Tasks

Each task requires RAM that is used to hold the task state, and used by the task as its stack. If a task is created using xTaskCreate() then the required RAM is automatically allocated from the FreeRTOS heap. If a task is created using xTaskCreateStatic() then the RAM is provided by the application writer, so it can be statically allocated at compile time. See the Static Vs Dynamic allocation page for more information.

If you are using FreeRTOS-MPU then we recommend that you use xTaskCreateRestricted() instead of xTaskCreate().

Parameters:

- *pvTaskCode*
  Pointer to the task entry function (just the name of the function that implements the task, see the example below). Tasks are normally implemented as an infinite loop; the function which implements the task must never attempt to return or exit. Tasks can, however, delete themselves.
- *pcName*
  A descriptive name for the task. This is mainly used to facilitate debugging, but can also be used to obtain a task handle. The maximum length of a task's name is defined by configMAX_TASK_NAME_LEN in FreeRTOSConfig.h.

```
BaseType_t xTaskCreate( TaskFunction_t pvTaskCode,
                        const char * const pcName,
                        const configSTACK_DEPTH_TYPE uxStackDepth,
                        void *pvParameters,
                        UBaseType_t uxPriority,
                        TaskHandle_t *pxCreatedTask
                      );
```

The priority at which the created task will execute. Systems that include MPU support can optionally create a task in a privileged (system) mode by setting the bit portPRIVILEGE_BIT in uxPriority. For example, to create a privileged task at priority 2 set uxPriority to (2 | portPRIVILEGE_BIT). Priorities are asserted to be less than configMAX_PRIORITIES. If configASSERT is undefined, priorities are silently capped at (configMAX_PRIORITIES - 1)

- *pxCreatedTask*
  Used to pass a handle to the created task out of the xTaskCreate() function. pxCreatedTask is optional and can be set to NULL.

```
/* Task to be created. */
void vTaskCode( void * pvParameters )
{
    /* The parameter value is expected to be 1 as 1 is passed in
the
       pvParameters value in the call to xTaskCreate() below. */

    configASSERT( ( ( uint32_t ) pvParameters ) == 1 );

    for( ;; )
    {
        /* Task code goes here. */
    }
}

/* Function that creates a task. */
void vOtherFunction( void )
{
    BaseType_t xReturned;
    TaskHandle_t xHandle = NULL;

    /* Create the task, storing the handle. */
    xReturned = xTaskCreate(
                    vTaskCode,       /* Function that implements
the task. */
                    "NAME",          /* Text name for the task. */
                    STACK_SIZE,      /* Stack size in words, not
bytes. */
                    ( void * ) 1,    /* Parameter passed into the
task. */
                    tskIDLE_PRIORITY,/* Priority at which the task
is created. */
                    &xHandle );      /* Used to pass out the
created task's handle. */

    if( xReturned == pdPASS )
    {
        /* The task was created. Use the task's handle to delete
the task. */
        vTaskDelete( xHandle );
    }
}
```