 }
00190
00191 return LOC_Error_State_ReturnState;
00192 }

00193 /*Set Port Value Function */
00194 Error_State MDIO_Error_State_SetPortValue(u8 Copy_u8PortNumber,u8 Copy_u8PortValue)
00195 {
00196     Error_State LOC_Error_State_ReturnState=OK;
00197     switch(Copy_u8PortNumber)
00198     {
00199         case MDIO_PORTA:
00200             PORTA=Copy_u8PortValue;
00201             break;
00202         case MDIO_PORTB:
00203             PORTB=Copy_u8PortValue;
00204             break;
00205         case MDIO_PORTC:
00206             PORTC=Copy_u8PortValue;
00207             break;
00208         case MDIO_PORTD:
00209             PORTD=Copy_u8PortValue;
00210             break;
00211         default:
00212             LOC_Error_State_ReturnState=NOK;
00213             break;
00214     }
00215     return LOC_Error_State_ReturnState;
00216 }
00217 }

00218 /*Get Pin Value Function */
00219 Error_State MDIO_Error_State_GetPinValue(u8 Copy_u8PinNumber,u8 Copy_u8PortNumber,u8* P_u8PinValue)
00220 {
00221     Error_State LOC_Error_State_ReturnState=OK;
00222
00223     if(((Copy_u8PinNumber>=0) && (Copy_u8PinNumber<8)) && (P_u8PinValue!=NULL_POINTER))
00224     {
00225         switch(Copy_u8PortNumber)
00226         {
00227             case MDIO_PORTA:
00228                 *P_u8PinValue=GET_BIT(PINA,Copy_u8PinNumber);
00229                 break;
00230             case MDIO_PORTB:
00231                 *P_u8PinValue=GET_BIT(PINB,Copy_u8PinNumber);
00232                 break;
00233             case MDIO_PORTC:
00234                 *P_u8PinValue=GET_BIT(PINC,Copy_u8PinNumber);
00235                 break;
00236             case MDIO_PORTD:
00237                 *P_u8PinValue=GET_BIT(PIND,Copy_u8PinNumber);
00238                 break;
00239             default:
00240                 LOC_Error_State_ReturnState=NOK;
00241                 break;
00242         }
00243     }
00244     else
00245     {
00246         LOC_Error_State_ReturnState=NOK;
00247     }
00248 }
00249
00250 return LOC_Error_State_ReturnState;
00251 }

00252 /*Set Nipple Direction Function */
00253
00254 /*Set Nipple Values Function */
00255 Error_State MDIO_Error_State_SetNippleValue(u8 Copy_u8PinStart,u8 Copy_u8PortNumber,u8 Copy_u8Value)
00256 {
00257     Error_State LOC_Error_State_ReturnState=OK;
00258     if(Copy_u8PinStart<=4)

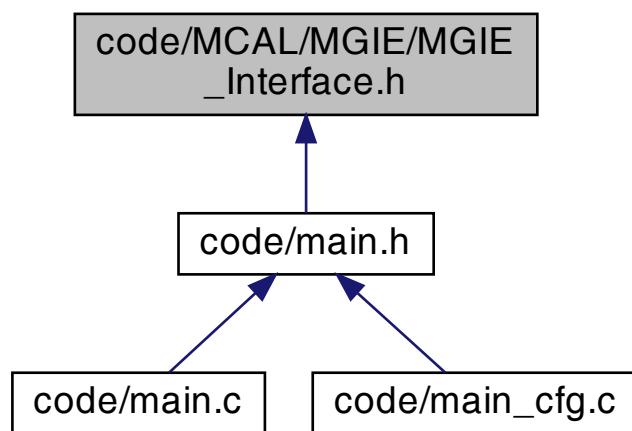
```

```

00259     {
00260         switch(Copy_u8PortNumber)
00261         {
00262             case MDIO_PORTA:
00263                 /*for(u8 i=Copy_u8PinStart;i<=7;i++)
00264                 {
00265                     CLR_BIT(PORTA,i);
00266                 }*/
00267                 PORTA&=(~(0x0F<<Copy_u8PinStart));
00268                 PORTA|=Copy_u8Value;
00269                 break;
00270             case MDIO_PORTB:
00271                 /*for(u8 i=Copy_u8PinStart;i<=Copy_u8PinEnd;i++)
00272                 {
00273                     CLR_BIT(PORTA,i);
00274                 }*/
00275                 PORTB&=(~(0x0F<<Copy_u8PinStart));
00276                 PORTB|=Copy_u8Value;
00277                 break;
00278             case MDIO_PORTC:
00279                 /*for(u8 i=Copy_u8PinStart;i<=Copy_u8PinEnd;i++)
00280                 {
00281                     CLR_BIT(PORTA,i);
00282                 }*/
00283                 PORTC&=(~(0x0F<<Copy_u8PinStart));
00284                 PORTC|=Copy_u8Value;
00285                 break;
00286             case MDIO_PORTD:
00287                 /*for(u8 i=Copy_u8PinStart;i<=Copy_u8PinEnd;i++)
00288                 {
00289                     CLR_BIT(PORTA,i);
00290                 }*/
00291                 PORTD&=(~(0x0F<<Copy_u8PinStart));
00292                 PORTD|=Copy_u8Value;
00293                 break;
00294             default:
00295                 LOC_Error_State_ReturnState=NOK;
00296             }
00297         }
00298     else
00299     {
00300         LOC_Error_State_ReturnState=NOK;
00301     }
00302     return LOC_Error_State_ReturnState;
00303 }
```

22.107 code/MCAL/MGIE/MGIE_Interface.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void MGIE_VidEnable (void)
- void MGIE_VidDisable (void)

22.107.1 Function Documentation

22.107.1.1 MGIE_VidDisable() void MGIE_VidDisable (void)

Definition at line 19 of file MGIE_Program.c.

22.107.1.2 MGIE_VidEnable() void MGIE_VidEnable (void)

Definition at line 14 of file MGIE_Program.c.

Here is the caller graph for this function:

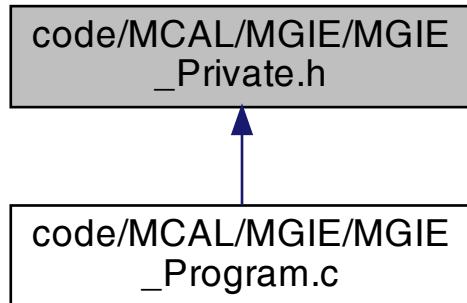


22.108 MGIE_Interface.h

```
00001 /*
00002 * MGIE_Interface.h
00003 *
00004 * Created on: Nov 15, 2021
00005 * Author: gerges
00006 */
00007
00008 #ifndef MCAL_MGIE_MGIE_INTERFACE_H_
00009 #define MCAL_MGIE_MGIE_INTERFACE_H_
00010
00011 /*GIE Enable Function*/
00012 void MGIE_VidEnable(void);
00013 /*GIE Disable Function*/
00014 void MGIE_VidDisable(void);
00015
00016 #endif /* MCAL_MGIE_MGIE_INTERFACE_H_ */
```

22.109 code/MCAL/MGIE/MGIE_Private.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define SREG *((volatile u8*)0x5F)

22.109.1 Macro Definition Documentation

22.109.1.1 SREG #define SREG *((volatile u8*)0x5F)

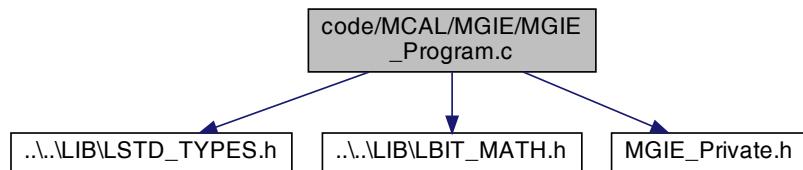
Definition at line 11 of file [MGIE_Private.h](#).

22.110 MGIE_Private.h

```
00001 /*
00002  * MGIE_Private.h
00003  *
00004  *   Created on: Nov 15, 2021
00005  *       Author: gerges
00006 */
00007
00008 #ifndef MCAL_MGIE_MGIE_PRIVATE_H_
00009 #define MCAL_MGIE_MGIE_PRIVATE_H_
00010
00011 #define SREG    *((volatile u8*)0x5F)
00012
00013 #endif /* MCAL_MGIE_MGIE_PRIVATE_H_ */
```

22.111 code/MCAL/MGIE/MGIE_Program.c File Reference

```
#include "..\..\LIB\LSTD_TYPES.h"
#include "..\..\LIB\LBIT_MATH.h"
#include "MGIE_Private.h"
Include dependency graph for MGIE_Program.c:
```



Functions

- void [MGIE_VidEnable](#) (void)
- void [MGIE_VidDisable](#) (void)

22.111.1 Function Documentation

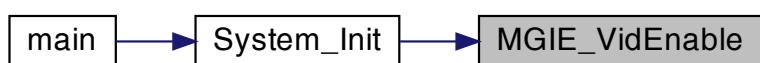
22.111.1.1 MGIE_VidDisable() void MGIE_VidDisable (void)

Definition at line 19 of file [MGIE_Program.c](#).

22.111.1.2 MGIE_VidEnable() void MGIE_VidEnable (void)

Definition at line 14 of file [MGIE_Program.c](#).

Here is the caller graph for this function:

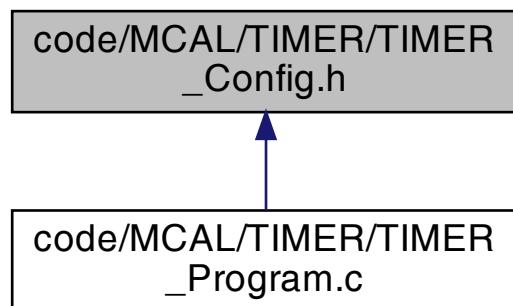


22.112 MGIE_Program.c

```
00001 /*  
00002 * MGIE_Program.c  
00003 *  
00004 * Created on: Nov 15, 2021  
00005 * Author: gerges  
00006 */  
00007  
00008 #include"..\..\LIB\LSTD_TYPES.h"  
00009 #include"..\..\LIB\LBIT_MATH.h"  
00010  
00011 #include"MGIE_Private.h"  
00012  
00013 /*GIE Enable Function*/  
00014 void MGIE_VidEnable(void)  
00015 {  
00016     SET_BIT(SREG,7);  
00017 }  
00018 /*GIE Disable Function*/  
00019 void MGIE_VidDisable(void)  
00020 {  
00021     CLR_BIT(SREG,7);  
00022 }  
00023
```

22.113 code/MCAL/TIMER/TIMER_Config.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define TIMER_SET_PRESCALER TIMER_256_PRESCALER
- #define TIMER0_SET_MODE TIMER0_CTC_MODE
- #define TIMER0_SET_CTC_INTERRUPT TIMER0_CTC_INTERRUPT_ENABLED
- #define TIMER0_SET_OC0_PIN_MODE TIMER0_OC0_PIN_DISCONNECTED
- #define TIMER0_SET_PWM_MODE TIMER0_NON_INVERTING_PWM

22.113.1 Macro Definition Documentation

22.113.1.1 TIMER0_SET_CTC_INTERRUPT #define TIMER0_SET_CTC_INTERRUPT TIMER0_CTC_INTERRUPT_ENABLED

Definition at line 37 of file [TIMER_Config.h](#).

22.113.1.2 TIMER0_SET_MODE #define TIMER0_SET_MODE TIMER0_CTC_MODE

Definition at line 30 of file [TIMER_Config.h](#).

22.113.1.3 TIMER0_SET_OCO_PIN_MODE #define TIMER0_SET_OCO_PIN_MODE TIMER0_OCO_PIN_DISCONNECTED

Definition at line 45 of file [TIMER_Config.h](#).

22.113.1.4 TIMER0_SET_PWM_MODE #define TIMER0_SET_PWM_MODE TIMER0_NON_INVERTING_PWM

Definition at line 51 of file [TIMER_Config.h](#).

22.113.1.5 TIMER_SET_PRESCALER #define TIMER_SET_PRESCALER TIMER_256_PRESCALER

Definition at line 21 of file [TIMER_Config.h](#).

22.114 TIMER_Config.h

```
00001 /*
00002  * TIMER_Config.h
00003  *
00004  * Created on: Nov 18, 2021
00005  * Author: gerges
00006 */
00007
00008 #ifndef MCAL_TIMER_TIMER_CONFIG_H_
00009 #define MCAL_TIMER_TIMER_CONFIG_H_
00010
00011 /*Timer Prescaler Options:
00012  * 0- TIMER_STOPPED
00013  * 1- TIMER_NO_PRESCALER
00014  * 2- TIMER_8_PRESCALER
00015  * 3- TIMER_64_PRESCALER
00016  * 4- TIMER_256_PRESCALER
00017  * 5- TIMER_1024_PRESCALER
00018  * 6- TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE
00019  * 7- TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE
00020 */
00021 #define TIMER_SET_PRESCALER TIMER_256_PRESCALER
00022
00023 /*TIMER0 Modes Options:
00024  * 1- TIMER0_NORMAL_MODE
00025  * 2- TIMER0_CTC_MODE
00026  * 3- TIMER0_PHASECORRECT_PWM_MODE
00027  * 4- TIMER0_FAST_PWM_MODE
00028 */
00029
00030 #define TIMER0_SET_MODE TIMER0_CTC_MODE
00031
00032 /*Timer0 CTC Interrupt Options:
00033  * 1- TIMER0_CTC_INTERRUPT_ENABLED
00034  * 2- TIMER0_CTC_INTERRUPT_DISABLED
```

```

00035  */
00036
00037 #define TIMER0_SET_CTC_INTERRUPT    TIMER0_CTC_INTERRUPT_ENABLED
00038
00039 /*Timer0 CTC OCO PIN Options:
00040 * 1- TIMER0_OCO_PIN_DISCONNECTED
00041 * 2- TIMER0_OCO_PIN_TOGGLE
00042 * 3- TIMER0_OCO_PIN_SET
00043 * 4- TIMER0_OCO_PIN_CLR
00044 */
00045 #define TIMER0_SET_OCO_PIN_MODE    TIMER0_OCO_PIN_DISCONNECTED
00046
00047 /*Timer0 PWM MODE Options:
00048 * 1-TIMER0_NON_INVERTING_PWM
00049 * 2- TIMER0_INVERTING_PWM
00050 */
00051 #define TIMER0_SET_PWM_MODE    TIMER0_NON_INVERTING_PWM
00052
00053
00054
00055
00056
00057
00058 #endif /* MCAL_TIMER_TIMER_CONFIG_H_ */

```

22.115 code/MCAL/TIMER/TIMER_Interface.h File Reference

Functions

- void [TIMER0_VidInit](#) (void)
- void [TIMER0_VidSetPreload](#) ([u8](#) Copy_u8Preload)
- void [TIMER0_VidSetCTCValue](#) ([u8](#) Copy_u8CTCValue)
- void [TIMER0_VidOVF_SetCallBack](#) (void(*Copy_VidCallBack)(void))
- void [TIMER0_VidCTC_SetCallBack](#) (void(*Copy_VidCallBack)(void))

22.115.1 Function Documentation

22.115.1.1 [TIMER0_VidCTC_SetCallBack\(\)](#) void TIMER0_VidCTC_SetCallBack (void(*)(void) Copy_VidCallBack)

Definition at line 98 of file [TIMER_Program.c](#).

22.115.1.2 [TIMER0_VidInit\(\)](#) void TIMER0_VidInit (void)

Definition at line 16 of file [TIMER_Program.c](#).

22.115.1.3 [TIMER0_VidOVF_SetCallBack\(\)](#) void TIMER0_VidOVF_SetCallBack (void(*)(void) Copy_VidCallBack)

Definition at line 94 of file [TIMER_Program.c](#).

22.115.1.4 TIMER0_VidSetCTCValue() void TIMER0_VidSetCTCValue (
 8 Copy_u8CTCValue)

Definition at line 90 of file [TIMER_Program.c](#).

22.115.1.5 TIMER0_VidSetPreload() void TIMER0_VidSetPreload (
 8 Copy_u8Preload)

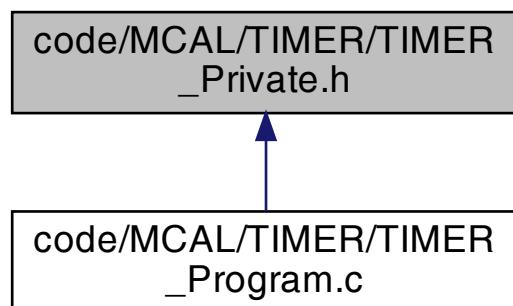
Definition at line 85 of file [TIMER_Program.c](#).

22.116 TIMER_Interface.h

```
00001 /*
00002  * TIMER_Interface.h
00003 *
00004  * Created on: Nov 18, 2021
00005  * Author: gerges
00006 */
00007
00008 #ifndef MCAL_TIMER_TIMER_INTERFACE_H_
00009 #define MCAL_TIMER_TIMER_INTERFACE_H_
00010
00011 /*Timer0 Initialization*/
00012 void TIMER0_VidInit(void);
00013 /*Set Preload Function*/
00014 void TIMER0_VidSetPreload(u8 Copy_u8Preload);
00015 /*Set CTC Value Function*/
00016 void TIMER0_VidSetCTCValue(u8 Copy_u8CTCValue);
00017
00018 void TIMER0_VidOVF_SetCallBack(void(*Copy_VidCallBack)(void));
00019 void TIMER0_VidCTC_SetCallBack(void(*Copy_VidCallBack)(void));
00020
00021 #endif /* MCAL_TIMER_TIMER_INTERFACE_H_ */
```

22.117 code/MCAL/TIMER/TIMER_Private.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define TCCR0 *((volatile u8*)0x53)
- #define TIMSK *((volatile u8*)0x59)
- #define TCNT0 *((volatile u8*)0x52)
- #define OCR0 *((volatile u8*)0x5C)
- #define TIMER_STOPPED 0
- #define TIMER_NO_PRESCALER 1
- #define TIMER_8_PRESCALER 2
- #define TIMER_64_PRESCALER 3
- #define TIMER_256_PRESCALER 4
- #define TIMER_1024_PRESCALER 5
- #define TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE 6
- #define TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE 7
- #define TIMER0_NORMAL_MODE 1
- #define TIMER0_CTC_MODE 2
- #define TIMER0_PHASECORRECT_PWM_MODE 3
- #define TIMER0_FAST_PWM_MODE 4
- #define TIMER0_CTC_INTERRUPT_ENABLED 0
- #define TIMER0_CTC_INTERRUPT_DISABLED 1
- #define TIMER0_OC0_PIN_DISCONNECTED 0
- #define TIMER0_OC0_PIN_TOGGLE 1
- #define TIMER0_OC0_PIN_SET 2
- #define TIMER0_OC0_PIN_CLR 3

22.117.1 Macro Definition Documentation

22.117.1.1 OCR0 #define OCR0 *((volatile u8*)0x5C)

Definition at line 14 of file [TIMER_Private.h](#).

22.117.1.2 TCCR0 #define TCCR0 *((volatile u8*)0x53)

Definition at line 11 of file [TIMER_Private.h](#).

22.117.1.3 TCNT0 #define TCNT0 *((volatile u8*)0x52)

Definition at line 13 of file [TIMER_Private.h](#).

22.117.1.4 TIMER0_CTC_INTERRUPT_DISABLED #define TIMER0_CTC_INTERRUPT_DISABLED 1

Definition at line 33 of file [TIMER_Private.h](#).

22.117.1.5 TIMER0_CTC_INTERRUPT_ENABLED #define TIMER0_CTC_INTERRUPT_ENABLED 0

Definition at line 32 of file [TIMER_Private.h](#).

22.117.1.6 TIMER0_CTC_MODE #define TIMER0_CTC_MODE 2

Definition at line 28 of file [TIMER_Private.h](#).

22.117.1.7 TIMER0_FAST_PWM_MODE #define TIMER0_FAST_PWM_MODE 4

Definition at line 30 of file [TIMER_Private.h](#).

22.117.1.8 TIMER0_NORMAL_MODE #define TIMER0_NORMAL_MODE 1

Definition at line 27 of file [TIMER_Private.h](#).

22.117.1.9 TIMER0_OC0_PIN_CLR #define TIMER0_OC0_PIN_CLR 3

Definition at line 39 of file [TIMER_Private.h](#).

22.117.1.10 TIMER0_OC0_PIN_DISCONNECTED #define TIMER0_OC0_PIN_DISCONNECTED 0

Definition at line 36 of file [TIMER_Private.h](#).

22.117.1.11 TIMER0_OC0_PIN_SET #define TIMER0_OC0_PIN_SET 2

Definition at line 38 of file [TIMER_Private.h](#).

22.117.1.12 TIMER0_OC0_PIN_TOGGLE #define TIMER0_OC0_PIN_TOGGLE 1

Definition at line 37 of file [TIMER_Private.h](#).

22.117.1.13 TIMER0_PHASECORRECT_PWM_MODE #define TIMER0_PHASECORRECT_PWM_MODE 3

Definition at line 29 of file [TIMER_Private.h](#).

22.117.1.14 TIMER_1024_PRESCALER #define TIMER_1024_PRESCALER 5

Definition at line 22 of file [TIMER_Private.h](#).

22.117.1.15 TIMER_256_PRESCALER #define TIMER_256_PRESCALER 4

Definition at line 21 of file [TIMER_Private.h](#).

22.117.1.16 TIMER_64_PRESCALER #define TIMER_64_PRESCALER 3

Definition at line 20 of file [TIMER_Private.h](#).

22.117.1.17 TIMER_8_PRESCALER #define TIMER_8_PRESCALER 2

Definition at line 19 of file [TIMER_Private.h](#).

22.117.1.18 TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE #define TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE 6

Definition at line 23 of file [TIMER_Private.h](#).

22.117.1.19 TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE #define TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE 7

Definition at line 24 of file [TIMER_Private.h](#).

22.117.1.20 TIMER_NO_PRESCALER #define TIMER_NO_PRESCALER 1

Definition at line 18 of file [TIMER_Private.h](#).

22.117.1.21 TIMER_STOPPED #define TIMER_STOPPED 0

Definition at line 17 of file [TIMER_Private.h](#).

22.117.1.22 TIMSK #define TIMSK *((volatile u8*)0x59)

Definition at line 12 of file [TIMER_Private.h](#).

22.118 TIMER_Private.h

```

00001 /*
00002  * TIMER_Private.h
00003 *
00004  * Created on: Nov 18, 2021
00005  * Author: gerges
00006 */
00007
00008 #ifndef MCAL_TIMER_TIMER_PRIVATE_H_
00009 #define MCAL_TIMER_TIMER_PRIVATE_H_
00010
00011 #define TCCR0      *((volatile u8*)0x53)
00012 #define TIMSK     *((volatile u8*)0x59)
00013 #define TCNT0     *((volatile u8*)0x52)
00014 #define OCR0      *((volatile u8*)0x5C)
00015
00016
00017 #define TIMER_STOPPED      0
00018 #define TIMER_NO_PRESCALER   1
00019 #define TIMER_8_PRESCALER    2
00020 #define TIMER_64_PRESCALER   3
00021 #define TIMER_256_PRESCALER  4
00022 #define TIMER_1024_PRESCALER 5
00023 #define TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE 6
00024 #define TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE   7
00025
00026
00027 #define TIMER0_NORMAL_MODE      1
00028 #define TIMER0_CTC_MODE        2
00029 #define TIMER0_PHASECORRECT_PWM_MODE 3
00030 #define TIMER0_FAST_PWM_MODE   4
00031
00032 #define TIMER0_CTC_INTERRUPT_ENABLED 0
00033 #define TIMER0_CTC_INTERRUPT_DISABLED 1
00034
00035
00036 #define TIMER0_OCO_PIN_DISCONNECTED 0
00037 #define TIMER0_OCO_PIN_TOGGLE    1
00038 #define TIMER0_OCO_PIN_SET       2
00039 #define TIMER0_OCO_PIN_CLR       3
00040
00041
00042 #endif /* MCAL_TIMER_TIMER_PRIVATE_H_ */

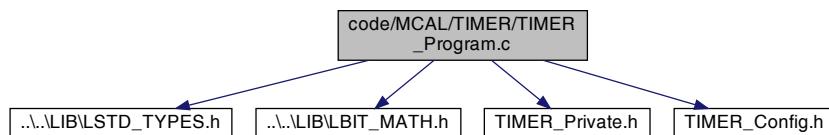
```

22.119 code/MCAL/TIMER/TIMER_Program.c File Reference

```

#include "..\..\LIB\LSTD_TYPES.h"
#include "..\..\LIB\LBIT_MATH.h"
#include "TIMER_Private.h"
#include "TIMER_Config.h"
Include dependency graph for TIMER_Program.c:

```



Functions

- void **TIMER0_VidInit** (void)
- void **TIMER0_VidSetPreload** (**u8** Copy_u8Preload)
- void **TIMER0_VidSetCTCValue** (**u8** Copy_u8CTCValue)
- void **TIMER0_VidOVF_SetCallBack** (void(*Copy_VidCallBack)(void))
- void **TIMER0_VidCTC_SetCallBack** (void(*Copy_VidCallBack)(void))
- void **__vector_11** (void)
- void **__vector_10** (void)

Variables

- void(* **TIMER0_CallBack**)(void)

22.119.1 Function Documentation

22.119.1.1 __vector_10() void __vector_10 (void)

Definition at line 107 of file [TIMER_Program.c](#).

22.119.1.2 __vector_11() void __vector_11 (void)

Definition at line 102 of file [TIMER_Program.c](#).

22.119.1.3 TIMER0_VidCTC_SetCallBack() void TIMER0_VidCTC_SetCallBack (void(*)(void) Copy_VidCallBack)

Definition at line 98 of file [TIMER_Program.c](#).

22.119.1.4 TIMER0_VidInit() void TIMER0_VidInit (void)

Definition at line 16 of file [TIMER_Program.c](#).

22.119.1.5 TIMER0_VidOVF_SetCallBack() void TIMER0_VidOVF_SetCallBack (void(*)(void) Copy_VidCallBack)

Definition at line 94 of file [TIMER_Program.c](#).

22.119.1.6 TIMER0_VidSetCTCValue() void TIMER0_VidSetCTCValue (u8 Copy_u8CTCValue)

Definition at line 90 of file [TIMER_Program.c](#).

22.119.1.7 TIMER0_VidSetPreload() void TIMER0_VidSetPreload (u8 Copy_u8Preload)

Definition at line 85 of file [TIMER_Program.c](#).

22.119.2 Variable Documentation

22.119.2.1 TIMER0_CallBack void(* TIMER0_CallBack) (void) (void)

Definition at line 13 of file [TIMER_Program.c](#).

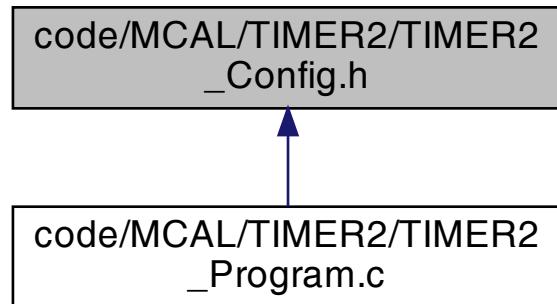
22.120 TIMER_Program.c

```
00001 /*
00002  * TIMER_Program.c
00003  *
00004  *   Created on: Nov 18, 2021
00005  *       Author: gerges
00006 */
00007
00008 #include"..\..\LIB\LSTD_TYPES.h"
00009 #include"..\..\LIB\LBIT_MATH.h"
00010 #include"TIMER_Private.h"
00011 #include"TIMER_Config.h"
00012
00013 void (*TIMER0_CallBack)(void);
00014
00015 /*Timer0 Initialization*/
00016 void TIMER0_VidInit(void)
00017 {
00018     /*Select the suitable Prescaler */
00019     TCCR0&=0xF8;
00020     TCCR0|=TIMER_SET_PRESCALER;
00021     /*Mode Select*/
00022 #if TIMER0_SET_MODE == TIMER0_NORMAL_MODE
00023     CLR_BIT(TCCR0,6);
00024     CLR_BIT(TCCR0,3);
00025     /*Timer Overflow Interrupt Enable*/
00026     SET_BIT(TIMSK,0);
00027
00028 #elif TIMER0_SET_MODE == TIMER0_CTC_MODE
00029     CLR_BIT(TCCR0,6);
00030     SET_BIT(TCCR0,3);
00031 #if TIMER0_SET_CTC_INTERRUPT == TIMER0_CTC_INTERRUPT_ENABLED
00032     SET_BIT(TIMSK,1);
00033 #elif TIMER0_SET_CTC_INTERRUPT == TIMER0_CTC_INTERRUPT_DISABLED
```

```
00034     CLR_BIT(TIMSK,1);
00035 #else
00036 #error "TIMER0 CTC Interrupt Mode is not valid..."
00037 #endif
00038 #if TIMER0_SET_OCO_PIN_MODE == TIMER0_OCO_PIN_DISCONNECTED
00039     CLR_BIT(TCCR0,5);
00040     CLR_BIT(TCCR0,4);
00041 #elif TIMER0_SET_OCO_PIN_MODE == TIMER0_OCO_PIN_TOGGLE
00042     CLR_BIT(TCCR0,5);
00043     SET_BIT(TCCR0,4);
00044 #elif TIMER0_SET_OCO_PIN_MODE == TIMER0_OCO_PIN_SET
00045     SET_BIT(TCCR0,5);
00046     SET_BIT(TCCR0,4);
00047 #elif TIMER0_SET_OCO_PIN_MODE == TIMER0_OCO_PIN_CLR
00048     SET_BIT(TCCR0,5);
00049     CLR_BIT(TCCR0,4);
00050 #else
00051 #error "TIMER0 CTC OCO Mode is not valid..."
00052 #endif
00053 #elif TIMER0_SET_MODE == TIMER0_PHASECORRECT_PWM_MODE
00054     SET_BIT(TCCR0,6);
00055     CLR_BIT(TCCR0,3);
00056 #if TIMER0_SET_PWM_MODE == TIMER0_NON_INVERTING_PWM
00057     SET_BIT(TCCR0,5);
00058     CLR_BIT(TCCR0,4);
00059
00060 #elif TIMER0_SET_PWM_MODE == TIMER0_INVERTING_PWM
00061     SET_BIT(TCCR0,5);
00062     SET_BIT(TCCR0,4);
00063 #else
00064 #error "PWM Mode is not valid..."
00065 #endif
00066 #elif TIMER0_SET_MODE == TIMER0_FAST_PWM_MODE
00067     SET_BIT(TCCR0,6);
00068     SET_BIT(TCCR0,3);
00069 #if TIMER0_SET_PWM_MODE == TIMER0_NON_INVERTING_PWM
00070     SET_BIT(TCCR0,5);
00071     CLR_BIT(TCCR0,4);
00072
00073 #elif TIMER0_SET_PWM_MODE == TIMER0_INVERTING_PWM
00074     SET_BIT(TCCR0,5);
00075     SET_BIT(TCCR0,4);
00076 #else
00077 #error "PWM Mode is not valid..."
00078 #endif
00079 #else
00080 #error "TIMER0 Mode is not valid..."
00081 #endif
00082
00083 }
00084 /*Set Preload Function*/
00085 void TIMER0_VidSetPreload(u8 Copy_u8Preload)
00086 {
00087     TCNT0=Copy_u8Preload;
00088 }
00089 /*Set CTC Value Function*/
00090 void TIMER0_VidSetCTCValue(u8 Copy_u8CTCValue)
00091 {
00092     OCRO=Copy_u8CTCValue;
00093 }
00094 void TIMER0_VidOVF_SetCallBack(void(*Copy_VidCallBack) (void))
00095 {
00096     TIMER0_CallBack=Copy_VidCallBack;
00097 }
00098 void TIMER0_VidCTC_SetCallBack(void(*Copy_VidCallBack) (void))
00099 {
00100     TIMER0_CallBack=Copy_VidCallBack;
00101 }
00102 void __vector_11(void)    __attribute__((signal));
00103 void __vector_11(void)
00104 {
00105     TIMER0_CallBack();
00106 }
00107 void __vector_10(void)    __attribute__((signal));
00108 void __vector_10(void)
00109 {
00110     TIMER0_CallBack();
00111 }
```

22.121 code/MCAL/TIMER2/TIMER2_Config.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define TIMER_SET_PRESCALER TIMER_256_PRESCALER
- #define TIMER2_SET_MODE TIMER2_FAST_PWM_MODE
- #define TIMER2_SET_CTC_INTERRUPT TIMER2_CTC_INTERRUPT_ENABLED
- #define TIMER2_SET_OC2_PIN_MODE TIMER0_OC2_PIN_DISCONNECTED
- #define TIMER2_SET_PWM_MODE TIMER2_NON_INVERTING_PWM

22.121.1 Macro Definition Documentation

22.121.1.1 TIMER2_SET_CTC_INTERRUPT #define TIMER2_SET_CTC_INTERRUPT TIMER2_CTC_INTERRUPT_ENABLED

Definition at line 30 of file [TIMER2_Config.h](#).

22.121.1.2 TIMER2_SET_MODE #define TIMER2_SET_MODE TIMER2_FAST_PWM_MODE

Definition at line 23 of file [TIMER2_Config.h](#).

22.121.1.3 TIMER2_SET_OC2_PIN_MODE #define TIMER2_SET_OC2_PIN_MODE TIMER0_OC2_PIN_DISCONNECTED

Definition at line 38 of file [TIMER2_Config.h](#).

22.121.1.4 TIMER2_SET_PWM_MODE #define TIMER2_SET_PWM_MODE TIMER2_NON_INVERTING_PWM

Definition at line 44 of file [TIMER2_Config.h](#).

22.121.1.5 TIMER_SET_PRESCALER #define TIMER_SET_PRESCALER TIMER_256_PRESCALER

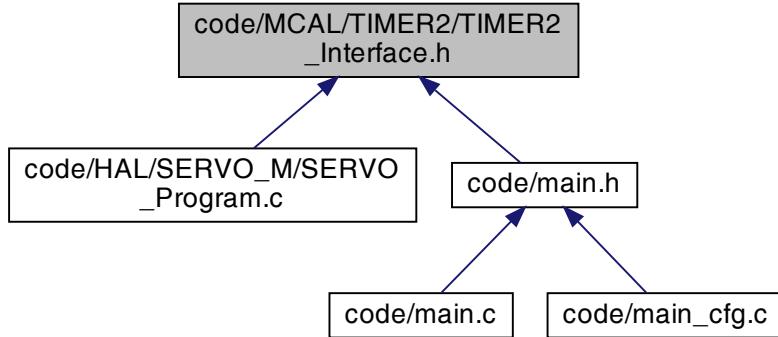
Definition at line 14 of file [TIMER2_Config.h](#).

22.122 TIMER2_Config.h

```
00001 #ifndef MCAL_TIMER2_TIMER2_CONFIG_H_
00002 #define MCAL_TIMER2_TIMER2_CONFIG_H_
00003
00004 /*Timer Prescaler Options:
00005 * 0- TIMER_STOPPED
00006 * 1- TIMER_NO_PRESCALER
00007 * 2- TIMER_8_PRESCALER
00008 * 3- TIMER_64_PRESCALER
00009 * 4- TIMER_256_PRESCALER
00010 * 5- TIMER_1024_PRESCALER
00011 * 6- TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE
00012 * 7- TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE
00013 */
00014 #define TIMER_SET_PRESCALER TIMER_256_PRESCALER
00015
00016 /*TIMER0 Modes Options:
00017 * 1- TIMER2_NORMAL_MODE
00018 * 2- TIMER2_CTC_MODE
00019 * 3- TIMER2_PHASECORRECT_PWM_MODE
00020 * 4- TIMER2_FAST_PWM_MODE
00021 */
00022
00023 #define TIMER2_SET_MODE TIMER2_FAST_PWM_MODE
00024
00025 /*Timer2 CTC Interrupt Options:
00026 * 1- TIMER2_CTC_INTERRUPT_ENABLED
00027 * 2- TIMER2_CTC_INTERRUPT_DISABLED
00028 */
00029
00030 #define TIMER2_SET_CTC_INTERRUPT TIMER2_CTC_INTERRUPT_ENABLED
00031
00032 /*Timer2 CTC OC2 PIN Options:
00033 * 1- TIMERO_OC2_PIN_DISCONNECTED
00034 * 2- TIMERO_OC2_PIN_TOGGLE
00035 * 3- TIMERO_OC2_PIN_SET
00036 * 4- TIMERO_OC2_PIN_CLR
00037 */
00038 #define TIMER2_SET_OC2_PIN_MODE TIMERO_OC2_PIN_DISCONNECTED
00039
00040 /*Timer0 PWM MODE Options:
00041 * 1-TIMER2_NON_INVERTING_PWM
00042 * 2- TIMER2_INVERTING_PWM
00043 */
00044 #define TIMER2_SET_PWM_MODE TIMER2_NON_INVERTING_PWM
00045
00046
00047
00048
00049
00050
00051 #endif /* MCAL_TIMER2_TIMER2_CONFIG_H_ */
```

22.123 code/MCAL/TIMER2/TIMER2_Interface.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void [TIMER2_VidInit](#) (void)
- void [TIMER2_VidSetPreload](#) ([u8](#) Copy_u8Preload)
- void [TIMER2_VidSetCTCValue](#) ([u8](#) Copy_u8CTCValue)
- void [TIMER2_VidOVF_SetCallBack](#) (void(*Copy_VidCallBack)(void))
- void [TIMER2_VidCTC_SetCallBack](#) (void(*Copy_VidCallBack)(void))
- void [M_Pwm2Init](#) (void)
- void [M_Pwm2SetDutyCycle](#) ([f64](#) f64_local_dutyCycle)
- void [M_Pwm2Start](#) (void)
- void [M_Pwm2Stop](#) (void)

22.123.1 Function Documentation

22.123.1.1 M_Pwm2Init()

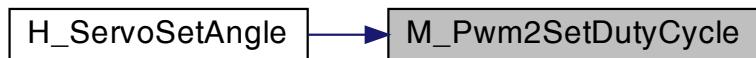
```
void M_Pwm2Init (
    void )
```

Definition at line [110](#) of file [TIMER2_Program.c](#).

```
22.123.1.2 M_Pwm2SetDutyCycle() void M_Pwm2SetDutyCycle (
    f64 local_dutyCycle )
```

Definition at line 117 of file [TIMER2_Program.c](#).

Here is the caller graph for this function:



```
22.123.1.3 M_Pwm2Start() void M_Pwm2Start (
    void )
```

Definition at line 121 of file [TIMER2_Program.c](#).

Here is the caller graph for this function:



```
22.123.1.4 M_Pwm2Stop() void M_Pwm2Stop (
    void )
```

Definition at line 132 of file [TIMER2_Program.c](#).

Here is the caller graph for this function:



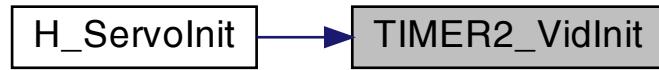
22.123.1.5 TIMER2_VidCTC_SetCallBack() void TIMER2_VidCTC_SetCallBack (void(*)(void) Copy_VidCallBack)

Definition at line 92 of file [TIMER2_Program.c](#).

22.123.1.6 TIMER2_VidInit() void TIMER2_VidInit (void)

Definition at line 10 of file [TIMER2_Program.c](#).

Here is the caller graph for this function:



22.123.1.7 TIMER2_VidOVF_SetCallBack() void TIMER2_VidOVF_SetCallBack (void(*)(void) Copy_VidCallBack)

Definition at line 88 of file [TIMER2_Program.c](#).

22.123.1.8 TIMER2_VidSetCTCValue() void TIMER2_VidSetCTCValue (u8 Copy_u8CTCValue)

Definition at line 84 of file [TIMER2_Program.c](#).

22.123.1.9 TIMER2_VidSetPreload() void TIMER2_VidSetPreload (u8 Copy_u8Preload)

Definition at line 79 of file [TIMER2_Program.c](#).

22.124 TIMER2_Interface.h

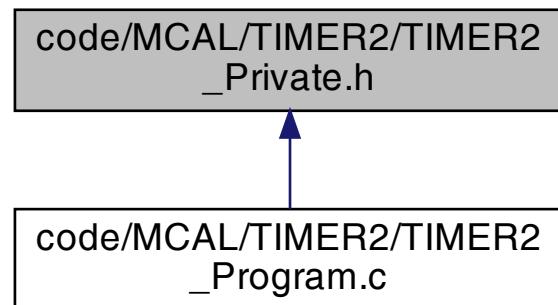
```

00001
00002
00003 #ifndef MCAL_TIMER_TIMER_INTERFACE_H_
00004 #define MCAL_TIMER_TIMER_INTERFACE_H_
00005
00006 /*Timer2 Initialization*/
00007 void TIMER2_VidInit(void);
00008 /*Set Preload Function*/
00009 void TIMER2_VidSetPreload(u8 Copy_u8Preload);
00010 /*Set CTC Value Function*/
00011 void TIMER2_VidSetCTCValue(u8 Copy_u8CTCValue);
00012
00013
00014 void TIMER2_VidOVF_SetCallBack(void(*Copy_VidCallBack) (void));
00015 void TIMER2_VidCTC_SetCallBack(void(*Copy_VidCallBack) (void));
00016
00017 /*INITIATE PWM*/
00018 void M_Pwm2Init(void);
00019
00020 /*SET DUTY CYCLE*/
00021 void M_Pwm2SetDutyCycle(f64 f64_local_dutyCycle);
00022
00023 void M_Pwm2Start(void);
00024 void M_Pwm2Stop(void);
00025 #endif /* MCAL_TIMER_TIMER_INTERFACE_H_ */

```

22.125 code/MCAL/TIMER2/TIMER2_Private.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define TCCR2 *((volatile u8*)0x45)
- #define TIMSK *((volatile u8*)0x59)
- #define TCNT2 *((volatile u8*)0x44)
- #define OCR2 *((volatile u8*)0x43)
- #define TIMER_STOPPED 0
- #define TIMER_NO_PRESCALER 1
- #define TIMER_8_PRESCALER 2
- #define TIMER_64_PRESCALER 3
- #define TIMER_256_PRESCALER 4
- #define TIMER_1024_PRESCALER 5
- #define TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE 6

- #define TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE 7
- #define TIMER2_NORMAL_MODE 1
- #define TIMER2_CTC_MODE 2
- #define TIMER2_PHASECORRECT_PWM_MODE 3
- #define TIMER2_FAST_PWM_MODE 4
- #define TIMER2_CTC_INTERRUPT_ENABLED 0
- #define TIMER2_CTC_INTERRUPT_DISABLED 1
- #define TIMER2_OC2_PIN_DISCONNECTED 0
- #define TIMER2_OC2_PIN_TOGGLE 1
- #define TIMER2_OC2_PIN_SET 2
- #define TIMER2_OC2_PIN_CLR 3

22.125.1 Macro Definition Documentation

22.125.1.1 OCR2 #define OCR2 *((volatile u8*)0x43)

Definition at line 7 of file [TIMER2_Private.h](#).

22.125.1.2 TCCR2 #define TCCR2 *((volatile u8*)0x45)

Definition at line 4 of file [TIMER2_Private.h](#).

22.125.1.3 TCNT2 #define TCNT2 *((volatile u8*)0x44)

Definition at line 6 of file [TIMER2_Private.h](#).

22.125.1.4 TIMER2_CTC_INTERRUPT_DISABLED #define TIMER2_CTC_INTERRUPT_DISABLED 1

Definition at line 26 of file [TIMER2_Private.h](#).

22.125.1.5 TIMER2_CTC_INTERRUPT_ENABLED #define TIMER2_CTC_INTERRUPT_ENABLED 0

Definition at line 25 of file [TIMER2_Private.h](#).

22.125.1.6 TIMER2_CTC_MODE #define TIMER2_CTC_MODE 2

Definition at line 21 of file [TIMER2_Private.h](#).

22.125.1.7 TIMER2_FAST_PWM_MODE #define TIMER2_FAST_PWM_MODE 4

Definition at line 23 of file [TIMER2_Private.h](#).

22.125.1.8 TIMER2_NORMAL_MODE #define TIMER2_NORMAL_MODE 1

Definition at line 20 of file [TIMER2_Private.h](#).

22.125.1.9 TIMER2_OC2_PIN_CLR #define TIMER2_OC2_PIN_CLR 3

Definition at line 32 of file [TIMER2_Private.h](#).

22.125.1.10 TIMER2_OC2_PIN_DISCONNECTED #define TIMER2_OC2_PIN_DISCONNECTED 0

Definition at line 29 of file [TIMER2_Private.h](#).

22.125.1.11 TIMER2_OC2_PIN_SET #define TIMER2_OC2_PIN_SET 2

Definition at line 31 of file [TIMER2_Private.h](#).

22.125.1.12 TIMER2_OC2_PIN_TOGGLE #define TIMER2_OC2_PIN_TOGGLE 1

Definition at line 30 of file [TIMER2_Private.h](#).

22.125.1.13 TIMER2_PHASECORRECT_PWM_MODE #define TIMER2_PHASECORRECT_PWM_MODE 3

Definition at line 22 of file [TIMER2_Private.h](#).

22.125.1.14 TIMER_1024_PRESCALER #define TIMER_1024_PRESCALER 5

Definition at line 15 of file [TIMER2_Private.h](#).

22.125.1.15 TIMER_256_PRESCALER #define TIMER_256_PRESCALER 4

Definition at line 14 of file [TIMER2_Private.h](#).

22.125.1.16 TIMER_64_PRESCALER #define TIMER_64_PRESCALER 3

Definition at line 13 of file [TIMER2_Private.h](#).

22.125.1.17 TIMER_8_PRESCALER #define TIMER_8_PRESCALER 2

Definition at line 12 of file [TIMER2_Private.h](#).

22.125.1.18 TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE #define TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE 6

Definition at line 16 of file [TIMER2_Private.h](#).

22.125.1.19 TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE #define TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE 7

Definition at line 17 of file [TIMER2_Private.h](#).

22.125.1.20 TIMER_NO_PRESCALER #define TIMER_NO_PRESCALER 1

Definition at line 11 of file [TIMER2_Private.h](#).

22.125.1.21 TIMER_STOPPED #define TIMER_STOPPED 0

Definition at line 10 of file [TIMER2_Private.h](#).

22.125.1.22 TIMSK #define TIMSK *((volatile u8*) 0x59)

Definition at line 5 of file [TIMER2_Private.h](#).

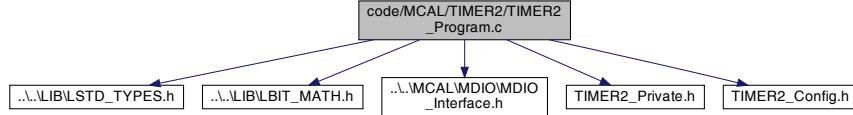
22.126 TIMER2_Private.h

```
00001 #ifndef MCAL_TIMER2_TIMER2_PRIVATE_H_
00002 #define MCAL_TIMER2_TIMER2_PRIVATE_H_
00003
00004 #define TCCR2      *((volatile u8*)0x45)
00005 #define TIMSK      *((volatile u8*)0x59)
00006 #define TCNT2      *((volatile u8*)0x44)
00007 #define OCR2      *((volatile u8*)0x43)
00008
00009
00010 #define TIMER_STOPPED      0
00011 #define TIMER_NO_PRESCALER  1
00012 #define TIMER_8_PRESCALER   2
00013 #define TIMER_64_PRESCALER  3
00014 #define TIMER_256_PRESCALER 4
00015 #define TIMER_1024_PRESCALER 5
00016 #define TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE 6
00017 #define TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE    7
00018
00019
00020 #define TIMER2_NORMAL_MODE      1
00021 #define TIMER2_CTC_MODE        2
00022 #define TIMER2_PHASECORRECT_PWM_MODE 3
00023 #define TIMER2_FAST_PWM_MODE   4
00024
00025 #define TIMER2_CTC_INTERRUPT_ENABLED 0
00026 #define TIMER2_CTC_INTERRUPT_DISABLED 1
00027
00028
00029 #define TIMER2_OC2_PIN_DISCONNECTED 0
00030 #define TIMER2_OC2_PIN_TOGGLE    1
00031 #define TIMER2_OC2_PIN_SET      2
00032 #define TIMER2_OC2_PIN_CLR      3
00033
00034
00035 #endif /* MCAL_TIMER_TIMER_PRIVATE_H_ */
```

22.127 code/MCAL/TIMER2/TIMER2_Program.c File Reference

```
#include "..\..\LIB\STD_TYPES.h"
#include "..\..\LIB\BIT_MATH.h"
#include "..\..\MCAL\MDIO\MDIO_Interface.h"
#include "TIMER2_Private.h"
#include "TIMER2_Config.h"
```

Include dependency graph for `TIMER2_Program.c`:



Functions

- void `TIMER2_VidInit` (void)
- void `TIMER2_VidSetPreload` (`u8 Copy_u8Preload`)
- void `TIMER2_VidSetCTCValue` (`u8 Copy_u8CTCValue`)
- void `TIMER2_VidOVF_SetCallBack` (`(void)(*Copy_VidCallBack)(void)`)

- void [TIMER2_VidCTC_SetCallBack](#) (void(*Copy_VidCallBack)(void))
- void [__vector_4](#) (void)
- void [__vector_5](#) (void)
- void [M_Pwm2Init](#) (void)
- void [M_Pwm2SetDutyCycle](#) ([f64](#) f64_local_dutyCycle)
- void [M_Pwm2Start](#) (void)
- void [M_Pwm2Stop](#) (void)

Variables

- void(* [TIMER2_CallBack](#))(void)

22.127.1 Function Documentation

22.127.1.1 [__vector_4\(\)](#) void __vector_4 (void)

Definition at line 98 of file [TIMER2_Program.c](#).

22.127.1.2 [__vector_5\(\)](#) void __vector_5 (void)

Definition at line 103 of file [TIMER2_Program.c](#).

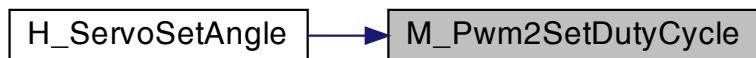
22.127.1.3 [M_Pwm2Init\(\)](#) void M_Pwm2Init (void)

Definition at line 110 of file [TIMER2_Program.c](#).

22.127.1.4 [M_Pwm2SetDutyCycle\(\)](#) void M_Pwm2SetDutyCycle ([f64](#) f64_local_dutyCycle)

Definition at line 117 of file [TIMER2_Program.c](#).

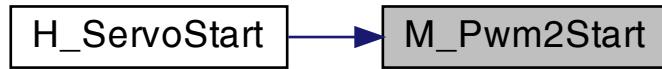
Here is the caller graph for this function:



```
22.127.1.5 M_Pwm2Start() void M_Pwm2Start (
    void )
```

Definition at line 121 of file [TIMER2_Program.c](#).

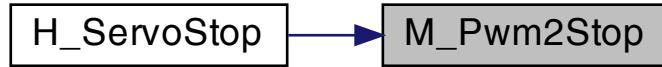
Here is the caller graph for this function:



```
22.127.1.6 M_Pwm2Stop() void M_Pwm2Stop (
    void )
```

Definition at line 132 of file [TIMER2_Program.c](#).

Here is the caller graph for this function:



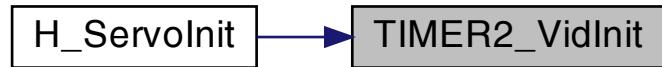
```
22.127.1.7 TIMER2_VidCTC_SetCallBack() void TIMER2_VidCTC_SetCallBack (
    void(*)(void) Copy_VidCallBack )
```

Definition at line 92 of file [TIMER2_Program.c](#).

22.127.1.8 TIMER2_VidInit() void TIMER2_VidInit (void)

Definition at line 10 of file [TIMER2_Program.c](#).

Here is the caller graph for this function:



22.127.1.9 TIMER2_VidOVF_SetCallBack() void TIMER2_VidOVF_SetCallBack (void(*)(void) Copy_VidCallBack)

Definition at line 88 of file [TIMER2_Program.c](#).

22.127.1.10 TIMER2_VidSetCTCValue() void TIMER2_VidSetCTCValue (u8 Copy_u8CTCValue)

Definition at line 84 of file [TIMER2_Program.c](#).

22.127.1.11 TIMER2_VidSetPreload() void TIMER2_VidSetPreload (u8 Copy_u8Preload)

Definition at line 79 of file [TIMER2_Program.c](#).

22.127.2 Variable Documentation

22.127.2.1 TIMER2_CallBack void(* TIMER2_CallBack) (void) (void)

Definition at line 7 of file [TIMER2_Program.c](#).

22.128 TIMER2_Program.c

```

00001 #include"..\..\LIB\LSTD_TYPES.h"
00002 #include"..\..\LIB\LBIT_MATH.h"
00003 #include"..\..\MCAL\MDIO\MDIO_Interface.h"
00004 #include"TIMER2_Private.h"
00005 #include"TIMER2_Config.h"
00006
00007 void (*TIMER2_CallBack) (void);
00008
00009 /*Timer2 Initialization*/
0010 void TIMER2_VidInit(void)
0011 {
0012     /*Select the suitable Prescaler */
0013     TCCR2&=0xF8;
0014     TCCR2|=TIMER_SET_PRESCALER;
0015     /*Mode Select*/
0016 #if TIMER_SET_MODE == TIMER2_NORMAL_MODE
0017     CLR_BIT(TCCR2,6);
0018     CLR_BIT(TCCR2,3);
0019     /*Timer Overflow Interrupt Enable*/
0020     SET_BIT(TIMSK,7);
0021
0022 #elif TIMER2_SET_MODE == TIMER2_CTC_MODE
0023     CLR_BIT(TCCR2,6);
0024     SET_BIT(TCCR2,3);
0025 #if TIMER2_SET_INTERRUPT == TIMER2_CTC_INTERRUPT_ENABLED
0026     SET_BIT(TIMSK,6);
0027 #elif TIMER2_SET_CTC_INTERRUPT == TIMER2_CTC_INTERRUPT_DISABLED
0028     CLR_BIT(TIMSK,6);
0029 #else
0030     #error "TIMER2 CTC Interrupt Mode is not valid..."
0031 #endif
0032 #if TIMER2_SET_OC2_PIN_MODE == TIMER2_OC2_PIN_DISCONNECTED
0033     CLR_BIT(TCCR2,5);
0034     CLR_BIT(TCCR2,4);
0035 #elif TIMER2_SET_OC2_PIN_MODE == TIMER2_OC2_PIN_TOGGLE
0036     CLR_BIT(TCCR2,5);
0037     SET_BIT(TCCR2,4);
0038 #elif TIMER2_SET_OC2_PIN_MODE == TIMER2_OC2_PIN_SET
0039     SET_BIT(TCCR2,5);
0040     SET_BIT(TCCR2,4);
0041 #elif TIMER2_SET_OC2_PIN_MODE == TIMER2_OC2_PIN_CLR
0042     SET_BIT(TCCR2,5);
0043     CLR_BIT(TCCR2,4);
0044 #else
0045     #error "TIMER2 CTC OC2 Mode is not valid..."
0046 #endif
0047 #elif TIMER2_SET_MODE == TIMER2_PHASECORRECT_PWM_MODE
0048     SET_BIT(TCCR2,6);
0049     CLR_BIT(TCCR2,3);
0050 #if TIMER2_SET_PWM_MODE == TIMER2_NON_INVERTING_PWM
0051     SET_BIT(TCCR2,5);
0052     CLR_BIT(TCCR2,4);
0053
0054 #elif TIMER2_SET_PWM_MODE == TIMER2_INVERTING_PWM
0055     SET_BIT(TCCR2,5);
0056     SET_BIT(TCCR2,4);
0057 #else
0058     #error "PWM Mode is not valid..."
0059 #endif
0060 #elif TIMER2_SET_MODE == TIMER2_FAST_PWM_MODE
0061     SET_BIT(TCCR2,6);
0062     SET_BIT(TCCR2,3);
0063 #if TIMER2_SET_PWM_MODE == TIMER2_NON_INVERTING_PWM
0064     SET_BIT(TCCR2,5);
0065     CLR_BIT(TCCR2,4);
0066
0067 #elif TIMER2_SET_PWM_MODE == TIMER2_INVERTING_PWM
0068     SET_BIT(TCCR2,5);
0069     SET_BIT(TCCR2,4);
0070 #else
0071     #error "PWM Mode is not valid..."
0072 #endif
0073 #else
0074     #error "TIMER2 Mode is not valid..."
0075 #endif
0076
0077 }
0078 /*Set Preload Function*/
0079 void TIMER2_VidSetPreload(u8 Copy_u8Preload)
0080 {
0081     TCNT2=Copy_u8Preload;
0082 }
0083 /*Set CTC Value Function*/
0084 void TIMER2_VidSetCTCValue(u8 Copy_u8CTCValue)
0085 {

```

```

00086     OCR2=Copy_u8CTCValue;
00087 }
00088 void TIMER2_VidOVF_SetCallBack(void(*Copy_VidCallBack) (void))
00089 {
00090     TIMER2_CallBack=Copy_VidCallBack;
00091 }
00092 void TIMER2_VidCTC_SetCallBack(void(*Copy_VidCallBack) (void))
00093 {
00094     TIMER2_CallBack=Copy_VidCallBack;
00095 }
00096
00097
00098 void __vector_4(void)    __attribute__((signal));
00099 void __vector_4(void)
00100 {
00101     TIMER2_CallBack();
00102 }
00103 void __vector_5(void)    __attribute__((signal));
00104 void __vector_5(void)
00105 {
00106     TIMER2_CallBack();
00107 }
00108
00109
00110 void M_Pwm2Init(void)
00111 {
00112     CLR_BIT(TCCR2,0);
00113     SET_BIT(TCCR2,1);
00114
00115 }
00116
00117 void M_Pwm2SetDutyCycle(f64 f64_local_dutyCycle)
00118 {
00119     OCR2 = ((f64_local_dutyCycle * 1250 ) / 100 ) - 1;
00120 }
00121 void M_Pwm2Start(void)
00122 {
00123     // to select 1024 division factor
00124     /*SET_BIT(TCCR2,0);
00125     CLR_BIT(TCCR2,1);
00126     SET_BIT(TCCR2,2);*/
00127     CLR_BIT(TCCR2,0);
00128     CLR_BIT(TCCR2,1);
00129     SET_BIT(TCCR2,2);
00130
00131 }
00132 void M_Pwm2Stop(void)
00133 {
00134     CLR_BIT(TCCR2,0);
00135     CLR_BIT(TCCR2,1);
00136     CLR_BIT(TCCR2,2);
00137 }

```

22.129 code/Release/FreeRTOS/croutine.d File Reference

22.130 croutine.d

```

00001 FreeRTOS/croutine.o FreeRTOS/croutine.o: ./FreeRTOS/croutine.c \
00002     ./FreeRTOS/FreeRTOS.h ./FreeRTOS/projdefs.h \
00003     ./FreeRTOS/FreeRTOSConfig.h ./FreeRTOS/portable.h \
00004     ./FreeRTOS/portmacro.h ./FreeRTOS/mpu_wrappers.h \
00005     ./FreeRTOS/task.h ./FreeRTOS/list.h ./FreeRTOS/croutine.h
00006
00007 ./FreeRTOS/FreeRTOS.h:
00008
00009 ./FreeRTOS/projdefs.h:
00010
00011 ./FreeRTOS/FreeRTOSConfig.h:
00012
00013 ./FreeRTOS/portable.h:
00014
00015 ./FreeRTOS/portmacro.h:
00016
00017 ./FreeRTOS/mpu_wrappers.h:
00018
00019 ./FreeRTOS/task.h:
00020
00021 ./FreeRTOS/list.h:
00022
00023 ./FreeRTOS/croutine.h:

```

22.131 code/Release/FreeRTOS/heap_1.d File Reference

22.132 heap_1.d

```
00001 FreeRTOS/heap_1.o FreeRTOS/heap_1.o: ../../FreeRTOS/heap_1.c \
00002     ../../FreeRTOS/FreeRTOS.h ../../FreeRTOS/projdefs.h \
00003     ../../FreeRTOS/FreeRTOSConfig.h ../../FreeRTOS/portable.h \
00004     ../../FreeRTOS/portmacro.h ../../FreeRTOS/mpu_wrappers.h \
00005     ../../FreeRTOS/task.h ../../FreeRTOS/list.h
00006
00007 ../../FreeRTOS/FreeRTOS.h:
00008
00009 ../../FreeRTOS/projdefs.h:
00010
00011 ../../FreeRTOS/FreeRTOSConfig.h:
00012
00013 ../../FreeRTOS/portable.h:
00014
00015 ../../FreeRTOS/portmacro.h:
00016
00017 ../../FreeRTOS/mpu_wrappers.h:
00018
00019 ../../FreeRTOS/task.h:
00020
00021 ../../FreeRTOS/list.h:
```

22.133 code/Release/FreeRTOS/list.d File Reference

22.134 list.d

```
00001 FreeRTOS/list.o FreeRTOS/list.o: ../../FreeRTOS/list.c \
00002     ../../FreeRTOS/FreeRTOS.h ../../FreeRTOS/projdefs.h \
00003     ../../FreeRTOS/FreeRTOSConfig.h ../../FreeRTOS/portable.h \
00004     ../../FreeRTOS/portmacro.h ../../FreeRTOS/mpu_wrappers.h \
00005     ../../FreeRTOS/list.h
00006
00007 ../../FreeRTOS/FreeRTOS.h:
00008
00009 ../../FreeRTOS/projdefs.h:
00010
00011 ../../FreeRTOS/FreeRTOSConfig.h:
00012
00013 ../../FreeRTOS/portable.h:
00014
00015 ../../FreeRTOS/portmacro.h:
00016
00017 ../../FreeRTOS/mpu_wrappers.h:
00018
00019 ../../FreeRTOS/list.h:
```

22.135 code/Release/FreeRTOS/port.d File Reference

22.136 port.d

```
00001 FreeRTOS/port.o FreeRTOS/port.o: ../../FreeRTOS/port.c \
00002     ../../FreeRTOS/FreeRTOS.h ../../FreeRTOS/projdefs.h \
00003     ../../FreeRTOS/FreeRTOSConfig.h ../../FreeRTOS/portable.h \
00004     ../../FreeRTOS/portmacro.h ../../FreeRTOS/mpu_wrappers.h \
00005     ../../FreeRTOS/task.h ../../FreeRTOS/list.h
00006
00007 ../../FreeRTOS/FreeRTOS.h:
00008
00009 ../../FreeRTOS/projdefs.h:
00010
00011 ../../FreeRTOS/FreeRTOSConfig.h:
00012
00013 ../../FreeRTOS/portable.h:
00014
00015 ../../FreeRTOS/portmacro.h:
00016
00017 ../../FreeRTOS/mpu_wrappers.h:
00018
00019 ../../FreeRTOS/task.h:
00020
00021 ../../FreeRTOS/list.h:
```

22.137 code/Release/FreeRTOS/queue.d File Reference**22.138 queue.d**

```
00001 FreeRTOS/queue.o FreeRTOS/queue.o: ../../FreeRTOS/queue.c \
00002     ../../FreeRTOS/FreeRTOS.h ../../FreeRTOS/projdefs.h \
00003     ../../FreeRTOS/FreeRTOSConfig.h ../../FreeRTOS/portable.h \
00004     ../../FreeRTOS/portmacro.h ../../FreeRTOS/mpu_wrappers.h \
00005     ../../FreeRTOS/task.h ../../FreeRTOS/list.h
00006
00007 ../../FreeRTOS/FreeRTOS.h:
00008
00009 ../../FreeRTOS/projdefs.h:
00010
00011 ../../FreeRTOS/FreeRTOSConfig.h:
00012
00013 ../../FreeRTOS/portable.h:
00014
00015 ../../FreeRTOS/portmacro.h:
00016
00017 ../../FreeRTOS/mpu_wrappers.h:
00018
00019 ../../FreeRTOS/task.h:
00020
00021 ../../FreeRTOS/list.h:
```

22.139 code/Release/FreeRTOS/tasks.d File Reference**22.140 tasks.d**

```
00001 FreeRTOS/tasks.o FreeRTOS/tasks.o: ../../FreeRTOS/tasks.c \
00002     ../../FreeRTOS/FreeRTOS.h ../../FreeRTOS/projdefs.h \
00003     ../../FreeRTOS/FreeRTOSConfig.h ../../FreeRTOS/portable.h \
00004     ../../FreeRTOS/portmacro.h ../../FreeRTOS/mpu_wrappers.h \
00005     ../../FreeRTOS/task.h ../../FreeRTOS/list.h ../../FreeRTOS/timers.h \
00006     ../../FreeRTOS/StackMacros.h
00007
00008 ../../FreeRTOS/FreeRTOS.h:
00009
00010 ../../FreeRTOS/projdefs.h:
00011
00012 ../../FreeRTOS/FreeRTOSConfig.h:
00013
00014 ../../FreeRTOS/portable.h:
00015
00016 ../../FreeRTOS/portmacro.h:
00017
00018 ../../FreeRTOS/mpu_wrappers.h:
00019
00020 ../../FreeRTOS/task.h:
00021
00022 ../../FreeRTOS/list.h:
00023
00024 ../../FreeRTOS/timers.h:
00025
00026 ../../FreeRTOS/StackMacros.h:
```

22.141 code/Release/FreeRTOS/timers.d File Reference**22.142 timers.d**

```
00001 FreeRTOS/timers.o FreeRTOS/timers.o: ../../FreeRTOS/timers.c \
00002     ../../FreeRTOS/FreeRTOS.h ../../FreeRTOS/projdefs.h \
00003     ../../FreeRTOS/FreeRTOSConfig.h ../../FreeRTOS/portable.h \
00004     ../../FreeRTOS/portmacro.h ../../FreeRTOS/mpu_wrappers.h \
00005     ../../FreeRTOS/task.h ../../FreeRTOS/list.h ../../FreeRTOS/queue.h \
00006     ../../FreeRTOS/timers.h
00007
00008 ../../FreeRTOS/FreeRTOS.h:
00009
00010 ../../FreeRTOS/projdefs.h:
00011
00012 ../../FreeRTOS/FreeRTOSConfig.h:
00013
00014 ../../FreeRTOS/portable.h:
```

```

00015
00016 .../FreeRTOS/portmacro.h:
00017
00018 .../FreeRTOS/mpu_wrappers.h:
00019
00020 .../FreeRTOS/task.h:
00021
00022 .../FreeRTOS/list.h:
00023
00024 .../FreeRTOS/queue.h:
00025
00026 .../FreeRTOS/timers.h:

```

22.143 code/Release/HAL/DC_MOTOR/DC_MOTOR.d File Reference

22.144 DC_MOTOR.d

```

00001 HAL/DC_MOTOR/DC_MOTOR.o HAL/DC_MOTOR/DC_MOTOR.o: \
00002 .../HAL/DC_MOTOR/DC_MOTOR.c .../HAL/DC_MOTOR/DC_MOTOR.h \
00003 .../HAL/DC_MOTOR/...\\LIB\\LSTD_TYPES.h .../HAL/DC_MOTOR/DC_MOTOR_CFG.h \
00004 .../HAL/DC_MOTOR/...\\MCAL\\MDIO\\MDIO_Interface.h \
00005 .../HAL/DC_MOTOR/...\\MCAL\\MDIO\\MDIO_Interface.h
00006
00007 .../HAL/DC_MOTOR/DC_MOTOR.h:
00008
00009 .../HAL/DC_MOTOR/...\\LIB\\LSTD_TYPES.h:
00010
00011 .../HAL/DC_MOTOR/DC_MOTOR_CFG.h:
00012
00013 .../HAL/DC_MOTOR/...\\MCAL\\MDIO\\MDIO_Interface.h:
00014
00015 .../HAL/DC_MOTOR/...\\MCAL\\MDIO\\MDIO_Interface.h:

```

22.145 code/Release/HAL/DC_MOTOR_POT/DC_MOTOR_POT.d File Reference

22.146 DC_MOTOR_POT.d

```

00001 HAL/DC_MOTOR_POT/DC_MOTOR_POT.o HAL/DC_MOTOR_POT/DC_MOTOR_POT.o: \
00002 .../HAL/DC_MOTOR_POT/DC_MOTOR_POT.c .../HAL/DC_MOTOR_POT/DC_MOTOR_POT.h \
00003 .../HAL/DC_MOTOR_POT/...\\LIB\\LSTD_TYPES.h \
00004 .../HAL/DC_MOTOR_POT/DC_MOTOR_POT_CFG.h \
00005 .../HAL/DC_MOTOR_POT/...\\MCAL\\MDIO\\MDIO_Interface.h \
00006 .../HAL/DC_MOTOR_POT/...\\MCAL\\MDIO\\MDIO_Interface.h
00007
00008 .../HAL/DC_MOTOR_POT/DC_MOTOR_POT.h:
00009
00010 .../HAL/DC_MOTOR_POT/...\\LIB\\LSTD_TYPES.h:
00011
00012 .../HAL/DC_MOTOR_POT/DC_MOTOR_POT_CFG.h:
00013
00014 .../HAL/DC_MOTOR_POT/...\\MCAL\\MDIO\\MDIO_Interface.h:
00015
00016 .../HAL/DC_MOTOR_POT/...\\MCAL\\MDIO\\MDIO_Interface.h:

```

22.147 code/Release/HAL/HCLCD/HCLCD_Program.d File Reference

22.148 HCLCD_Program.d

```

00001 HAL/HCLCD/HCLCD_Program.o HAL/HCLCD/HCLCD_Program.o: \
00002 .../HAL/HCLCD/HCLCD_Program.c .../HAL/HCLCD/...\\LIB\\LSTD_TYPES.h \
00003 .../HAL/HCLCD/...\\MCAL\\MDIO\\MDIO_Interface.h \
00004 .../HAL/HCLCD/HCLCD_Private.h .../HAL/HCLCD/HCLCD_Config.h \
00005 .../HAL/HCLCD/HCLCD_Interface.h
00006
00007 .../HAL/HCLCD/...\\LIB\\LSTD_TYPES.h:
00008
00009 .../HAL/HCLCD/...\\MCAL\\MDIO\\MDIO_Interface.h:
00010
00011 .../HAL/HCLCD/HCLCD_Private.h:
00012
00013 .../HAL/HCLCD/HCLCD_Config.h:
00014
00015 .../HAL/HCLCD/HCLCD_Interface.h:

```

22.149 code/Release/HAL/KEY_PAD/HKPD_Program.d File Reference

22.150 HKPD_Program.d

```

00001 HAL/KEY_PAD/HKPD_Program.o HAL/KEY_PAD/HKPD_Program.o: \
00002   ..\HAL/KEY_PAD/HKPD_Program.c ..\HAL/KEY_PAD\..\..\LIB\LSTD_TYPES.h \
00003   ..\HAL/KEY_PAD\..\..\MCAL\MDIO\MDIO_Interface.h \
00004   ..\HAL/KEY_PAD/HKPD_Config.h ..\HAL/KEY_PAD/HKPD_Interface.h
00005
00006   ..\HAL/KEY_PAD\..\..\LIB\LSTD_TYPES.h:
00007
00008   ..\HAL/KEY_PAD\..\..\MCAL\MDIO\MDIO_Interface.h:
00009
00010   ..\HAL/KEY_PAD/HKPD_Config.h:
00011
00012   ..\HAL/KEY_PAD/HKPD_Interface.h:

```

22.151 code/Release/HAL/SERVO_M/SERVO_Program.d File Reference

22.152 SERVO_Program.d

```

00001 HAL/SERVO_M/SERVO_Program.o HAL/SERVO_M/SERVO_Program.o: \
00002   ..\HAL/SERVO_M/SERVO_Program.c ..\HAL/SERVO_M\..\..\LIB\LSTD_TYPES.h \
00003   ..\HAL/SERVO_M\..\..\LIB\LBIT_MATH.h \
00004   ..\HAL/SERVO_M\..\..\MCAL/TIMER2/TIMER2_Interface.h \
00005   ..\HAL/SERVO_M\..\..\MCAL/MDIO/MDIO_Interface.h
00006
00007   ..\HAL/SERVO_M\..\..\LIB\LSTD_TYPES.h:
00008
00009   ..\HAL/SERVO_M\..\..\LIB\LBIT_MATH.h:
00010
00011   ..\HAL/SERVO_M\..\..\MCAL/TIMER2/TIMER2_Interface.h:
00012
00013   ..\HAL/SERVO_M\..\..\MCAL/MDIO/MDIO_Interface.h:

```

22.153 code/Release/main.d File Reference

22.154 main.d

```

00001 main.o main.o: ..\main.c ..\main.h ..\LIB\LSTD_TYPES.h ..\LIB\LBIT_MATH.h \
00002   ..\MCAL\MDIO/MDIO_Interface.h ..\MCAL/MGIE/MGIE_Interface.h \
00003   ..\MCAL/MADC/MADC_Interface.h ..\MCAL/TIMER2/TIMER2_Interface.h \
00004   ..\HAL/SERVO_M/SERVO_Interface.h ..\HAL/SERVO_M\..\..\LIB\LSTD_TYPES.h \
00005   ..\HAL/HCLCD/HCLCD_Interface.h ..\HAL/KEY_PAD/HKPD_Interface.h \
00006   ..\HAL/DC_MOTOR/DC_MOTOR.h ..\HAL/DC_MOTOR\..\..\LIB\LSTD_TYPES.h \
00007   ..\HAL/DC_MOTOR/DC_MOTOR_CFG.h \
00008   ..\HAL/DC_MOTOR\..\..\MCAL\MDIO\MDIO_Interface.h \
00009   ..\HAL/DC_MOTOR_POT/DC_MOTOR_POT.h \
00010   ..\HAL/DC_MOTOR_POT\..\..\LIB\LSTD_TYPES.h \
00011   ..\HAL/DC_MOTOR_POT/DC_MOTOR_POT_CFG.h \
00012   ..\HAL/DC_MOTOR_POT\..\..\MCAL\MDIO\MDIO_Interface.h \
00013   ..\FreeRTOS/FreeRTOS.h ..\FreeRTOS/projdefs.h \
00014   ..\FreeRTOS/FreeRTOSConfig.h ..\FreeRTOS/portable.h \
00015   ..\FreeRTOS/portmacro.h ..\FreeRTOS/mpu_wrappers.h \
00016   ..\FreeRTOS/task.h ..\FreeRTOS/list.h ..\FreeRTOS/semp.h \
00017   ..\FreeRTOS/queue.h ..\main_cfg.h
00018
00019   ..\main.h:
00020
00021   ..\LIB\LSTD_TYPES.h:
00022
00023   ..\LIB\LBIT_MATH.h:
00024
00025   ..\MCAL\MDIO/MDIO_Interface.h:
00026
00027   ..\MCAL/MGIE/MGIE_Interface.h:
00028
00029   ..\MCAL/MADC/MADC_Interface.h:
00030
00031   ..\MCAL/TIMER2/TIMER2_Interface.h:
00032
00033   ..\HAL/SERVO_M/SERVO_Interface.h:
00034
00035   ..\HAL/SERVO_M\..\..\LIB\LSTD_TYPES.h:
00036

```

```

00037 ../HAL/HCLCD/HCLCD_Interface.h:
00038
00039 ../HAL/KEY_PAD/HKPD_Interface.h:
00040
00041 ../HAL/DC_MOTOR/DC_MOTOR.h:
00042
00043 ../HAL/DC_MOTOR/../../LIB\LSTD_TYPES.h:
00044
00045 ../HAL/DC_MOTOR/DC_MOTOR_CFG.h:
00046
00047 ../HAL/DC_MOTOR/../../MCAL\MDIO\MDIO_Interface.h:
00048
00049 ../HAL/DC_MOTOR_POT/DC_MOTOR_POT.h:
00050
00051 ../HAL/DC_MOTOR_POT/../../LIB\LSTD_TYPES.h:
00052
00053 ../HAL/DC_MOTOR_POT/DC_MOTOR_POT_CFG.h:
00054
00055 ../HAL/DC_MOTOR_POT/../../MCAL\MDIO\MDIO_Interface.h:
00056
00057 ./FreeRTOS/Freertos.h:
00058
00059 ./FreeRTOS/projdefs.h:
00060
00061 ./FreeRTOS/FreertosConfig.h:
00062
00063 ./FreeRTOS/portable.h:
00064
00065 ./FreeRTOS/portmacro.h:
00066
00067 ./FreeRTOS/mpu_wrappers.h:
00068
00069 ./FreeRTOS/task.h:
00070
00071 ./FreeRTOS/list.h:
00072
00073 ./FreeRTOS/semphr.h:
00074
00075 ./FreeRTOS/queue.h:
00076
00077 ./main_cfg.h:
```

22.155 code/Release/main_cfg.d File Reference

22.156 main_cfg.d

```

00001 main_cfg.o main_cfg.o: ./main_cfg.c ../../LIB\LSTD_TYPES.h \
00002 ../../LIB\LBIT_MATH.h ../../MCAL\MDIO\MDIO_Interface.h \
00003 ../../MCAL\MGIE\MGIE_Interface.h ../../MCAL\MADC\MADC_Interface.h \
00004 ../../MCAL\TIMER2\TIMER2_INTERFACE.h ../../HAL\SERVO_M\SERVO_Interface.h \
00005 ../../HAL\SERVO_M/../../LIB\LSTD_TYPES.h ../../HAL\HCLCD\HCLCD_Interface.h \
00006 ../../HAL\KEY_PAD\HKPD_Interface.h ../../HAL\DC_MOTOR\DC_MOTOR.h \
00007 ../../HAL\DC_MOTOR/../../LIB\LSTD_TYPES.h ../../HAL\DC_MOTOR\DC_MOTOR_CFG.h \
00008 ../../HAL\DC_MOTOR/../../MCAL\MDIO\MDIO_Interface.h \
00009 ../../HAL\DC_MOTOR_POT\DC_MOTOR_POT.h \
00010 ../../HAL\DC_MOTOR_POT/../../LIB\LSTD_TYPES.h \
00011 ../../HAL\DC_MOTOR_POT\DC_MOTOR_POT_CFG.h \
00012 ../../HAL\DC_MOTOR_POT/../../MCAL\MDIO\MDIO_Interface.h \
00013 ../../FreeRTOS/Freertos.h ../../FreeRTOS/projdefs.h \
00014 ../../FreeRTOS/FreertosConfig.h ../../FreeRTOS/portable.h \
00015 ../../FreeRTOS/portmacro.h ../../FreeRTOS/mpu_wrappers.h \
00016 ../../FreeRTOS/task.h ../../FreeRTOS/list.h ../../FreeRTOS/semphr.h \
00017 ../../FreeRTOS/queue.h ./main_cfg.h \
00018
00019 ../../LIB\LSTD_TYPES.h:
00020
00021 ../../LIB\LBIT_MATH.h:
00022
00023 ../../main.h:
00024
00025 ../../MCAL\MDIO\MDIO_Interface.h:
00026
00027 ../../MCAL\MGIE\MGIE_Interface.h:
00028
00029 ../../MCAL\MADC\MADC_Interface.h:
00030
00031 ../../MCAL\TIMER2\TIMER2_INTERFACE.h:
00032
00033 ../../HAL\SERVO_M\SERVO_Interface.h:
00034
00035 ../../HAL\SERVO_M/../../LIB\LSTD_TYPES.h:
00036
```

```
00037 ../../HAL/HCLCD/HCLCD_Interface.h:  
00038  
00039 ../../HAL/KEY_PAD/HKPD_Interface.h:  
00040  
00041 ../../HAL/DC_MOTOR/DC_MOTOR.h:  
00042  
00043 ../../HAL/DC_MOTOR/../../LIB/LSTD_TYPES.h:  
00044  
00045 ../../HAL/DC_MOTOR/DC_MOTOR_CFG.h:  
00046  
00047 ../../HAL/DC_MOTOR/../../MCAL/MDIO/MDIO_Interface.h:  
00048  
00049 ../../HAL/DC_MOTOR_POT/DC_MOTOR_POT.h:  
00050  
00051 ../../HAL/DC_MOTOR_POT/../../LIB/LSTD_TYPES.h:  
00052  
00053 ../../HAL/DC_MOTOR_POT/DC_MOTOR_POT_CFG.h:  
00054  
00055 ../../HAL/DC_MOTOR_POT/../../MCAL/MDIO/MDIO_Interface.h:  
00056  
00057 ./FreeRTOS/Freertos.h:  
00058  
00059 ./FreeRTOS/projdefs.h:  
00060  
00061 ./FreeRTOS/FreeRTOSConfig.h:  
00062  
00063 ./FreeRTOS/portable.h:  
00064  
00065 ./FreeRTOS/portmacro.h:  
00066  
00067 ./FreeRTOS/mpu_wrappers.h:  
00068  
00069 ./FreeRTOS/task.h:  
00070  
00071 ./FreeRTOS/list.h:  
00072  
00073 ./FreeRTOS/semphr.h:  
00074  
00075 ./FreeRTOS/queue.h:  
00076  
00077 ./main_cfg.h:
```

22.157 code/Release/MCAL/MADC/MADC_Program.d File Reference

22.158 MADC_Program.d

```
00001 MCAL/MADC/MADC_Program.o MCAL/MADC/MADC_Program.o: \  
00002 ../../MCAL/MADC/MADC_Program.c ../../MCAL/MADC/../../LIB/LSTD_TYPES.h \  
00003 ../../MCAL/MADC/../../LIB/LBIT_MATH.h ../../MCAL/MADC/MADC_Private.h \  
00004 ../../MCAL/MADC/MADC_Config.h  
00005  
00006 ../../MCAL/MADC/../../LIB/LSTD_TYPES.h:  
00007  
00008 ../../MCAL/MADC/../../LIB/LBIT_MATH.h:  
00009  
00010 ../../MCAL/MADC/MADC_Private.h:  
00011  
00012 ../../MCAL/MADC/MADC_Config.h:
```

22.159 code/Release/MCAL/MDIO/MDIO_Program.d File Reference

22.160 MDIO_Program.d

```
00001 MCAL/MDIO/MDIO_Program.o MCAL/MDIO/MDIO_Program.o: \  
00002 ../../MCAL/MDIO/MDIO_Program.c ../../MCAL/MDIO/../../LIB/LSTD_TYPES.h \  
00003 ../../MCAL/MDIO/../../LIB/LBIT_MATH.h ../../MCAL/MDIO/MDIO_Private.h \  
00004 ../../MCAL/MDIO/MDIO_Interface.h  
00005  
00006 ../../MCAL/MDIO/../../LIB/LSTD_TYPES.h:  
00007  
00008 ../../MCAL/MDIO/../../LIB/LBIT_MATH.h:  
00009  
00010 ../../MCAL/MDIO/MDIO_Private.h:  
00011  
00012 ../../MCAL/MDIO/MDIO_Interface.h:
```

22.161 code/Release/MCAL/MGIE/MGIE_Program.d File Reference**22.162 MGIE_Program.d**

```
00001 MCAL/MGIE/MGIE_Program.o MCAL/MGIE/MGIE_Program.o: \
00002   ..\MCAL\MGIE\MGIE_Program.c ..\MCAL\MGIE\..\..\LIB\LSTD_TYPES.h \
00003   ..\MCAL\MGIE\..\..\LIB\LBIT_MATH.h ..\MCAL\MGIE\Private.h
00004
00005 ..\MCAL\MGIE\..\..\LIB\LSTD_TYPES.h:
00006
00007 ..\MCAL\MGIE\..\..\LIB\LBIT_MATH.h:
00008
00009 ..\MCAL\MGIE\Private.h:
```

22.163 code/Release/MCAL/TIMER/TIMER_Program.d File Reference**22.164 TIMER_Program.d**

```
00001 MCAL/TIMER/TIMER_Program.o MCAL/TIMER/TIMER_Program.o: \
00002   ..\MCAL/TIMER/TIMER_Program.c ..\MCAL/TIMER\..\..\LIB\LSTD_TYPES.h \
00003   ..\MCAL/TIMER\..\..\LIB\LBIT_MATH.h ..\MCAL/TIMER/TIMER_Private.h \
00004   ..\MCAL/TIMER/TIMER_Config.h
00005
00006 ..\MCAL/TIMER\..\..\LIB\LSTD_TYPES.h:
00007
00008 ..\MCAL/TIMER\..\..\LIB\LBIT_MATH.h:
00009
00010 ..\MCAL/TIMER/TIMER_Private.h:
00011
00012 ..\MCAL/TIMER/TIMER_Config.h:
```

22.165 code/Release/MCAL/TIMER2/TIMER2_Program.d File Reference**22.166 TIMER2_Program.d**

```
00001 MCAL/TIMER2/TIMER2_Program.o MCAL/TIMER2/TIMER2_Program.o: \
00002   ..\MCAL/TIMER2/TIMER2_Program.c ..\MCAL/TIMER2\..\..\LIB\LSTD_TYPES.h \
00003   ..\MCAL/TIMER2\..\..\LIB\LBIT_MATH.h \
00004   ..\MCAL/TIMER2\..\..\MCAL\MDIO\MDIO_Interface.h \
00005   ..\MCAL/TIMER2/TIMER2_Private.h ..\MCAL/TIMER2/TIMER2_Config.h
00006
00007 ..\MCAL/TIMER2\..\..\LIB\LSTD_TYPES.h:
00008
00009 ..\MCAL/TIMER2\..\..\LIB\LBIT_MATH.h:
00010
00011 ..\MCAL/TIMER2\..\..\MCAL\MDIO\MDIO_Interface.h:
00012
00013 ..\MCAL/TIMER2/TIMER2_Private.h:
00014
00015 ..\MCAL/TIMER2/TIMER2_Config.h:
```


Index

__vector_10
 TIMER_Program.c, 411

__vector_11
 TIMER_Program.c, 411

__vector_16
 MADC_Private.h, 380
 MADC_Program.c, 382

__vector_4
 TIMER2_Program.c, 424

__vector_5
 TIMER2_Program.c, 424

A_CLK_W
 DC_MOTOR.h, 286
 DC_MOTOR_POT.h, 294

ADC
 MADC_Private.h, 378

ADC_Semaphore
 main.c, 351

ADC_SetNotification
 main.c, 341
 main.h, 357

ADCSRA
 MADC_Private.h, 378

ADMUX
 MADC_Private.h, 378

attempts
 PASSWORD_t, 61

buzzer
 FLAG_t, 60

CHANNEL_0
 MADC_Interface.h, 374

CHANNEL_1
 MADC_Interface.h, 374

CHANNEL_2
 MADC_Interface.h, 374

CHANNEL_3
 MADC_Interface.h, 374

CHANNEL_4
 MADC_Interface.h, 374

CHANNEL_5
 MADC_Interface.h, 374

CHANNEL_6
 MADC_Interface.h, 375

CHANNEL_7
 MADC_Interface.h, 375

CLK_W
 DC_MOTOR.h, 286
 DC_MOTOR_POT.h, 294

CLR_BIT
 LBIT_MATH.h, 336
 macros.h, 128

code/FreeRTOS/croutine.c, 76, 78

code/FreeRTOS/croutine.h, 83, 88

code/FreeRTOS/FreeRTOS.h, 90, 101

code/FreeRTOS/FreeRTOSConfig.h, 107, 111

code/FreeRTOS/heap_1.c, 112, 114

code/FreeRTOS/list.c, 116, 118

code/FreeRTOS/list.h, 120, 124

code/FreeRTOS/macros.h, 128, 129

code/FreeRTOS/mpu_wrappers.h, 130, 131

code/FreeRTOS/port.c, 132, 135

code/FreeRTOS/portable.h, 140, 144

code/FreeRTOS/portmacro.h, 148, 152

code/FreeRTOS/projdefs.h, 154, 156

code/FreeRTOS/queue.c, 156, 163

code/FreeRTOS/queue.h, 181, 190

code/FreeRTOS/semphr.h, 192, 197

code/FreeRTOS/StackMacros.h, 199, 200

code/FreeRTOS/task.h, 202, 213

code/FreeRTOS/tasks.c, 217, 225

code/FreeRTOS/timers.c, 255, 256

code/FreeRTOS/timers.h, 264, 280

code/HAL/DC_MOTOR/DC_MOTOR.c, 281, 284

code/HAL/DC_MOTOR/DC_MOTOR.h, 284, 288

code/HAL/DC_MOTOR/DC_MOTOR_CFG.h, 288, 290

code/HAL/DC_MOTOR_POT/DC_MOTOR_POT.c, 290, 292

code/HAL/DC_MOTOR_POT/DC_MOTOR_POT.h, 293, 296

code/HAL/DC_MOTOR_POT/DC_MOTOR_POT_CFG.h, 296, 297

code/HAL/HLCD/HLCD_Config.h, 298, 300

code/HAL/HLCD/HLCD_Interface.h, 300, 308

code/HAL/HLCD/HLCD_Private.h, 308, 311

code/HAL/HLCD/HLCD_Program.c, 312, 319

code/HAL/KEY_PAD/HKPD_Config.h, 321, 324

code/HAL/KEY_PAD/HKPD_Interface.h, 324, 326

code/HAL/KEY_PAD/HKPD_Private.h, 326

code/HAL/KEY_PAD/HKPD_Program.c, 326, 329

code/HAL/SERVO_M/SERVO_Config.h, 330

code/HAL/SERVO_M/SERVO_Interface.h, 330, 333

code/HAL/SERVO_M/SERVO_Private.h, 333

code/HAL/SERVO_M/SERVO_Program.c, 333, 335

code/LIB/LBIT_MATH.h, 335, 336

code/LIB/LSTD_TYPES.h, 336, 339

code/main.c, 339, 352

code/main.h, 356, 367

code/main_cfg.c, 368, 369

code/main_cfg.h, 370, 371

code/MCAL/MADC/MADC_Config.h, 372, 373

code/MCAL/MADC/MADC_Interface.h, 373, 377

code/MCAL/MADC/MADC_Private.h, 377, 381

code/MCAL/MADC/MADC_Program.c, 381, 384

code/MCAL/MDIO/MDIO_Config.h, 385

code/MCAL/MDIO/MDIO_Interface.h, 385, 389

code/MCAL/MDIO/MDIO_Private.h, 390, 392

code/MCAL/MDIO/MDIO_Program.c, 393, 396

code/MCAL/MGIE/MGIE_Interface.h, 399, 400

code/MCAL/MGIE/MGIE_Private.h, 401
 code/MCAL/MGIE/MGIE_Program.c, 402, 403
 code/MCAL/TIMER/TIMER_Config.h, 403, 404
 code/MCAL/TIMER/TIMER_Interface.h, 405, 406
 code/MCAL/TIMER/TIMER_Private.h, 406, 410
 code/MCAL/TIMER/TIMER_Program.c, 410, 412
 code/MCAL/TIMER2/TIMER2_Config.h, 414, 415
 code/MCAL/TIMER2/TIMER2_Interface.h, 416, 419
 code/MCAL/TIMER2/TIMER2_Private.h, 419, 423
 code/MCAL/TIMER2/TIMER2_Program.c, 423, 427
 code/Release/FreeRTOS/croutine.d, 428
 code/Release/FreeRTOS/heap_1.d, 429
 code/Release/FreeRTOS/list.d, 429
 code/Release/FreeRTOS/port.d, 429
 code/Release/FreeRTOS/queue.d, 430
 code/Release/FreeRTOS/tasks.d, 430
 code/Release/FreeRTOS/timers.d, 430
 code/Release/HAL/DC_MOTOR/DC_MOTOR.d, 431
 code/Release/HAL/DC_MOTOR_POT/DC_MOTOR_POT.d, 431
 code/Release/HAL/HCLCD/HCLCD_Program.d, 431
 code/Release/HAL/KEY_PAD/HKPD_Program.d, 432
 code/Release/HAL/SERVO_M/SERVO_Program.d, 432
 code/Release/main.d, 432
 code/Release/main_cfg.d, 433
 code/Release/MCAL/MADC/MADC_Program.d, 434
 code/Release/MCAL/MDIO/MDIO_Program.d, 434
 code/Release/MCAL/MGIE/MGIE_Program.d, 435
 code/Release/MCAL/TIMER/TIMER_Program.d, 435
 code/Release/MCAL/TIMER2/TIMER2_Program.d, 435
 COL_END
 HKPD_Config.h, 322
 COL_INIT
 HKPD_Config.h, 322
 COL_PIN0
 HKPD_Config.h, 322
 COL_PIN1
 HKPD_Config.h, 322
 COL_PIN2
 HKPD_Config.h, 322
 COL_PIN3
 HKPD_Config.h, 322
 COL_PORT
 HKPD_Config.h, 323
 comparePasswords
 main.c, 341
 main.h, 357
 configASSERT
 FreeRTOS.h, 92
 configCHECK_FOR_STACK_OVERFLOW
 FreeRTOS.h, 92
 configCPU_CLOCK_HZ
 FreeRTOSConfig.h, 108
 configGENERATE_RUN_TIME_STATS
 FreeRTOS.h, 92
 configIDLE_SHOULD_YIELD
 FreeRTOSConfig.h, 108
 configMAX_CO_ROUTINE_PRIORITIES
 FreeRTOSConfig.h, 108
 configMAX_PRIORITIES
 FreeRTOSConfig.h, 108
 configMAX_TASK_NAME_LEN
 FreeRTOSConfig.h, 108
 configMINIMAL_STACK_SIZE
 FreeRTOSConfig.h, 108
 configQUEUE_REGISTRY_SIZE
 FreeRTOSConfig.h, 108
 configTICK_RATE_HZ
 FreeRTOSConfig.h, 109
 configTOTAL_HEAP_SIZE
 FreeRTOSConfig.h, 109
 configUSE_16_BIT_TICKS
 FreeRTOSConfig.h, 109
 configUSE_ALTERNATIVE_API
 FreeRTOS.h, 92
 configUSE_APPLICATION_TASK_TAG
 FreeRTOS.h, 92
 configUSE_CO_Routines
 FreeRTOSConfig.h, 109
 configUSE_COUNTING_SEMAPHORES
 FreeRTOSConfig.h, 109
 configUSE_IDLE_HOOK
 FreeRTOSConfig.h, 109
 configUSE_MALLOC_FAILED_HOOK
 FreeRTOS.h, 92
 configUSE_MUTEXES
 FreeRTOS.h, 92
 configUSE_PREEMPTION
 FreeRTOSConfig.h, 109
 configUSE_RECURSIVE_MUTEXES
 FreeRTOS.h, 92
 configUSE_TICK_HOOK
 FreeRTOSConfig.h, 109
 configUSE_TIMERS
 FreeRTOS.h, 93
 configUSE_TRACEFacility
 FreeRTOSConfig.h, 110
 CONTROL_PORT
 HCLCD_Config.h, 298
 controlDoor
 main.c, 341
 main.h, 358
 corCoRoutineControlBlock, 58
 pxCoRoutineFunction, 58
 uxIndex, 58
 uxPriority, 59
 uxState, 59
 xEventListItem, 59
 xGenericListItem, 59
 corCRCB
 croutine.h, 86
 corINITIAL_STATE
 croutine.c, 76
 correctPassword
 FLAG_t, 60
 crCOROUTINE_CODE

croutine.h, 86
crDELAY, 13
 croutine.h, 84
crEND
 croutine.h, 84
croutine.c
 corINITIAL_STATE, 76
 prvAddCoRoutineToReadyQueue, 76
 pxCurrentCoRoutine, 78
 vCoRoutineAddToDelayedList, 77
 vCoRoutineSchedule, 77
 xCoRoutineCreate, 77
 xCoRoutineRemoveFromEventList, 77
croutine.h
 corCRCB, 86
 crCOROUTINE_CODE, 86
 crDELAY, 84
 crEND, 84
 crQUEUE_RECEIVE, 84
 crQUEUE_RECEIVE_FROM_ISR, 85
 crQUEUE_SEND, 85
 crQUEUE_SEND_FROM_ISR, 85
 crSET_STATE0, 86
 crSET_STATE1, 86
 crSTART, 86
 vCoRoutineAddToDelayedList, 87
 vCoRoutineSchedule, 87
 xCoRoutineCreate, 87
 xCoRoutineHandle, 86
 xCoRoutineRemoveFromEventList, 87
crQUEUE_RECEIVE, 15
 croutine.h, 84
crQUEUE_RECEIVE_FROM_ISR, 17
 croutine.h, 85
crQUEUE_SEND, 14
 croutine.h, 85
crQUEUE_SEND_FROM_ISR, 16
 croutine.h, 85
crSET_STATE0
 croutine.h, 86
crSET_STATE1
 croutine.h, 86
crSTART, 12
 croutine.h, 86

DATA_PORT
 HCLCD_Config.h, 298

DC_M_PORT
 DC_MOTOR_CFG.h, 289

DC_M_POT_PORT
 DC_MOTOR_POT_CFG.h, 297

DC_MOTOR.c
 H_DcMotorInit, 282
 H_DcMotorStart, 283
 H_DcMotorStop, 283

DC_MOTOR.h
 A_CLK_W, 286
 CLK_W, 286
 H_DcMotorInit, 286

 H_DcMotorSetDirection, 286
 H_DcMotorSetSpeed, 286
 H_DcMotorStart, 287
 H_DcMotorStop, 287

DC_MOTOR_CFG.h
 DC_M_PORT, 289
 IN_1, 289
 IN_2, 289

DC_MOTOR_POT.c
 H_DcMotorPotInit, 290
 H_DcMotorPotStart, 291
 H_DcMotorPotStop, 291

DC_MOTOR_POT.h
 A_CLK_W, 294
 CLK_W, 294
 H_DcMotorPotInit, 294
 H_DcMotorPotStart, 294
 H_DcMotorPotStop, 295
 H_DcMotorSetDirection, 295
 H_DcMotorSetSpeed, 296

DC_MOTOR_POT_CFG.h
 DC_M_POT_PORT, 297
 IN_1_POT, 297

DDRA
 MDIO_Private.h, 391

DDRB
 MDIO_Private.h, 391

DDRC
 MDIO_Private.h, 391

DDRD
 MDIO_Private.h, 391

DISPLAY_CLEAR
 HCLCD_Interface.h, 301

DISPLAY_OFF
 HCLCD_Private.h, 309

DISPLAY_ON_CURSOR_OFF
 HCLCD_Private.h, 309

DISPLAY_ON_CURSOR_ON_BLINKING_OFF
 HCLCD_Private.h, 309

DISPLAY_ON_CURSOR_ON_BLINKING_ON
 HCLCD_Private.h, 309

displayDoorState
 main.c, 342
 main.h, 358

displayEntryPasswordSecurity
 main.c, 343
 main.h, 359

displayFireState
 main.c, 343
 main.h, 360

displayTemperatureValue
 main.c, 344
 main.h, 361

E
 HCLCD_Config.h, 298

ENTRY_MODE_SET_DECREASE
 HCLCD_Private.h, 309

ENTRY_MODE_SET_DECREASE_WITH_SHIFT

HLCD_Private.h, 309
 ENTRY_MODE_SET_INCREASE
 LCD_Private.h, 310
 ENTRY_MODE_SET_INCREASE_WITH_SHIFT
 LCD_Private.h, 310
 errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY
 projdefs.h, 154
 errNO_TASK_TO_RUN
 projdefs.h, 154
 Error_State
 LSTD_TYPES.h, 338
 errQUEUE_BLOCKED
 projdefs.h, 154
 errQUEUE_EMPTY
 projdefs.h, 154
 errQUEUE_FULL
 projdefs.h, 155
 errQUEUE_YIELD
 projdefs.h, 155

 f32
 LSTD_TYPES.h, 337

 f64
 LSTD_TYPES.h, 337
 fire
 FLAG_t, 60
 flag
 SYSTEM_t, 67
 FLAG_t, 59
 buzzer, 60
 correctPassword, 60
 fire, 60
 openDoor, 60
 FreeRTOS.h
 configASSERT, 92
 configCHECK_FOR_STACK_OVERFLOW, 92
 configGENERATE_RUN_TIME_STATS, 92
 configUSE_ALTERNATIVE_API, 92
 configUSE_APPLICATION_TASK_TAG, 92
 configUSE_MALLOC_FAILED_HOOK, 92
 configUSE_MUTEXES, 92
 configUSE_RECURSIVE_MUTEXES, 92
 configUSE_TIMERS, 93
 INCLUDE_pcTaskGetTaskName, 93
 INCLUDE_uxTaskGetStackHighWaterMark, 93
 INCLUDE_xTaskGetCurrentTaskHandle, 93
 INCLUDE_xTaskGetIdleTaskHandle, 93
 INCLUDE_xTaskGetSchedulerState, 93
 INCLUDE_xTaskResumeFromISR, 93
 INCLUDE_xTimerGetTimerDaemonTaskHandle,
 93
 pdTASK_HOOK_CODE, 101
 portCLEAR_INTERRUPT_MASK_FROM_ISR, 94
 portCONFIGURE_TIMER_FOR_RUN_TIME_STATS,
 94
 portCRITICAL_NESTING_IN_TCB, 94
 portPOINTER_SIZE_TYPE, 94
 portPRIVILEGE_BIT, 94
 portSET_INTERRUPT_MASK_FROM_ISR, 94

 portYIELD_WITHIN_API, 94
 pvPortMallocAligned, 94
 traceBLOCKING_ON_QUEUE_RECEIVE, 95
 traceBLOCKING_ON_QUEUE_SEND, 95
 traceCREATE_COUNTING_SEMAPHORE, 95
 traceCREATE_COUNTING_SEMAPHORE FAILED,
 95
 traceCREATE_MUTEX, 95
 traceCREATE_MUTEX FAILED, 95
 traceEND, 95
 traceGIVE_MUTEX_RECURSIVE, 96
 traceGIVE_MUTEX_RECURSIVE FAILED, 96
 traceQUEUE_CREATE, 96
 traceQUEUE_CREATE FAILED, 96
 traceQUEUE_DELETE, 96
 traceQUEUE_PEEK, 96
 traceQUEUE_RECEIVE, 96
 traceQUEUE_RECEIVE FAILED, 97
 traceQUEUE_RECEIVE_FROM_ISR, 97
 traceQUEUE_RECEIVE_FROM_ISR FAILED, 97
 traceQUEUE_SEND, 97
 traceQUEUE_SEND FAILED, 97
 traceQUEUE_SEND_FROM_ISR, 97
 traceQUEUE_SEND_FROM_ISR FAILED, 97
 traceSTART, 98
 traceTAKE_MUTEX_RECURSIVE, 98
 traceTAKE_MUTEX_RECURSIVE FAILED, 98
 traceTASK_CREATE, 98
 traceTASK_CREATE FAILED, 98
 traceTASK_DELAY, 98
 traceTASK_DELAY_UNTIL, 98
 traceTASK_DELETE, 98
 traceTASK_INCREMENT_TICK, 99
 traceTASK_PRIORITY_SET, 99
 traceTASK_RESUME, 99
 traceTASK_RESUME_FROM_ISR, 99
 traceTASK_SUSPEND, 99
 traceTASK_SWITCHED_IN, 99
 traceTASK_SWITCHED_OUT, 99
 traceTIMER_COMMAND RECEIVED, 100
 traceTIMER_COMMAND_SEND, 100
 traceTIMER_CREATE, 100
 traceTIMER_CREATE FAILED, 100
 traceTIMER_EXPIRED, 100
 vPortFreeAligned, 100
 vQueueAddToRegistry, 100
 vQueueUnregisterQueue, 101

 FreeRTOSConfig.h
 configCPU_CLOCK_HZ, 108
 configIDLE_SHOULD_YIELD, 108
 configMAX_CO_ROUTINE_PRIORITIES, 108
 configMAX_PRIORITIES, 108
 configMAX_TASK_NAME_LEN, 108
 configMINIMAL_STACK_SIZE, 108
 configQUEUE_REGISTRY_SIZE, 108
 configTICK_RATE_HZ, 109
 configTOTAL_HEAP_SIZE, 109
 configUSE_16_BIT TICKS, 109

configUSE_CO_ROUTINES, 109
configUSE_COUNTING_SEMAPHORES, 109
configUSE_IDLE_HOOK, 109
configUSE_PREEMPTION, 109
configUSE_TICK_HOOK, 109
configUSE_TRACE_FACILITY, 110
INCLUDE_uxTaskPriorityGet, 110
INCLUDE_vTaskCleanUpResources, 110
INCLUDE_vTaskDelay, 110
INCLUDE_vTaskDelayUntil, 110
INCLUDE_vTaskDelete, 110
INCLUDE_vTaskPrioritySet, 110
INCLUDE_vTaskSuspend, 110
FUNCTION_SET_4BITS_1LINES
 HCLCD_Private.h, 310
FUNCTION_SET_4BITS_2LINES
 HCLCD_Private.h, 310
FUNCTION_SET_8BITS_1LINES
 HCLCD_Private.h, 310
FUNCTION_SET_8BITS_2LINES
 HCLCD_Private.h, 310

gas
 SYSTEM_INPUTS_t, 65
GET_BIT
 LBIT_MATH.h, 336

H_DcMotorInit
 DC_MOTOR.c, 282
 DC_MOTOR.h, 286
H_DcMotorPotInit
 DC_MOTOR_POT.c, 290
 DC_MOTOR_POT.h, 294
H_DcMotorPotStart
 DC_MOTOR_POT.c, 291
 DC_MOTOR_POT.h, 294
H_DcMotorPotStop
 DC_MOTOR_POT.c, 291
 DC_MOTOR_POT.h, 295
H_DcMotorSetDirection
 DC_MOTOR.h, 286
 DC_MOTOR_POT.h, 295
H_DcMotorSetSpeed
 DC_MOTOR.h, 286
 DC_MOTOR_POT.h, 296
H_DcMotorStart
 DC_MOTOR.c, 283
 DC_MOTOR.h, 287
H_DcMotorStop
 DC_MOTOR.c, 283
 DC_MOTOR.h, 287
H_ServoInit
 SERVO_interface.h, 331
 SERVO_Program.c, 333
H_ServoSetAngle
 SERVO_interface.h, 331
 SERVO_Program.c, 334
H_ServoStart
 SERVO_interface.h, 332

SERVO_Program.c, 334
H_ServoStop
 SERVO_interface.h, 332
 SERVO_Program.c, 334
HCLCD_Config.h
 CONTROL_PORT, 298
 DATA_PORT, 298
 E, 298
 HCLCD_DISPLAY_ON_OFF, 299
 HCLCD_ENTRY_MODE_SET, 299
 HCLCD_FUNCTION_SET, 299
 HCLCD_PINEND, 299
 HCLCD_PINSTART, 299
 RS, 299
HCLCD_DISPLAY_ON_OFF
 HCLCD_Config.h, 299
HCLCD_ENTRY_MODE_SET
 HCLCD_Config.h, 299
HCLCD_FUNCTION_SET
 HCLCD_Config.h, 299
HCLCD_Interface.h
 DISPLAY_CLEAR, 301
 HCLCD_LINE1, 301
 HCLCD_LINE2, 301
 HCLCD_Vid4Bits_Init, 301
 HCLCD_Vid8Bits_Init, 302
 HCLCD_VidSendChar_4Bits, 302
 HCLCD_VidSetPosition, 303
 HCLCD_VidSetPosition_4BitsMode, 303
 HCLCD_VidWriteChar_8Bits, 304
 HCLCD_VidWriteCommand_4Bits, 305
 HCLCD_VidWriteCommand_8Bits, 305
 HCLCD_VidWriteNumber_4Bits, 306
 HCLCD_VidWriteNumber_8Bits, 306
 HCLCD_VidWriteString_4Bits, 307
 HCLCD_VidWriteString_8Bits, 307
 HCLCD_LINE1
 HCLCD_Interface.h, 301
 HCLCD_LINE2
 HCLCD_Interface.h, 301
 HCLCD_PINEND
 HCLCD_Config.h, 299
 HCLCD_PINSTART
 HCLCD_Config.h, 299
 HCLCD_Private.h
 DISPLAY_OFF, 309
 DISPLAY_ON_CURSOR_OFF, 309
 DISPLAY_ON_CURSOR_ON_BLINKING_OFF,
 309
 DISPLAY_ON_CURSOR_ON_BLINKING_ON,
 309
 ENTRY_MODE_SET_DECREASE, 309
 ENTRY_MODE_SET_DECREASE_WITH_SHIFT,
 309
 ENTRY_MODE_SET_INCREASE, 310
 ENTRY_MODE_SET_INCREASE_WITH_SHIFT,
 310
 FUNCTION_SET_4BITS_1LINES, 310

FUNCTION_SET_4BITS_2LINES, 310
 FUNCTION_SET_8BITS_1LINES, 310
 FUNCTION_SET_8BITS_2LINES, 310
 HCLCD_VidWriteChar_4Bits, 311
 LINE1_OFFSET_ADDRESS, 310
 LINE2_OFFSET_ADDRESS, 310
HCLCD_Program.c
 HCLCD_Vid4Bits_Init, 312
 HCLCD_Vid8Bits_Init, 313
 HCLCD_VidSendChar_4Bits, 313
 HCLCD_VidsetPosition, 313
 HCLCD_VidsetPosition_4BitsMode, 314
 HCLCD_VidWriteChar_4Bits, 314
 HCLCD_VidWriteChar_8Bits, 315
 HCLCD_VidWriteCommand_4Bits, 316
 HCLCD_VidWriteCommand_8Bits, 316
 HCLCD_VidWriteNumber_4Bits, 317
 HCLCD_VidWriteNumber_8Bits, 317
 HCLCD_VidWriteString_4Bits, 318
 HCLCD_VidWriteString_8Bits, 318
HCLCD_Vid4Bits_Init
 HCLCD_Interface.h, 301
 HCLCD_Program.c, 312
HCLCD_Vid8Bits_Init
 HCLCD_Interface.h, 302
 HCLCD_Program.c, 313
HCLCD_VidSendChar_4Bits
 HCLCD_Interface.h, 302
 HCLCD_Program.c, 313
HCLCD_VidsetPosition
 HCLCD_Interface.h, 303
 HCLCD_Program.c, 313
HCLCD_VidsetPosition_4BitsMode
 HCLCD_Interface.h, 303
 HCLCD_Program.c, 314
HCLCD_VidWriteChar_4Bits
 HCLCD_Private.h, 311
 HCLCD_Program.c, 314
HCLCD_VidWriteChar_8Bits
 HCLCD_Interface.h, 304
 HCLCD_Program.c, 315
HCLCD_VidWriteCommand_4Bits
 HCLCD_Interface.h, 305
 HCLCD_Program.c, 316
HCLCD_VidWriteCommand_8Bits
 HCLCD_Interface.h, 305
 HCLCD_Program.c, 316
HCLCD_VidWriteNumber_4Bits
 HCLCD_Interface.h, 306
 HCLCD_Program.c, 317
HCLCD_VidWriteNumber_8Bits
 HCLCD_Interface.h, 306
 HCLCD_Program.c, 317
HCLCD_VidWriteString_4Bits
 HCLCD_Interface.h, 307
 HCLCD_Program.c, 318
HCLCD_VidWriteString_8Bits
 HCLCD_Interface.h, 307
 HCLCD_Program.c, 318
heap_1.c
 MPU_WRAPPERS_INCLUDED_FROM_API_FILE, 113
 pvPortMalloc, 113
 vPortFree, 113
 vPortInitialiseBlocks, 114
 xPortGetFreeHeapSize, 114
HIGH_NIBBLE
 macros.h, 129
HKPD_Config.h
 COL_END, 322
 COL_INIT, 322
 COL_PIN0, 322
 COL_PIN1, 322
 COL_PIN2, 322
 COL_PIN3, 322
 COL_PORT, 323
 ROW_END, 323
 ROW_INIT, 323
 ROW_PIN0, 323
 ROW_PIN1, 323
 ROW_PIN2, 323
 ROW_PIN3, 323
 ROW_PORT, 323
HKPD_Interface.h
 HKPD_U8GetKeyPressed, 325
 HKPD_VidInit, 325
 NOT_PRESSED, 325
HKPD_Program.c
 HKPD_U8GetKeyPressed, 327
 HKPD_VidInit, 327
 KPD_u8SwitchVal, 328
HKPD_U8GetKeyPressed
 HKPD_Interface.h, 325
 HKPD_Program.c, 327
HKPD_VidInit
 HKPD_Interface.h, 325
 HKPD_Program.c, 327
IN_1
 DC_MOTOR_CFG.h, 289
IN_1_POT
 DC_MOTOR_POT_CFG.h, 297
IN_2
 DC_MOTOR_CFG.h, 289
INCLUDE_pcTaskGetTaskName
 FreeRTOS.h, 93
INCLUDE_uxTaskGetStackHighWaterMark
 FreeRTOS.h, 93
INCLUDE_uxTaskPriorityGet
 FreeRTOSConfig.h, 110
INCLUDE_vTaskCleanUpResources
 FreeRTOSConfig.h, 110
INCLUDE_vTaskDelay
 FreeRTOSConfig.h, 110
INCLUDE_vTaskDelayUntil
 FreeRTOSConfig.h, 110
INCLUDE_vTaskDelete

FreeRTOSConfig.h, 110
INCLUDE_vTaskPrioritySet
 FreeRTOSConfig.h, 110
INCLUDE_vTaskSuspend
 FreeRTOSConfig.h, 110
INCLUDE_xTaskGetCurrentTaskHandle
 FreeRTOS.h, 93
INCLUDE_xTaskGetIdleTaskHandle
 FreeRTOS.h, 93
INCLUDE_xTaskGetSchedulerState
 FreeRTOS.h, 93
INCLUDE_xTaskResumeFromISR
 FreeRTOS.h, 93
INCLUDE_xTimerGetTimerDaemonTaskHandle
 FreeRTOS.h, 93
inputs
 SYSTEM_t, 67
KPD_u8SwitchVal
 HKPD_Program.c, 328
LBIT_MATH.h
 CLR_BIT, 336
 GET_BIT, 336
 SET_BIT, 336
 TOGGLE_BIT, 336
LINE1_OFFSET_ADDRESS
 HCLCD_Private.h, 310
LINE2_OFFSET_ADDRESS
 HCLCD_Private.h, 310
list.c
 vListInitialise, 117
 vListInitialiseItem, 117
 vListInsert, 117
 vListInsertEnd, 117
 vListRemove, 117
list.h
 listCURRENT_LIST_LENGTH, 121
 listGET_ITEM_VALUE_OF_HEAD_ENTRY, 121
 listGET_LIST_ITEM_VALUE, 121
 listGET_OWNER_OF_HEAD_ENTRY, 121
 listGET_OWNER_OF_NEXT_ENTRY, 122
 listIS_CONTAINED_WITHIN, 122
 listLIST_IS_EMPTY, 122
 listSET_LIST_ITEM_OWNER, 122
 listSET_LIST_ITEM_VALUE, 122
 vListInitialise, 123
 vListInitialiseItem, 123
 vListInsert, 123
 vListInsertEnd, 123
 vListRemove, 124
 xList, 123
 xListItem, 123
 xMiniListItem, 123
listCURRENT_LIST_LENGTH
 list.h, 121
listGET_ITEM_VALUE_OF_HEAD_ENTRY
 list.h, 121
listGET_LIST_ITEM_VALUE
 list.h, 121
list.h, 121
listGET_OWNER_OF_HEAD_ENTRY
 list.h, 121
listGET_OWNER_OF_NEXT_ENTRY
 list.h, 122
listIS_CONTAINED_WITHIN
 list.h, 122
listLIST_IS_EMPTY
 list.h, 122
listSET_LIST_ITEM_OWNER
 list.h, 122
listSET_LIST_ITEM_VALUE
 list.h, 122
LOW_NIBBLE
 macros.h, 129
LSTD_TYPES.h
 Error_State, 338
 f32, 337
 f64, 337
 NOK, 339
 NULL, 337
 NULL_POINTER, 337
 OK, 339
 s16, 338
 s32, 338
 s64, 338
 s8, 338
 u16, 338
 u32, 338
 u64, 338
 u8, 338
M_Pwm2Init
 TIMER2_Interface.h, 416
 TIMER2_Program.c, 424
M_Pwm2SetDutyCycle
 TIMER2_Interface.h, 416
 TIMER2_Program.c, 424
M_Pwm2Start
 TIMER2_Interface.h, 417
 TIMER2_Program.c, 424
M_Pwm2Stop
 TIMER2_Interface.h, 417
 TIMER2_Program.c, 425
macros.h
 CLR_BIT, 128
 HIGH_NIBBLE, 129
 LOW_NIBBLE, 129
 SET_BIT, 129
 TOG_BIT, 129
MADC_128_PRESCALER
 MADC_Private.h, 378
MADC_16_PRESCALER
 MADC_Private.h, 379
MADC_2_PRESCALER
 MADC_Private.h, 379
MADC_32_PRESCALER
 MADC_Private.h, 379
MADC_4_PRESCALER

MADC_Private.h, 379
 MADC_64_PRESCALER
 MADC_Private.h, 379
 MADC_8_PRESCALER
 MADC_Private.h, 379
 MADC_AVCC_REFERENCEVOLATGE
 MADC_Private.h, 379
 MADC_BIT_MASKING_CHANNEL
 MADC_Private.h, 379
 MADC_BIT_MASKING_PRESCALER
 MADC_Private.h, 380
 MADC_BIT_MASKING_REG_CHANNEL
 MADC_Private.h, 380
 MADC_CallBack
 MADC_Program.c, 383
 MADC_Config.h
 MADC_SET_ADJUST, 372
 MADC_SET_PRESCALER, 372
 MADC_SET_REFERENCEVOLATGE, 372
 MADC_Interface.h
 CHANNEL_0, 374
 CHANNEL_1, 374
 CHANNEL_2, 374
 CHANNEL_3, 374
 CHANNEL_4, 374
 CHANNEL_5, 374
 CHANNEL_6, 375
 CHANNEL_7, 375
 MADC_u16ADC_StartConversion, 375
 MADC_u16ADC_StartConversion_With_Interrupt,
 375
 MADC_u16ADCRead, 375
 MADC_VidADCInterruptEnable, 376
 MADC_VidInit, 376
 MADC_VidSetCallBack, 376
 MADC_INTERNAL_REFERENCEVOLATGE
 MADC_Private.h, 380
 MADC_LEFT_ADJUST
 MADC_Private.h, 380
 MADC_Private.h
 __vector_16, 380
 ADC, 378
 ADCSRA, 378
 ADMUX, 378
 MADC_128_PRESCALER, 378
 MADC_16_PRESCALER, 379
 MADC_2_PRESCALER, 379
 MADC_32_PRESCALER, 379
 MADC_4_PRESCALER, 379
 MADC_64_PRESCALER, 379
 MADC_8_PRESCALER, 379
 MADC_AVCC_REFERENCEVOLATGE, 379
 MADC_BIT_MASKING_CHANNEL, 379
 MADC_BIT_MASKING_PRESCALER, 380
 MADC_BIT_MASKING_REG_CHANNEL, 380
 MADC_INTERNAL_REFERENCEVOLATGE, 380
 MADC_LEFT_ADJUST, 380
 MADC_RIGHT_ADJUST, 380
 SFIOR, 380
 MADC_Program.c
 __vector_16, 382
 MADC_CallBack, 383
 MADC_u16ADC_StartConversion, 382
 MADC_u16ADC_StartConversion_With_Interrupt,
 382
 MADC_u16ADCRead, 382
 MADC_VidADCInterruptEnable, 382
 MADC_VidInit, 383
 MADC_VidSetCallBack, 383
 MADC_RIGHT_ADJUST
 MADC_Private.h, 380
 MADC_SET_ADJUST
 MADC_Config.h, 372
 MADC_SET_PRESCALER
 MADC_Config.h, 372
 MADC_SET_REFERENCEVOLATGE
 MADC_Config.h, 372
 MADC_u16ADC_StartConversion
 MADC_Interface.h, 375
 MADC_Program.c, 382
 MADC_u16ADC_StartConversion_With_Interrupt
 MADC_Interface.h, 375
 MADC_Program.c, 382
 MADC_u16ADCRead
 MADC_Interface.h, 375
 MADC_Program.c, 382
 MADC_VidADCInterruptEnable
 MADC_Interface.h, 376
 MADC_Program.c, 382
 MADC_VidInit
 MADC_Interface.h, 376
 MADC_Program.c, 383
 MADC_VidSetCallBack
 MADC_Interface.h, 376
 MADC_Program.c, 383
 main
 main.c, 345
 main.c
 ADC_Semaphore, 351
 ADC_SetNotification, 341
 comparePasswords, 341
 controlDoor, 341
 displayDoorState, 342
 displayEntryPasswordSecurity, 343
 displayFireState, 343
 displayTemperatureValue, 344
 main, 345
 System_Init, 346
 taskControlGasLevel, 346
 taskControlTemperature, 347
 taskDisplaySystemState, 348
 taskGetPassword, 349
 taskReadSensors, 350
 u8_AdcChannelToRead, 351
 main.h
 ADC_SetNotification, 357

comparePasswords, 357
controlDoor, 358
displayDoorState, 358
displayEntryPasswordSecurity, 359
displayFireState, 360
displayTemperatureValue, 361
System_Init, 361
taskControlGasLevel, 362
taskControlTemperature, 363
taskDisplaySystemState, 364
taskGetPassword, 365
taskReadSensors, 366
main_cfg.c
 system, 368
main_cfg.h
 PASSWORD_LENGTH, 370
 system, 371
maxAttempts
 PASSWORD_t, 61
MDIO_Error_State_GetPinValue
 MDIO_Interface.h, 387
 MDIO_Program.c, 393
MDIO_Error_State_SetNippleValue
 MDIO_Interface.h, 387
 MDIO_Program.c, 393
MDIO_Error_State_SetPinDirection
 MDIO_Interface.h, 387
 MDIO_Program.c, 394
MDIO_Error_State_SetPinValue
 MDIO_Interface.h, 388
 MDIO_Program.c, 394
MDIO_Error_State_SetPortDirection
 MDIO_Interface.h, 388
 MDIO_Program.c, 395
MDIO_Error_State_SetPortValue
 MDIO_Interface.h, 389
 MDIO_Program.c, 395
MDIO_Interface.h
 MDIO_Error_State_GetPinValue, 387
 MDIO_Error_State_SetNippleValue, 387
 MDIO_Error_State_SetPinDirection, 387
 MDIO_Error_State_SetPinValue, 388
 MDIO_Error_State_SetPortDirection, 388
 MDIO_Error_State_SetPortValue, 389
 MDIO_PORTA, 385
 MDIO_PORTB, 386
 MDIO_PORTC, 386
 MDIO_PORTD, 386
 PIN0, 387
 PIN1, 387
 PIN2, 387
 PIN3, 387
 PIN4, 387
 PIN5, 387
 PIN6, 387
 PIN7, 387
 PIN_HIGH, 386
 PIN_INPUT, 386
 PIN_LOW, 386
 PIN_OUTPUT, 386
 Pin_t, 386
 PORT_INPUT, 386
 PORT_OUTPUT, 386
MDIO_PORTA
 MDIO_Interface.h, 385
MDIO_PORTB
 MDIO_Interface.h, 386
MDIO_PORTC
 MDIO_Interface.h, 386
MDIO_PORTD
 MDIO_Interface.h, 386
MDIO_Private.h
 DDRA, 391
 DDRB, 391
 DDRC, 391
 DDRD, 391
 PINA, 391
 PINB, 391
 PINC, 391
 PIND, 391
 PORTA, 392
 PORTB, 392
 PORTC, 392
 PORTD, 392
MDIO_Program.c
 MDIO_Error_State_GetPinValue, 393
 MDIO_Error_State_SetNippleValue, 393
 MDIO_Error_State_SetPinDirection, 394
 MDIO_Error_State_SetPinValue, 394
 MDIO_Error_State_SetPortDirection, 395
 MDIO_Error_State_SetPortValue, 395
MGIE_Interface.h
 MGIE_VidDisable, 400
 MGIE_VidEnable, 400
MGIE_Private.h
 SREG, 401
MGIE_Program.c
 MGIE_VidDisable, 402
 MGIE_VidEnable, 402
MGIE_VidDisable
 MGIE_Interface.h, 400
 MGIE_Program.c, 402
MGIE_VidEnable
 MGIE_Interface.h, 400
 MGIE_Program.c, 402
mpu_wrappers.h
 portUSING_MPU_WRAPPERS, 130
 PRIVILEGED_DATA, 130
 PRIVILEGED_FUNCTION, 130
MPU_WRAPPERS_INCLUDED_FROM_API_FILE
 heap_1.c, 113
 queue.c, 158
 tasks.c, 219
 timers.c, 255
NOK
 LSTD_TYPES.h, 339

NOT_PRESSED
 HKPD_Interface.h, 325

NULL
 LSTD_TYPES.h, 337

NULL_POINTER
 LSTD_TYPES.h, 337

OCR0
 TIMER_Private.h, 407

OCR2
 TIMER2_Private.h, 420

OK
 LSTD_TYPES.h, 339

openDoor
 FLAG_t, 60

password
 SYSTEM_t, 67

PASSWORD_LENGTH
 main_cfg.h, 370

PASSWORD_t, 60
 attempts, 61
 maxAttempts, 61
 value, 61

pcHead
 QueueDefinition, 62

pcName
 xTASK_PARAMTERS, 74

pcReadFrom
 QueueDefinition, 62

pcTail
 QueueDefinition, 62

pcTaskGetTaskName
 task.h, 206

pcTaskName
 tskTaskControlBlock, 68

pcWriteTo
 QueueDefinition, 62

pdFAIL
 projdefs.h, 155

pdFALSE
 projdefs.h, 155

pdPASS
 projdefs.h, 155

pdTASK_CODE
 projdefs.h, 155

pdTASK_HOOK_CODE
 FreeRTOS.h, 101

pdTRUE
 projdefs.h, 155

PIN0
 MDIO_Interface.h, 387

PIN1
 MDIO_Interface.h, 387

PIN2
 MDIO_Interface.h, 387

PIN3
 MDIO_Interface.h, 387

PIN4
 MDIO_Interface.h, 387

PIN5
 MDIO_Interface.h, 387

PIN6
 MDIO_Interface.h, 387

PIN7
 MDIO_Interface.h, 387

PIN_HIGH
 MDIO_Interface.h, 386

PIN_INPUT
 MDIO_Interface.h, 386

PIN_LOW
 MDIO_Interface.h, 386

PIN_OUTPUT
 MDIO_Interface.h, 386

Pin_t
 MDIO_Interface.h, 386

PINA
 MDIO_Private.h, 391

PINB
 MDIO_Private.h, 391

PINC
 MDIO_Private.h, 391

PIND
 MDIO_Private.h, 391

port.c
 portCLEAR_COUNTER_ON_MATCH, 133
 portCLOCK_PRESCALER, 133
 portCOMPARE_MATCH_A_INTERRUPT_ENABLE,
 133
 portFLAGS_INT_ENABLED, 133
 portPRESCALE_64, 134
 portRESTORE_CONTEXT, 134
 portSAVE_CONTEXT, 134
 pxCurrentTCB, 135
 pxPortInitialiseStack, 134
 SIG_OUTPUT_COMPARE1A, 134
 tskTCB, 134
 vPortEndScheduler, 134
 vPortYield, 135
 vPortYieldFromTick, 135
 xPortStartScheduler, 135

PORT_INPUT
 MDIO_Interface.h, 386

PORT_OUTPUT
 MDIO_Interface.h, 386

PORTA
 MDIO_Private.h, 392

portable.h
 portBYTE_ALIGNMENT_MASK, 141
 portNUM_CONFIGURABLE_REGIONS, 141
 pvPortMalloc, 142
 pxPortInitialiseStack, 142
 vPortEndScheduler, 142
 vPortFree, 143
 vPortInitialiseBlocks, 143
 xPortGetFreeHeapSize, 143
 xPortStartScheduler, 143

PORTB
 MDIO_Private.h, 392

portBASE_TYPE
 portmacro.h, 149

portBYTE_ALIGNMENT
 portmacro.h, 149

portBYTE_ALIGNMENT_MASK
 portable.h, 141

PORTC
 MDIO_Private.h, 392

portCHAR
 portmacro.h, 149

portCLEAR_COUNTER_ON_MATCH
 port.c, 133

portCLEAR_INTERRUPT_MASK_FROM_ISR
 FreeRTOS.h, 94

portCLOCK_PRESCALER
 port.c, 133

portCOMPARE_MATCH_A_INTERRUPT_ENABLE
 port.c, 133

portCONFIGURE_TIMER_FOR_RUN_TIME_STATS
 FreeRTOS.h, 94

portCRITICAL_NESTING_IN_TCB
 FreeRTOS.h, 94

PORTD
 MDIO_Private.h, 392

portDISABLE_INTERRUPTS
 portmacro.h, 149

portDOUBLE
 portmacro.h, 149

portENABLE_INTERRUPTS
 portmacro.h, 150

portENTER_CRITICAL
 portmacro.h, 150

portEXIT_CRITICAL
 portmacro.h, 150

portFLAGS_INT_ENABLED
 port.c, 133

portFLOAT
 portmacro.h, 150

portLONG
 portmacro.h, 150

portmacro.h
 portBASE_TYPE, 149
 portBYTE_ALIGNMENT, 149
 portCHAR, 149
 portDISABLE_INTERRUPTS, 149
 portDOUBLE, 149
 portENABLE_INTERRUPTS, 150
 portENTER_CRITICAL, 150
 portEXIT_CRITICAL, 150
 portFLOAT, 150
 portLONG, 150
 portMAX_DELAY, 150
 portNOP, 150
 portSHORT, 151
 portSTACK_GROWTH, 151
 portSTACK_TYPE, 151

portTASK_FUNCTION, 151

portTASK_FUNCTION_PROTO, 151

portTICK_RATE_MS, 151

portTickType, 152

portYIELD, 151

vPortYield, 152

portMAX_DELAY
 portmacro.h, 150

portNOP
 portmacro.h, 150

portNUM_CONFIGURABLE_REGIONS
 portable.h, 141

portPOINTER_SIZE_TYPE
 FreeRTOS.h, 94

portPRESCALE_64
 port.c, 134

portPRIVILEGE_BIT
 FreeRTOS.h, 94

portRESTORE_CONTEXT
 port.c, 134

portSAVE_CONTEXT
 port.c, 134

portSET_INTERRUPT_MASK_FROM_ISR
 FreeRTOS.h, 94

portSHORT
 portmacro.h, 151

portSTACK_GROWTH
 portmacro.h, 151

portSTACK_TYPE
 portmacro.h, 151

portTASK_FUNCTION
 portmacro.h, 151

portTASK_FUNCTION_PROTO
 portmacro.h, 151

portTICK_RATE_MS
 portmacro.h, 151

portTickType
 portmacro.h, 152

portUSING_MPU_WRAPPERS
 mpu_wrappers.h, 130

portYIELD
 portmacro.h, 151

portYIELD_WITHIN_API
 FreeRTOS.h, 94

PRIVILEGED_DATA
 mpu_wrappers.h, 130

PRIVILEGED_FUNCTION
 mpu_wrappers.h, 130

projdefs.h
 errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY,
 154
 errNO_TASK_TO_RUN, 154
 errQUEUE_BLOCKED, 154
 errQUEUE_EMPTY, 154
 errQUEUE_FULL, 155
 errQUEUE_YIELD, 155
 pdFAIL, 155
 pdFALSE, 155

pdPASS, 155
 pdTASK_CODE, 155
 pdTRUE, 155
 prvAddCoRoutineToReadyQueue
 croutine.c, 76
 prvAddTaskToReadyQueue
 tasks.c, 219
 prvCheckDelayedTasks
 tasks.c, 219
 prvGetTCBFromHandle
 tasks.c, 219
 prvLockQueue
 queue.c, 158
 puxStackBuffer
 xTASK_PARAMTERS, 74
 pvBaseAddress
 xMEMORY_REGION, 72
 pvContainer
 xLIST_ITEM, 71
 pvOwner
 xLIST_ITEM, 71
 pvParameters
 xTASK_PARAMTERS, 74
 pvPortMalloc
 heap_1.c, 113
 portable.h, 142
 pvPortMallocAligned
 FreeRTOS.h, 94
 pvTaskCode
 xTASK_PARAMTERS, 74
 pvTimerGetTimerID
 timers.h, 276
 pxCoRoutineFunction
 corCoRoutineControlBlock, 58
 pxCurrentCoRoutine
 croutine.c, 78
 pxCurrentTCB
 port.c, 135
 tasks.c, 225
 pxIndex
 xLIST, 69
 pxMutexHolder
 queue.c, 158
 pxNext
 xLIST_ITEM, 71
 xMINI_LIST_ITEM, 73
 pxPortInitialiseStack
 port.c, 134
 portable.h, 142
 pxPrevious
 xLIST_ITEM, 71
 xMINI_LIST_ITEM, 73
 pxStack
 tskTaskControlBlock, 68
 pxTopOfStack
 tskTaskControlBlock, 68
 queue.c
 MPU_WRAPPERS_INCLUDED_FROM_API_FILE,
 158
 prvLockQueue, 158
 pxMutexHolder, 158
 queueDONT_BLOCK, 158
 queueERRONEOUS_UNBLOCK, 159
 queueLOCKED_UNMODIFIED, 159
 queueMUTEX_GIVE_BLOCK_TIME, 159
 queueQUEUE_IS_MUTEX, 159
 queueSEMAPHORE_QUEUE_ITEM_LENGTH,
 159
 queueSEND_TO_BACK, 159
 queueSEND_TO_FRONT, 159
 queueUNLOCKED, 159
 uxQueueMessagesWaiting, 160
 uxQueueMessagesWaitingFromISR, 160
 uxQueueType, 160
 uxRecursiveCallCount, 160
 vQueueDelete, 160
 xQUEUE, 160
 xQueueCreate, 161
 xQueueCreateCountingSemaphore, 161
 xQueueCreateMutex, 161
 xQueueGenericReceive, 162
 xQueueGenericSend, 162
 xQueueGenericSendFromISR, 162
 xQueueHandle, 160
 xQueueIsQueueEmptyFromISR, 162
 xQueueIsQueueFullFromISR, 162
 xQueueReceiveFromISR, 162
 queue.h
 queueSEND_TO_BACK, 183
 queueSEND_TO_FRONT, 183
 uxQueueMessagesWaiting, 186
 uxQueueMessagesWaitingFromISR, 186
 vQueueDelete, 186
 vQueueWaitForMessageRestricted, 187
 xQueueAltGenericReceive, 187
 xQueueAltGenericSend, 187
 xQueueAltPeek, 183
 xQueueAltReceive, 184
 xQueueAltSendToBack, 184
 xQueueAltSendToFront, 184
 xQueueCreate, 187
 xQueueCreateCountingSemaphore, 187
 xQueueCreateMutex, 188
 xQueueCRReceive, 188
 xQueueCRReceiveFromISR, 188
 xQueueCRSend, 188
 xQueueCRSendFromISR, 188
 xQueueGenericReceive, 188
 xQueueGenericSend, 189
 xQueueGenericSendFromISR, 189
 xQueueGiveMutexRecursive, 189
 xQueueHandle, 186
 xQueueIsQueueEmptyFromISR, 189
 xQueueIsQueueFullFromISR, 189
 xQueuePeek, 184

xQueueReceive, 184
xQueueReceiveFromISR, 189
xQueueSend, 185
xQueueSendFromISR, 185
xQueueSendToBack, 185
xQueueSendToBackFromISR, 185
xQueueSendToFront, 185
xQueueSendToFrontFromISR, 186
xQueueTakeMutexRecursive, 190
QueueDefinition, 61
 pcHead, 62
 pcReadFrom, 62
 pcTail, 62
 pcWriteTo, 62
 uxItemSize, 62
 uxLength, 63
 uxMessagesWaiting, 63
 xRxLock, 63
 xTasksWaitingToReceive, 63
 xTasksWaitingToSend, 63
 xTxLock, 63
queueDONT_BLOCK
 queue.c, 158
queueERRONEOUS_UNBLOCK
 queue.c, 159
queueLOCKED_UNMODIFIED
 queue.c, 159
queueMUTEX_GIVE_BLOCK_TIME
 queue.c, 159
queueQUEUE_IS_MUTEX
 queue.c, 159
queueSEMAPHORE_QUEUE_ITEM_LENGTH
 queue.c, 159
queueSEND_TO_BACK
 queue.c, 159
 queue.h, 183
queueSEND_TO_FRONT
 queue.c, 159
 queue.h, 183
queueUNLOCKED
 queue.c, 159

ROW_END
 HKPD_Config.h, 323
ROW_INIT
 HKPD_Config.h, 323
ROW_PIN0
 HKPD_Config.h, 323
ROW_PIN1
 HKPD_Config.h, 323
ROW_PIN2
 HKPD_Config.h, 323
ROW_PIN3
 HKPD_Config.h, 323
ROW_PORT
 HKPD_Config.h, 323
RS
 HCLCD_Config.h, 299

s16
 LSTD_TYPES.h, 338
s32
 LSTD_TYPES.h, 338
s64
 LSTD_TYPES.h, 338
s8
 LSTD_TYPES.h, 338
semBINARY_SEMAPHORE_QUEUE_LENGTH
 semphr.h, 194
semGIVE_BLOCK_TIME
 semphr.h, 194
semphr.h
 semBINARY_SEMAPHORE_QUEUE_LENGTH,
 194
 semGIVE_BLOCK_TIME, 194
 semSEMAPHORE_QUEUE_ITEM_LENGTH, 194
 vSemaphoreCreateBinary, 194
 vSemaphoreDelete, 195
 xSemaphoreAltGive, 195
 xSemaphoreAltTake, 195
 xSemaphoreCreateCounting, 195
 xSemaphoreCreateMutex, 195
 xSemaphoreCreateRecursiveMutex, 196
 xSemaphoreGive, 196
 xSemaphoreGiveFromISR, 196
 xSemaphoreGiveRecursive, 196
 xSemaphoreHandle, 197
 xSemaphoreTake, 196
 xSemaphoreTakeRecursive, 196
semSEMAPHORE_QUEUE_ITEM_LENGTH
 semphr.h, 194
SENSOR_t, 63
 u16_CurrentValueInBinary, 64
 u8_AdcChannel, 64
 u8_CriticalValue, 64
 u8_CurrentValue, 64
 u8_MaxValue, 64
SERVO_interface.h
 H_ServolInit, 331
 H_ServoSetAngle, 331
 H_ServoStart, 332
 H_ServoStop, 332
SERVO_Program.c
 H_ServolInit, 333
 H_ServoSetAngle, 334
 H_ServoStart, 334
 H_ServoStop, 334
SET_BIT
 LBIT_MATH.h, 336
 macros.h, 129
SFIOR
 MADC_Private.h, 380
SIG_OUTPUT_COMPARE1A
 port.c, 134
SREG
 MGIE_Private.h, 401
StackMacros.h

taskFIRST_CHECK_FOR_STACK_OVERFLOW,
 199
taskSECOND_CHECK_FOR_STACK_OVERFLOW,
 199
system
 main_cfg.c, 368
 main_cfg.h, 371
System_Init
 main.c, 346
 main.h, 361
SYSTEM_INPUTS_t, 65
 gas, 65
 temperature, 65
SYSTEM_t, 66
 flag, 67
 inputs, 67
 password, 67

task.h
 pcTaskGetTaskName, 206
 taskDISABLE_INTERRUPTS, 204
 taskENABLE_INTERRUPTS, 204
 taskENTER_CRITICAL, 204
 taskEXIT_CRITICAL, 205
 taskSCHEDULER_NOT_STARTED, 205
 taskSCHEDULER_RUNNING, 205
 taskSCHEDULER_SUSPENDED, 205
 taskYIELD, 205
 tskIDLE_PRIORITY, 205
 tskKERNEL_VERSION_NUMBER, 205
 ulTaskEndTrace, 206
 uxTaskGetNumberOfTasks, 207
 uxTaskGetStackHighWaterMark, 207
 uxTaskPriorityGet, 207
 vTaskAllocateMPURegions, 207
 vTaskDelay, 207
 vTaskDelayUntil, 208
 vTaskDelete, 208
 vTaskEndScheduler, 208
 vTaskGetRunTimeStats, 209
 vTaskIncrementTick, 209
 vTaskList, 209
 vTaskMissedYield, 209
 vTaskPlaceOnEventList, 209
 vTaskPlaceOnEventListRestricted, 209
 vTaskPriorityDisinherit, 209
 vTaskPriorityInherit, 209
 vTaskPrioritySet, 210
 vTaskResume, 210
 vTaskSetTimeOutState, 210
 vTaskStartScheduler, 210
 vTaskStartTrace, 210
 vTaskSuspend, 211
 vTaskSuspendAll, 211
 vTaskSwitchContext, 211
 xMemoryRegion, 206
 xTaskCallApplicationTaskHook, 211
 xTaskCheckForTimeOut, 211
 xTaskCreate, 205
 xTaskCreateRestricted, 206
 xTaskGenericCreate, 212
 xTaskGetCurrentTaskHandle, 212
 xTaskGetIdleTaskHandle, 212
 xTaskGetSchedulerState, 212
 xTaskGetTickCount, 212
 xTaskGetTickCountFromISR, 212
 xTaskHandle, 206
 xTaskIsTaskSuspended, 212
 xTaskParameters, 206
 xTaskRemoveFromEventList, 213
 xTaskResumeAll, 213
 xTaskResumeFromISR, 213
 xTimeOutType, 206
taskControlGasLevel
 main.c, 346
 main.h, 362
taskControlTemperature
 main.c, 347
 main.h, 363
taskDISABLE_INTERRUPTS
 task.h, 204
taskDisplaySystemState
 main.c, 348
 main.h, 364
taskENABLE_INTERRUPTS
 task.h, 204
taskENTER_CRITICAL
 task.h, 204
taskEXIT_CRITICAL
 task.h, 205
taskFIRST_CHECK_FOR_STACK_OVERFLOW
 StackMacros.h, 199
taskGetPassword
 main.c, 349
 main.h, 365
taskReadSensors
 main.c, 350
 main.h, 366
tasks.c
 MPU_WRAPPERS_INCLUDED_FROM_API_FILE,
 219
 prvAddTaskToReadyQueue, 219
 prvCheckDelayedTasks, 219
 prvGetTCBFromHandle, 219
 pxCurrentTCB, 225
 tskBLOCCED_CHAR, 220
 tskDELETED_CHAR, 220
 tskIDLE_STACK_SIZE, 220
 tskREADY_CHAR, 220
 tskSTACK_FILL_BYTE, 220
 tskSUSPENDED_CHAR, 220
 tskTCB, 221
 uxTaskGetNumberOfTasks, 221
 vApplicationStackOverflowHook, 221
 vApplicationTickHook, 221
 vTaskDelay, 221
 vTaskDelayUntil, 221

vTaskDelete, 222
vTaskEndScheduler, 222
vTaskIncrementTick, 222
vTaskMissedYield, 222
vTaskPlaceOnEventList, 222
vTaskSetTimeOutState, 223
vTaskStartScheduler, 223
vTaskSuspendAll, 223
vTaskSwitchContext, 224
vWriteTraceToBuffer, 220
xTaskCheckForTimeOut, 224
xTaskGetTickCount, 224
xTaskGetTickCountFromISR, 224
xTaskRemoveFromEventList, 224
xTaskResumeAll, 224
taskSCHEDULER_NOT_STARTED
 task.h, 205
taskSCHEDULER_RUNNING
 task.h, 205
taskSCHEDULER_SUSPENDED
 task.h, 205
taskSECOND_CHECK_FOR_STACK_OVERFLOW
 StackMacros.h, 199
taskYIELD
 task.h, 205
TCCR0
 TIMER_Private.h, 407
TCCR2
 TIMER2_Private.h, 420
TCNT0
 TIMER_Private.h, 407
TCNT2
 TIMER2_Private.h, 420
temperature
 SYSTEM_INPUTS_t, 65
TIMER0_CallBack
 TIMER_Program.c, 412
TIMER0_CTC_INTERRUPT_DISABLED
 TIMER_Private.h, 407
TIMER0_CTC_INTERRUPT_ENABLED
 TIMER_Private.h, 407
TIMER0_CTC_MODE
 TIMER_Private.h, 408
TIMER0_FAST_PWM_MODE
 TIMER_Private.h, 408
TIMER0_NORMAL_MODE
 TIMER_Private.h, 408
TIMER0_OCO_PIN_CLR
 TIMER_Private.h, 408
TIMER0_OCO_PIN_DISCONNECTED
 TIMER_Private.h, 408
TIMER0_OCO_PIN_SET
 TIMER_Private.h, 408
TIMER0_OCO_PIN_TOGGLE
 TIMER_Private.h, 408
TIMER0_PHASECORRECT_PWM_MODE
 TIMER_Private.h, 408
TIMER0_SET_CTC_INTERRUPT
 TIMER_Config.h, 403
 TIMER0_SET_MODE
 TIMER_Config.h, 404
 TIMER0_SET_OCO_PIN_MODE
 TIMER_Config.h, 404
 TIMER0_SET_PWM_MODE
 TIMER_Config.h, 404
 TIMER0_VidCTC_SetCallBack
 TIMER_Interface.h, 405
 TIMER_Program.c, 411
 TIMER0_VidInit
 TIMER_Interface.h, 405
 TIMER_Program.c, 411
 TIMER0_VidOVF_SetCallBack
 TIMER_Interface.h, 405
 TIMER_Program.c, 411
 TIMER0_VidSetCTCValue
 TIMER_Interface.h, 405
 TIMER_Program.c, 412
 TIMER0_VidSetPreload
 TIMER_Interface.h, 406
 TIMER_Program.c, 412
 TIMER2_CallBack
 TIMER2_Program.c, 426
 TIMER2_Config.h
 TIMER2_SET_CTC_INTERRUPT, 414
 TIMER2_SET_MODE, 414
 TIMER2_SET_OC2_PIN_MODE, 414
 TIMER2_SET_PWM_MODE, 414
 TIMER_SET_PRESCALER, 415
 TIMER2_CTC_INTERRUPT_DISABLED
 TIMER2_Private.h, 420
 TIMER2_CTC_INTERRUPT_ENABLED
 TIMER2_Private.h, 420
 TIMER2_CTC_MODE
 TIMER2_Private.h, 420
 TIMER2_FAST_PWM_MODE
 TIMER2_Private.h, 421
 TIMER2_Interface.h
 M_Pwm2Init, 416
 M_Pwm2SetDutyCycle, 416
 M_Pwm2Start, 417
 M_Pwm2Stop, 417
 TIMER2_VidCTC_SetCallBack, 417
 TIMER2_VidInit, 418
 TIMER2_VidOVF_SetCallBack, 418
 TIMER2_VidSetCTCValue, 418
 TIMER2_VidSetPreload, 418
 TIMER2_NORMAL_MODE
 TIMER2_Private.h, 421
 TIMER2_OC2_PIN_CLR
 TIMER2_Private.h, 421
 TIMER2_OC2_PIN_DISCONNECTED
 TIMER2_Private.h, 421
 TIMER2_OC2_PIN_SET
 TIMER2_Private.h, 421
 TIMER2_OC2_PIN_TOGGLE
 TIMER2_Private.h, 421

TIMER2_PHASECORRECT_PWM_MODE
 TIMER2_Private.h, 421
 TIMER2_Private.h
 OCR2, 420
 TCCR2, 420
 TCNT2, 420
 TIMER2_CTC_INTERRUPT_DISABLED, 420
 TIMER2_CTC_INTERRUPT_ENABLED, 420
 TIMER2_CTC_MODE, 420
 TIMER2_FAST_PWM_MODE, 421
 TIMER2_NORMAL_MODE, 421
 TIMER2_OC2_PIN_CLR, 421
 TIMER2_OC2_PIN_DISCONNECTED, 421
 TIMER2_OC2_PIN_SET, 421
 TIMER2_OC2_PIN_TOGGLE, 421
 TIMER2_PHASECORRECT_PWM_MODE, 421
 TIMER_1024_PRESCALER, 421
 TIMER_256_PRESCALER, 422
 TIMER_64_PRESCALER, 422
 TIMER_8_PRESCALER, 422
 TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE, TIMER0_SET_OC0_PIN_MODE, 404
 422
 TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE, TIMER_SET_PRESCALER, 404
 422
 TIMER_NO_PRESCALER, 422
 TIMER_STOPPED, 422
 TIMSK, 422
 TIMER2_Program.c
 __vector_4, 424
 __vector_5, 424
 M_Pwm2Init, 424
 M_Pwm2SetDutyCycle, 424
 M_Pwm2Start, 424
 M_Pwm2Stop, 425
 TIMER2_CallBack, 426
 TIMER2_VidCTC_SetCallBack, 425
 TIMER2_VidInit, 425
 TIMER2_VidOVF_SetCallBack, 426
 TIMER2_VidSetCTCValue, 426
 TIMER2_VidSetPreload, 426
 TIMER2_SET_CTC_INTERRUPT
 TIMER2_Config.h, 414
 TIMER2_SET_MODE
 TIMER2_Config.h, 414
 TIMER2_SET_OC2_PIN_MODE
 TIMER2_Config.h, 414
 TIMER2_SET_PWM_MODE
 TIMER2_Config.h, 414
 TIMER2_VidCTC_SetCallBack
 TIMER2_Interface.h, 417
 TIMER2_Program.c, 425
 TIMER2_VidInit
 TIMER2_Interface.h, 418
 TIMER2_Program.c, 425
 TIMER2_VidOVF_SetCallBack
 TIMER2_Interface.h, 418
 TIMER2_Program.c, 426
 TIMER2_VidSetCTCValue
 TIMER2_Interface.h, 418
 TIMER2_Program.c, 426
 TIMER2_Interface.h, 418
 TIMER2_Program.c, 426
 TIMER2_VidSetPreload
 TIMER2_Interface.h, 418
 TIMER2_Program.c, 426
 TIMER_1024_PRESCALER
 TIMER2_Private.h, 421
 TIMER_Private.h, 409
 TIMER_256_PRESCALER
 TIMER2_Private.h, 422
 TIMER_Private.h, 409
 TIMER_64_PRESCALER
 TIMER2_Private.h, 422
 TIMER_Private.h, 409
 TIMER_8_PRESCALER
 TIMER2_Private.h, 422
 TIMER_Private.h, 409
 TIMER_Config.h
 TIMER0_SET_CTC_INTERRUPT, 403
 TIMER0_SET_MODE, 404
 TIMER0_SET_OC0_PIN_MODE, 404
 TIMER0_SET_PWM_MODE, 404
 TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE, TIMER0_SET_PRESCALER, 404
 422
 TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE, TIMER2_Private.h, 422
 TIMER_Private.h, 409
 TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE
 TIMER2_Private.h, 422
 TIMER_Private.h, 409
 TIMER_INTERFACE.h
 TIMER0_VidCTC_SetCallBack, 405
 TIMER0_VidInit, 405
 TIMER0_VidOVF_SetCallBack, 405
 TIMER0_VidSetCTCValue, 405
 TIMER0_VidSetPreload, 406
 TIMER_NO_PRESCALER
 TIMER2_Private.h, 422
 TIMER_Private.h, 409
 TIMER_Private.h
 OCR0, 407
 TCCR0, 407
 TCNT0, 407
 TIMER0_CTC_INTERRUPT_DISABLED, 407
 TIMER0_CTC_INTERRUPT_ENABLED, 407
 TIMER0_CTC_MODE, 408
 TIMER0_FAST_PWM_MODE, 408
 TIMER0_NORMAL_MODE, 408
 TIMER0_OC0_PIN_CLR, 408
 TIMER0_OC0_PIN_DISCONNECTED, 408
 TIMER0_OC0_PIN_SET, 408
 TIMER0_OC0_PIN_TOGGLE, 408
 TIMER0_PHASECORRECT_PWM_MODE, 408
 TIMER_1024_PRESCALER, 409
 TIMER_256_PRESCALER, 409
 TIMER_64_PRESCALER, 409
 TIMER_8_PRESCALER, 409
 TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE, 409

TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE
409
TIMER_NO_PRESCALER, 409
TIMER_STOPPED, 409
TIMSK, 410
TIMER_Program.c
 __vector_10, 411
 __vector_11, 411
 TIMER0_CallBack, 412
 TIMER0_VidCTC_SetCallBack, 411
 TIMER0_VidInit, 411
 TIMER0_VidOVF_SetCallBack, 411
 TIMER0_VidSetCTCValue, 412
 TIMER0_VidSetPreload, 412
TIMER_SET_PRESCALER
 TIMER2_Config.h, 415
 TIMER_Config.h, 404
TIMER_STOPPED
 TIMER2_Private.h, 422
 TIMER_Private.h, 409
timers.c
 MPU_WRAPPERS_INCLUDED_FROM_API_FILE,
 255
timers.h
 pvTimerGetTimerID, 276
 tmrCOMMAND_CHANGE_PERIOD, 265
 tmrCOMMAND_DELETE, 265
 tmrCOMMAND_START, 266
 tmrCOMMAND_STOP, 266
 tmrTIMER_CALLBACK, 276
 xTimerChangePeriod, 266
 xTimerChangePeriodFromISR, 267
 xTimerCreate, 277
 xTimerCreateTimerTask, 278
 xTimerDelete, 268
 xTimerGenericCommand, 279
 xTimerGetTimerDaemonTaskHandle, 279
 xTimerHandle, 276
 xTimerIsTimerActive, 279
 xTimerReset, 269
 xTimerResetFromISR, 270
 xTimerStart, 272
 xTimerStartFromISR, 273
 xTimerStop, 274
 xTimerStopFromISR, 275
TIMSK
 TIMER2_Private.h, 422
 TIMER_Private.h, 410
tmrCOMMAND_CHANGE_PERIOD
 timers.h, 265
tmrCOMMAND_DELETE
 timers.h, 265
tmrCOMMAND_START
 timers.h, 266
tmrCOMMAND_STOP
 timers.h, 266
tmrTIMER_CALLBACK
 timers.h, 276
 TOGGLE_BIT
 macros.h, 129
 TOGGLE_BIT
 LBIT_MATH.h, 336
traceBLOCKING_ON_QUEUE_RECEIVE
 FreeRTOS.h, 95
traceBLOCKING_ON_QUEUE_SEND
 FreeRTOS.h, 95
traceCREATE_COUNTING_SEMAPHORE
 FreeRTOS.h, 95
traceCREATE_COUNTING_SEMAPHORE_FAILED
 FreeRTOS.h, 95
traceCREATE_MUTEX
 FreeRTOS.h, 95
traceCREATE_MUTEX_FAILED
 FreeRTOS.h, 95
traceEND
 FreeRTOS.h, 95
traceGIVE_MUTEX_RECURSIVE
 FreeRTOS.h, 96
traceGIVE_MUTEX_RECURSIVE_FAILED
 FreeRTOS.h, 96
traceQUEUE_CREATE
 FreeRTOS.h, 96
traceQUEUE_CREATE_FAILED
 FreeRTOS.h, 96
traceQUEUE_DELETE
 FreeRTOS.h, 96
traceQUEUE_PEEK
 FreeRTOS.h, 96
traceQUEUE_RECEIVE
 FreeRTOS.h, 96
traceQUEUE_RECEIVE_FAILED
 FreeRTOS.h, 97
traceQUEUE_RECEIVE_FROM_ISR
 FreeRTOS.h, 97
traceQUEUE_RECEIVE_FROM_ISR_FAILED
 FreeRTOS.h, 97
traceQUEUE_SEND
 FreeRTOS.h, 97
traceQUEUE_SEND_FAILED
 FreeRTOS.h, 97
traceQUEUE_SEND_FROM_ISR
 FreeRTOS.h, 97
traceQUEUE_SEND_FROM_ISR_FAILED
 FreeRTOS.h, 97
traceSTART
 FreeRTOS.h, 98
traceTAKE_MUTEX_RECURSIVE
 FreeRTOS.h, 98
traceTAKE_MUTEX_RECURSIVE_FAILED
 FreeRTOS.h, 98
traceTASK_CREATE
 FreeRTOS.h, 98
traceTASK_CREATE_FAILED
 FreeRTOS.h, 98
traceTASK_DELAY
 FreeRTOS.h, 98

traceTASK_DELAY_UNTIL
 FreeRTOS.h, 98

traceTASK_DELETE
 FreeRTOS.h, 98

traceTASK_INCREMENT_TICK
 FreeRTOS.h, 99

traceTASK_PRIORITY_SET
 FreeRTOS.h, 99

traceTASK_RESUME
 FreeRTOS.h, 99

traceTASK_RESUME_FROM_ISR
 FreeRTOS.h, 99

traceTASK_SUSPEND
 FreeRTOS.h, 99

traceTASK_SWITCHED_IN
 FreeRTOS.h, 99

traceTASK_SWITCHED_OUT
 FreeRTOS.h, 99

traceTIMER_COMMAND_RECEIVED
 FreeRTOS.h, 100

traceTIMER_COMMAND_SEND
 FreeRTOS.h, 100

traceTIMER_CREATE
 FreeRTOS.h, 100

traceTIMER_CREATE_FAILED
 FreeRTOS.h, 100

traceTIMER_EXPIRED
 FreeRTOS.h, 100

tskBLOCCED_CHAR
 tasks.c, 220

tskDELETED_CHAR
 tasks.c, 220

tskIDLE_PRIORITY
 task.h, 205

tskIDLE_STACK_SIZE
 tasks.c, 220

tskKERNEL_VERSION_NUMBER
 task.h, 205

tskREADY_CHAR
 tasks.c, 220

tskSTACK_FILL_BYTE
 tasks.c, 220

tskSUSPENDED_CHAR
 tasks.c, 220

tskTaskControlBlock, 67
 pcTaskName, 68
 pxStack, 68
 pxTopOfStack, 68
 uxPriority, 68
 xEventListItem, 68
 xGenericListItem, 68

tskTCB
 port.c, 134
 tasks.c, 221

u16
 LSTD_TYPES.h, 338

u16_CurrentValueInBinary
 SENSOR_t, 64

u32
 LSTD_TYPES.h, 338

u64
 LSTD_TYPES.h, 338

u8
 LSTD_TYPES.h, 338

u8_AdcChannel
 SENSOR_t, 64

u8_AdcChannelToRead
 main.c, 351

u8_CriticalValue
 SENSOR_t, 64

u8_CurrentValue
 SENSOR_t, 64

u8_MaxValue
 SENSOR_t, 64

ulLengthInBytes
 xMEMORY_REGION, 72

ulParameters
 xMEMORY_REGION, 72

ulTaskEndTrace
 task.h, 206

usStackDepth
 xTASK_PARAMTERS, 74

uxIndex
 corCoRoutineControlBlock, 58

uxItemSize
 QueueDefinition, 62

uxLength
 QueueDefinition, 63

uxMessagesWaiting
 QueueDefinition, 63

uxNumberOfItems
 xLIST, 70

uxPriority
 corCoRoutineControlBlock, 59
 tskTaskControlBlock, 68
 xTASK_PARAMTERS, 74

uxQueueMessagesWaiting
 queue.c, 160
 queue.h, 186

uxQueueMessagesWaitingFromISR
 queue.c, 160
 queue.h, 186

uxQueueType
 queue.c, 160

uxRecursiveCallCount
 queue.c, 160

uxState
 corCoRoutineControlBlock, 59

uxTaskGetNumberOfTasks
 task.h, 207
 tasks.c, 221

uxTaskGetStackHighWaterMark
 task.h, 207

uxTaskPriorityGet, 51
 task.h, 207

value

PASSWORD_t, 61
vApplicationStackOverflowHook
 tasks.c, 221
vApplicationTickHook
 tasks.c, 221
vCoRoutineAddToDelayedList
 croutine.c, 77
 croutine.h, 87
vCoRoutineSchedule, 11
 croutine.c, 77
 croutine.h, 87
vListInitialise
 list.c, 117
 list.h, 123
vListInitialiseItem
 list.c, 117
 list.h, 123
vListInsert
 list.c, 117
 list.h, 123
vListInsertEnd
 list.c, 117
 list.h, 123
vListRemove
 list.c, 117
 list.h, 124
vPortEndScheduler
 port.c, 134
 portable.h, 142
vPortFree
 heap_1.c, 113
 portable.h, 143
vPortFreeAligned
 FreeRTOS.h, 100
vPortInitialiseBlocks
 heap_1.c, 114
 portable.h, 143
vPortYield
 port.c, 135
 portmacro.h, 152
vPortYieldFromTick
 port.c, 135
vQueueAddToRegistry
 FreeRTOS.h, 100
vQueueDelete
 queue.c, 160
 queue.h, 186
vQueueUnregisterQueue
 FreeRTOS.h, 101
vQueueWaitForMessageRestricted
 queue.h, 187
vSemaphoreCreateBinary, 36
 semphr.h, 194
vSemaphoreCreateMutex, 43
vSemaphoreDelete
 semphr.h, 195
vTaskAllocateMPURegions
 task.h, 207
vTaskDelay, 49
 task.h, 207
 tasks.c, 221
vTaskDelayUntil, 50
 task.h, 208
 tasks.c, 221
vTaskDelete, 49
 task.h, 208
 tasks.c, 222
vTaskEndScheduler, 55
 task.h, 208
 tasks.c, 222
vTaskGetRunTimeStats
 task.h, 209
vTaskIncrementTick
 task.h, 209
 tasks.c, 222
vTaskList
 task.h, 209
vTaskMissedYield
 task.h, 209
 tasks.c, 222
vTaskPlaceOnEventList
 task.h, 209
 tasks.c, 222
vTaskPlaceOnEventListRestricted
 task.h, 209
vTaskPriorityDisinherit
 task.h, 209
vTaskPriorityInherit
 task.h, 209
vTaskPrioritySet, 52
 task.h, 210
vTaskResume, 53
 task.h, 210
vTaskResumeFromISR, 54
vTaskSetTimeOutState
 task.h, 210
 tasks.c, 223
vTaskStartScheduler, 55
 task.h, 210
 tasks.c, 223
vTaskStartTrace
 task.h, 210
vTaskSuspend, 52
 task.h, 211
vTaskSuspendAll, 56
 task.h, 211
 tasks.c, 223
vTaskSwitchContext
 task.h, 211
 tasks.c, 224
vWriteTraceToBuffer
 tasks.c, 220
xCORoutineCreate, 10
 croutine.c, 77
 croutine.h, 87
xCORoutineHandle

croutine.h, 86
 xCoRoutineRemoveFromEventList
 croutine.c, 77
 croutine.h, 87
 xEventListItem
 corCoRoutineControlBlock, 59
 tskTaskControlBlock, 68
 xGenericListItem
 corCoRoutineControlBlock, 59
 tskTaskControlBlock, 68
 xItemValue
 xLIST_ITEM, 71
 xMINI_LIST_ITEM, 73
 xLIST, 69
 pxIndex, 69
 uxNumberOfItems, 70
 xListEnd, 70
 xList
 list.h, 123
 xLIST_ITEM, 70
 pvContainer, 71
 pvOwner, 71
 pNext, 71
 pPrevious, 71
 xItemValue, 71
 xListEnd
 xLIST, 70
 xListItem
 list.h, 123
 xMEMORY_REGION, 71
 pvBaseAddress, 72
 ulLengthInBytes, 72
 ulParameters, 72
 xMemoryRegion
 task.h, 206
 xMINI_LIST_ITEM, 72
 pNext, 73
 pPrevious, 73
 xItemValue, 73
 xMiniListItem
 list.h, 123
 xOverflowCount
 xTIME_OUT, 75
 xPortGetFreeHeapSize
 heap_1.c, 114
 portable.h, 143
 xPortStartScheduler
 port.c, 135
 portable.h, 143
 xQUEUE
 queue.c, 160
 xQueueAltGenericReceive
 queue.h, 187
 xQueueAltGenericSend
 queue.h, 187
 xQueueAltPeek
 queue.h, 183
 xQueueAltReceive
 queue.h, 184
 xQueueAltSendToBack
 queue.h, 184
 xQueueAltSendToFront
 queue.h, 184
 xQueueCreate, 19
 queue.c, 161
 queue.h, 187
 xQueueCreateCountingSemaphore
 queue.c, 161
 queue.h, 187
 xQueueCreateMutex
 queue.c, 161
 queue.h, 188
 xQueueCRReceive
 queue.h, 188
 xQueueCRReceiveFromISR
 queue.h, 188
 xQueueCRSend
 queue.h, 188
 xQueueCRSendFromISR
 queue.h, 188
 xQueueGenericReceive
 queue.c, 162
 queue.h, 188
 xQueueGenericSend
 queue.c, 162
 queue.h, 189
 xQueueGenericSendFromISR
 queue.c, 162
 queue.h, 189
 xQueueGiveMutexRecursive
 queue.h, 189
 xQueueHandle
 queue.c, 160
 queue.h, 186
 xQueueIsQueueEmptyFromISR
 queue.c, 162
 queue.h, 189
 xQueueIsQueueFullFromISR
 queue.c, 162
 queue.h, 189
 xQueuePeek
 queue.h, 184
 xQueueReceive, 26
 queue.h, 184
 xQueueReceiveFromISR, 34
 queue.c, 162
 queue.h, 189
 xQueueSend, 20
 queue.h, 185
 xQueueSendFromISR, 30
 queue.h, 185
 xQueueSendToBack
 queue.h, 185
 xQueueSendToBackFromISR
 queue.h, 185
 xQueueSendToFront

queue.h, 185
xQueueSendToFrontFromISR
 queue.h, 186
xQueueTakeMutexRecursive
 queue.h, 190
xRegions
 xTASK_PARAMTERS, 75
xRxLock
 QueueDefinition, 63
xSemaphoreAltGive
 semphr.h, 195
xSemaphoreAltTake
 semphr.h, 195
xSemaphoreCreateCounting, 44
 semphr.h, 195
xSemaphoreCreateMutex
 semphr.h, 195
xSemaphoreCreateRecursiveMutex
 semphr.h, 196
xSemaphoreGive, 39
 semphr.h, 196
xSemaphoreGiveFromISR, 41
 semphr.h, 196
xSemaphoreGiveRecursive, 40
 semphr.h, 196
xSemaphoreHandle
 semphr.h, 197
xSemaphoreTake, 36
 semphr.h, 196
xSemaphoreTakeRecursive, 37
 semphr.h, 196
xTASK_PARAMTERS, 73
 pcName, 74
 pxStackBuffer, 74
 pvParameters, 74
 pvTaskCode, 74
 usStackDepth, 74
 uxPriority, 74
 xRegions, 75
xTaskCallApplicationTaskHook
 task.h, 211
xTaskCheckForTimeOut
 task.h, 211
 tasks.c, 224
xTaskCreate, 45
 task.h, 205
xTaskCreateRestricted, 47
 task.h, 206
xTaskGenericCreate
 task.h, 212
xTaskGetCurrentTaskHandle
 task.h, 212
xTaskGetIdleTaskHandle
 task.h, 212
xTaskGetSchedulerState
 task.h, 212
xTaskGetTickCount
 task.h, 212
 tasks.c, 224
xTaskGetTickCountFromISR
 task.h, 212
 tasks.c, 224
xTaskHandle
 task.h, 206
xTaskIsTaskSuspended
 task.h, 212
xTaskParameters
 task.h, 206
xTaskRemoveFromEventList
 task.h, 213
 tasks.c, 224
xTaskResumeAll, 57
 task.h, 213
 tasks.c, 224
xTaskResumeFromISR
 task.h, 213
xTasksWaitingToReceive
 QueueDefinition, 63
xTasksWaitingToSend
 QueueDefinition, 63
xTIME_OUT, 75
 xOverflowCount, 75
 xTimeOnEntering, 75
xTimeOnEntering
 xTIME_OUT, 75
xTimeOutType
 task.h, 206
xTimerChangePeriod
 timers.h, 266
xTimerChangePeriodFromISR
 timers.h, 267
xTimerCreate
 timers.h, 277
xTimerCreateTimerTask
 timers.h, 278
xTimerDelete
 timers.h, 268
xTimerGenericCommand
 timers.h, 279
xTimerGetTimerDaemonTaskHandle
 timers.h, 279
xTimerHandle
 timers.h, 276
xTimerIsTimerActive
 timers.h, 279
xTimerReset
 timers.h, 269
xTimerResetFromISR
 timers.h, 270
xTimerStart
 timers.h, 272
xTimerStartFromISR
 timers.h, 273
xTimerStop
 timers.h, 274
xTimerStopFromISR

timers.h, 275
xTxLock
QueueDefinition, 63