xTimerCreate

[Timer API]

timers.h

```
TimerHandle_t xTimerCreate
            ( const char * const pcTimerName,
              const TickType_t xTimerPeriod,
              const UBaseType_t uxAutoReload,
              void * const pvTimerID,
              TimerCallbackFunction_t 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:

pcTimerName	A text name that is assigned to the timer. This is done purely to assist debugging. The RTOS kernel itself only ever references a timer by its handle, and never by its name.
xTimerPeriod	The timer period. The time is defined in tick periods so the constant portTICK_PERIOD_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_PERIOD_MS) provided configTICK_RATE_HZ is less than or equal to 1000.
uxAutoReload	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 and enter the dormant state after it expires.
pvTimerID	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, or together with the vTimerSetTimerID() and pvTimerGetTimerID() API functions to save a value between calls to the timer's callback function.
pxCallbackFunction	The function to call when the timer expires. Callback functions must have the prototype defined by TimerCallbackFunction_t, which is "void vCallbackFunction(TimerHandle_t 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 NULL is returned.

Example usage:

```
#define NUM_TIMERS 5

/* An array to hold handles to the created timers. */
TimerHandle_t 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( TimerHandle_t 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 RTOS
    scheduler has been started means the timers will start running
    immediately that the RTOS scheduler starts. */
    for( x = 0; x < NUM_TIMERS; x++ )
    {
        xTimers[ x ] = xTimerCreate
                  (  /* Just a text name, not used by the RTOS kernel. */
                    "Timer",
                    /* The timer period in ticks, must be greater than 0. */
                    ( 100 * x ) + 100,
                    /* The timers will auto-reload themselves when they
                    expire. */
                    pdTRUE,
                    /* Assign each timer a unique id equal to its array
                    index. */
                    ( void * ) x,
                    /* Each timer calls the same callback when it expires. */
                    vTimerCallback
                  );

        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 RTOS 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 RTOS scheduler will start the timers running as they have
    already been set into the active state. */
    vTaskStartScheduler();

    /* Should not reach here. */
    for( ;; );
}
```

[Back to the top] [About FreeRTOS] [Sitemap] [Report an error on this page]