5.3 Deep Dive: FreeRTOS Software Timers

Creating, Starting, and Stopping Timers

Interacting with FreeRTOS software timers is done through a standard set of API functions. The core steps are to create a timer, start it, and, if needed, stop, reset, or delete it.

Creating a Timer

A software timer is created using the `xTimerCreate()` function. This function does not start the timer; it only allocates the necessary resources and returns a handle that you will use to reference the timer in other API calls.

The function signature is:

```
TimerHandle_t xTimerCreate( const char * const pcTimerName,
                            const TickType_t xTimerPeriodInTicks,
                            const UBaseType_t uxAutoReload,
                            void * const pvTimerID,
                            TimerCallbackFunction_t pxCallbackFunction );
```

Parameters:

1. `pcTimerName`: A descriptive name for the timer, used mainly for debugging.
2. `xTimerPeriodInTicks`: The timer's period in system ticks. You can use the `pdMS_TO_TICKS()` macro to easily convert milliseconds to ticks.
3. `uxAutoReload`: Set to `pdTRUE` for an auto-reload timer or `pdFALSE` for a one-shot timer.
4. `pvTimerID`: A unique identifier for the timer. This is an application-defined value that can be used within the callback function to determine which timer has expired.
5. `pxCallbackFunction`: A pointer to the function that will be executed when the timer expires.

It is crucial to **always check the return value** of `xTimerCreate()`. If it returns `NULL`, the timer could not be created, most likely due to insufficient FreeRTOS heap memory.

Controlling a Timer

Once you have a valid timer handle, you can control it with the following functions:

- **To start or restart a timer:**
  `xTimerStart(TimerHandle_t xTimer, TickType_t xBlockTime)`
  This places the timer into the active state. If the timer was already running, it will be reset to its initial period.
- **To stop a timer:**
  `xTimerStop(TimerHandle_t xTimer, TickType_t xBlockTime)`
  This stops the timer from running.
- **To reset a timer:**
  `xTimerReset(TimerHandle_t xTimer, TickType_t xBlockTime)`
  This is equivalent to calling xTimerStart() on a running timer. It resets the timer's period back to its starting value.
- **To delete a timer:**
  `xTimerDelete(TimerHandle_t xTimer, TickType_t xBlockTime)`
  This frees the memory allocated when the timer was created. Once deleted, the handle is no longer valid.

The xBlockTime parameter in these functions specifies how long the calling task should wait if the command cannot be sent to the timer daemon task immediately (because its command queue is full). Using `portMAX_DELAY` will cause the task to wait indefinitely, which is a safe option in most cases.

3.2 One-Shot vs. Auto-Reload Timers

FreeRTOS offers two types of software timers, defined at creation time by the uxAutoReload parameter.

One-Shot Timer (`uxAutoReload` = `pdFALSE`)

A one-shot timer will execute its callback function **only once** after its period expires. It is useful for performing a single, delayed action.

- **Example Use Case:** You want to turn off a motor 10 seconds after it has been started.

Example Creation:

```
TimerHandle_t xOneShotTimer;

void vOneShotCallback(TimerHandle_t xTimer); // Forward declaration

void setup() {
  xOneShotTimer = xTimerCreate(
      "OneShot",                // Timer name
      pdMS_TO_TICKS(2000),      // 2000ms period
      pdFALSE,                  // Set as a one-shot timer
```

```
      (void *) 0,               // Timer ID = 0
      vOneShotCallback          // Callback function
  );

  if (xOneShotTimer != NULL) {
    xTimerStart(xOneShotTimer, 0);
  }
}
```

Auto-Reload Timer (`uxAutoReload` = `pdTRUE`)

An auto-reload timer will execute its callback function **repeatedly** at a fixed interval. After the callback is executed, the timer automatically resets and starts counting down again.

- **Example Use Case:** You need to read a sensor and print its value every 1000 milliseconds.

Example Creation:

```
TimerHandle_t xAutoReloadTimer;

void vAutoReloadCallback(TimerHandle_t xTimer); // Forward declaration

void setup() {
  xAutoReloadTimer = xTimerCreate(
      "AutoReload",             // Timer name
      pdMS_TO_TICKS(1000),      // 1000ms period
      pdTRUE,                   // Set as an auto-reload timer
      (void *) 1,               // Timer ID = 1
      vAutoReloadCallback       // Callback function
  );

  if (xAutoReloadTimer != NULL) {
    xTimerStart(xAutoReloadTimer, 0);
  }
}
```

3.3 Writing Effective Timer Callback Functions

The callback function is the heart of the software timer. It's the code that runs when the timer expires. To ensure system stability, it must be written carefully.

The function must have the following signature:

```
void YourCallbackName(TimerHandle_t xTimer);
```

The single parameter, `xTimer`, is the handle of the timer that just expired. This is very useful when a single callback function is used for multiple timers. You can retrieve the Timer ID you assigned during creation to identify which timer it was.

Example Callback Implementation:

```
void vTimerCallback(TimerHandle_t xTimer) {
  // Get the ID of the timer that expired
  uint32_t ulTimerID = (uint32_t) pvTimerGetTimerID(xTimer);

  // Check which timer it was and perform an action
  if (ulTimerID == 0) {
    // This was the one-shot timer
    Serial.println("One-shot timer expired.");
  } else if (ulTimerID == 1) {
    // This was the auto-reload timer
    Serial.println("Auto-reload timer expired.");
  }
}
```

Rules for Writing Callback Functions

Timer callbacks execute in the context of the FreeRTOS timer daemon task, not an ISR. However, they share similar restrictions because multiple callbacks may need to execute in sequence.

1. **Keep them short and fast.** A long-running callback will delay the execution of other pending timer callbacks.
2. **Never block.** Do not call any function that could block, such as `vTaskDelay()` or waiting on a semaphore or queue with a long timeout. Doing so will halt the timer daemon task, preventing any other software timers in the system from running.

Revision #1
Created 2025-08-28 12:43:57 UTC by GI
Updated 2025-08-28 12:54:15 UTC by GI