FreeRTOS Timers

Norman McEntire
norman.mcentire@gmail.com

Copyright (c) 2019 Servin Corp

Textbook Reference

- Mastering the FreeRTOS Real Time Kernel by Richard Barry

  - Chapter 5: Software Timer Management

Copyright (c) 2019 Servin Corp

Topics

- 5.1 Intro and Scope
- 5.2 Software Timer Callback Functions
- 5.3 Attributes and States of a Software Timer
- 5.4 The Context of a Software Timer
- 5.5 Creating and Starting a Software Timer
- 5.6 The Timer ID
- 5.7 Changing the Period of a Timer
- 5.8 Resetting a Software Timer

Copyright (c) 2019 Servin Corp

5.1 Intro and Scope

- Two options for using software timers
  - Option 1.
    - Schedule the execution of a function at a **set time** in the future
  - Option 2.
    - Schedule the execution of a function **periodically** with a fixed frequency

Copyright (c) 2019 Servin Corp

Software Timers Implemented by FreeRTOS

- Software Timers are implemented by FreeRTOS
  - They are optional
    - configSOFTWARE_TIMERS must be set to 1 in FreeRTOSConfig.h
  - They do NOT require hardware support
    - They are NOT related to hardware timers
      - They do NOT use any processing time unless the callback is executing

Copyright (c) 2019 Servin Corp

What you will learn

- Characteristics of software timer compared to task
- The RTOS daemon task
- The timer command queue
- The difference between a one shot timer and a periodic
- How to create, start, reset, and change the period of a timer

Copyright (c) 2019 Servin Corp

5.2 Software Timer Callback Functions

- Software timer implemented a C function with this prototype
    - void MyTimerCallback(TimerHandle_t xTimer)
- Timer functions execute from start to finish
    - Must be kept short, and never enter Blocked state
- Note: Software timer functions execute in the context of a daemon task created by the FreeRTOS Scheduler

Copyright (c) 2019 Servin Corp

To repeat: Software Timers cannot call blocking APIs

- OK to call from Software Timer
  - xQueueReceive() with xTicksToWait set to 0 (no waiting)
- Not OK to call
  - vTaskDelay()

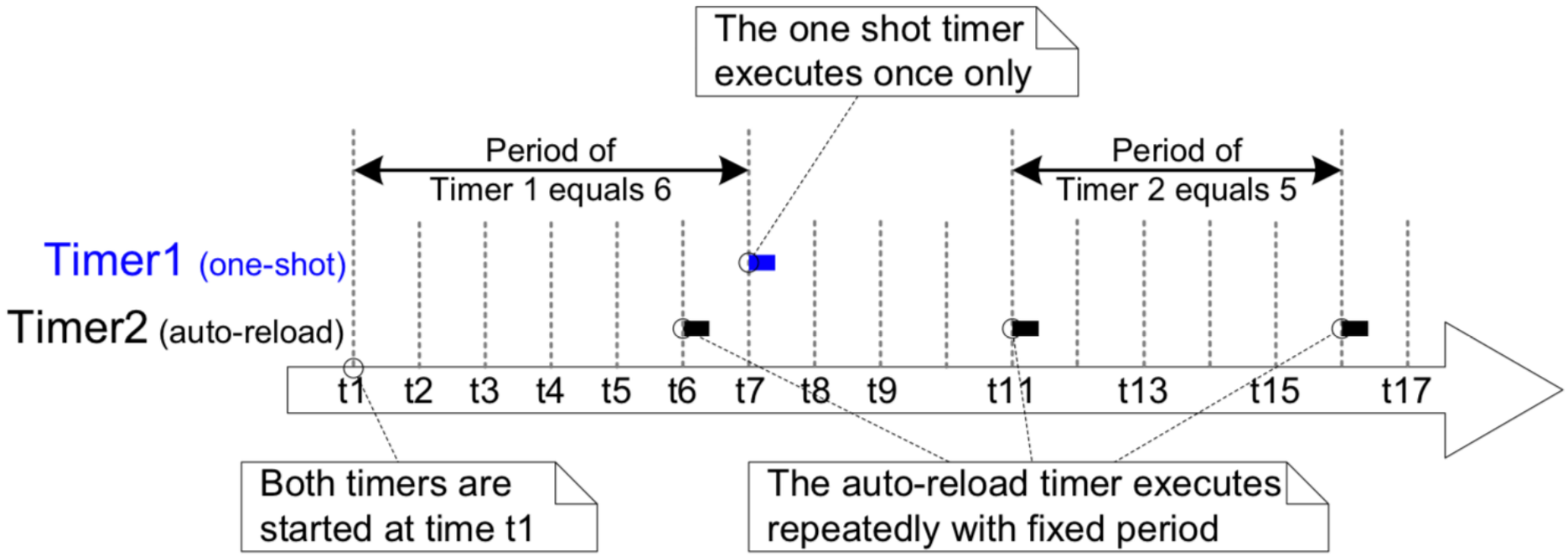
Copyright (c) 2019 Servin Corp

5.3 Attributes and States of a Software Timer

- Period of a software timer
  - The time between the software timer being started and the timers callback function executing
- One-Shot Timer
  - Once started, only calls callback function once
- Auto-Reload Timer
  - Auto starts after callback function called

Copyright (c) 2019 Servin Corp

Block Diagram



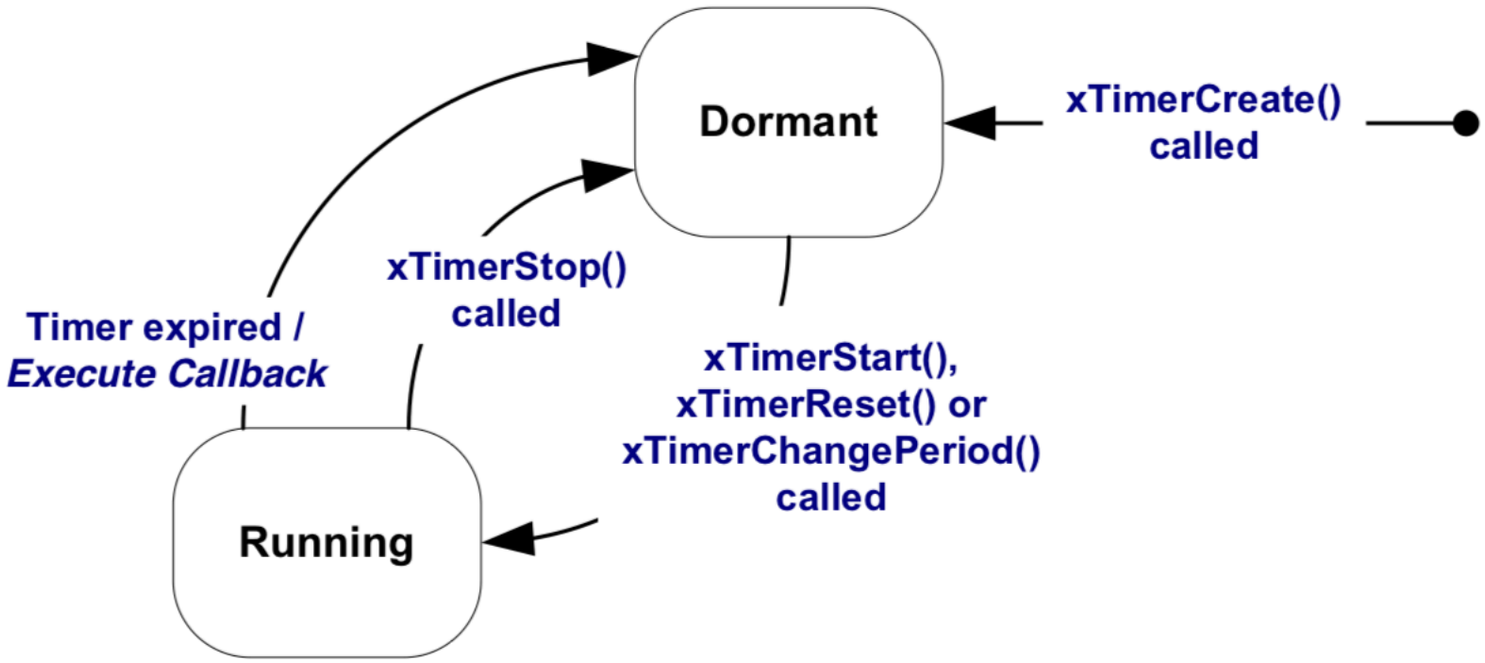
Copyright (c) 2019 Servin Corp

Software Timer State

- Software Timers have two states
  - Dormant
    - Timer exists but callback function not running
- Running
  - Callback function is executing

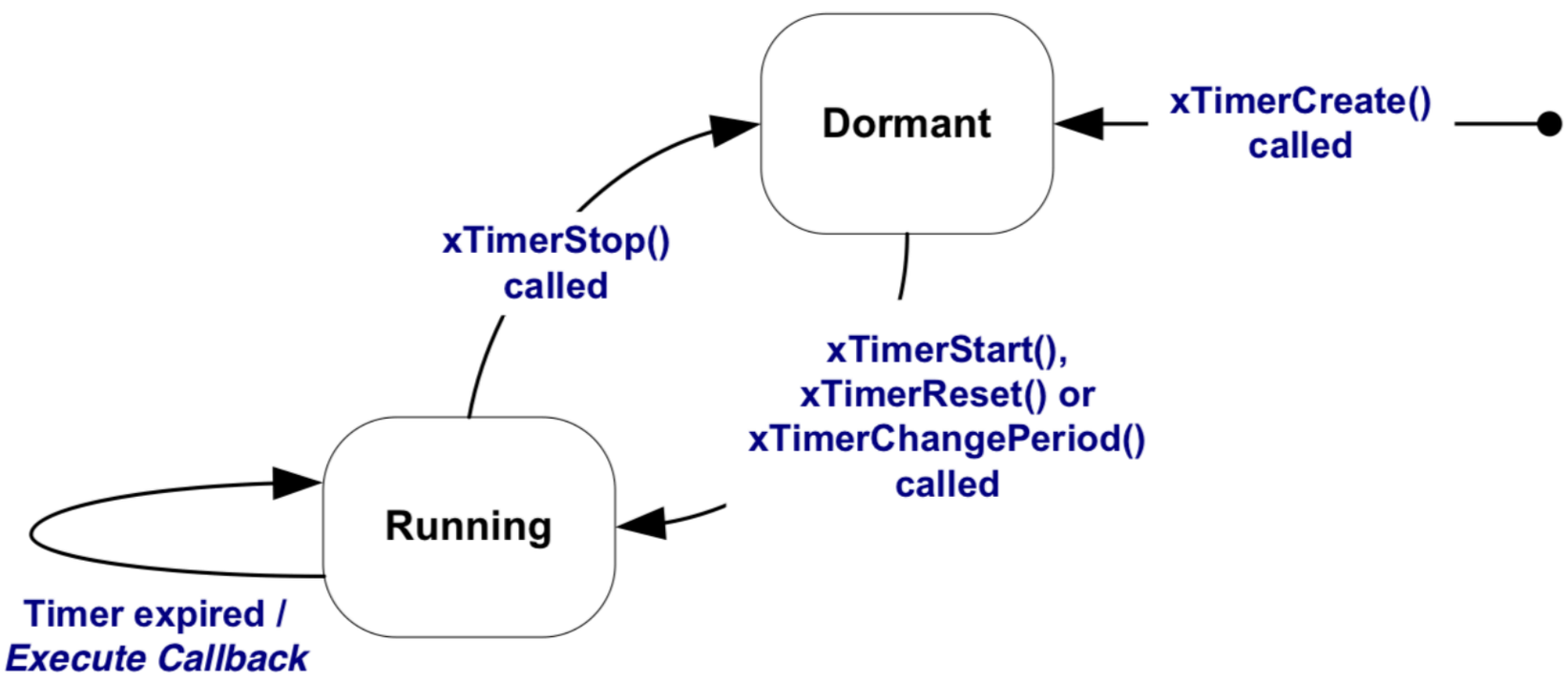
Copyright (c) 2019 Servin Corp

Block Diagram
One-Shot Timer



Copyright (c) 2019 Servin Corp

Block Diagram
Auto Reload Timer



Copyright (c) 2019 Servin Corp

5.4 The Context of a Software Timer

- All software timers execute in the context of the FreeRTOS daemon task
  - Also called the “timer service” task
- FreeRTOS daemon task created automatically when the scheduler starts
- Configuration of FreeRTOS daemon task
  - configTIMER_TASK_PRIORITY
  - configTIMER_TASK_STACK_DEPTH

Copyright (c) 2019 Servin Corp

Timer Command Queue

- Software Timer API functions send commands from the calling task to the daemon task on a queue
  - The “timer command queue”
    - Standard FreeRTOS queue created when Scheduler starts
      - configTIMER_QUEUE_LENGTH
- Example commands
  - Start a Timer
  - Stop a Timer
  - Reset a Timer

Copyright (c) 2019 Servin Corp

Block Diagram

Application Code

```
/* A function implemented in
an application task. */
void vAFunction( void )
{
    /* Write function code
    here. */
    ....
    /* At some point the
    xTimerReset() API
    function is called.
    The implementation of
    xTimerReset() writes to
    the timer command queue.
    */
    xTimerReset();

    /* Write the rest of the
    function code here. */
}
```

FreeRTOS (kernel) Code

```
/* A pseudo representation
of the FreeRTOS daemon task.
This is not the real
code! */
void prvTimerTask( ... )
{
    for( ;; )
    {
        /* Wait for a
        command. */
        xQueueReceive();

        /* Process the
        command. */
    }
}
```

The API function writes to the timer command queue

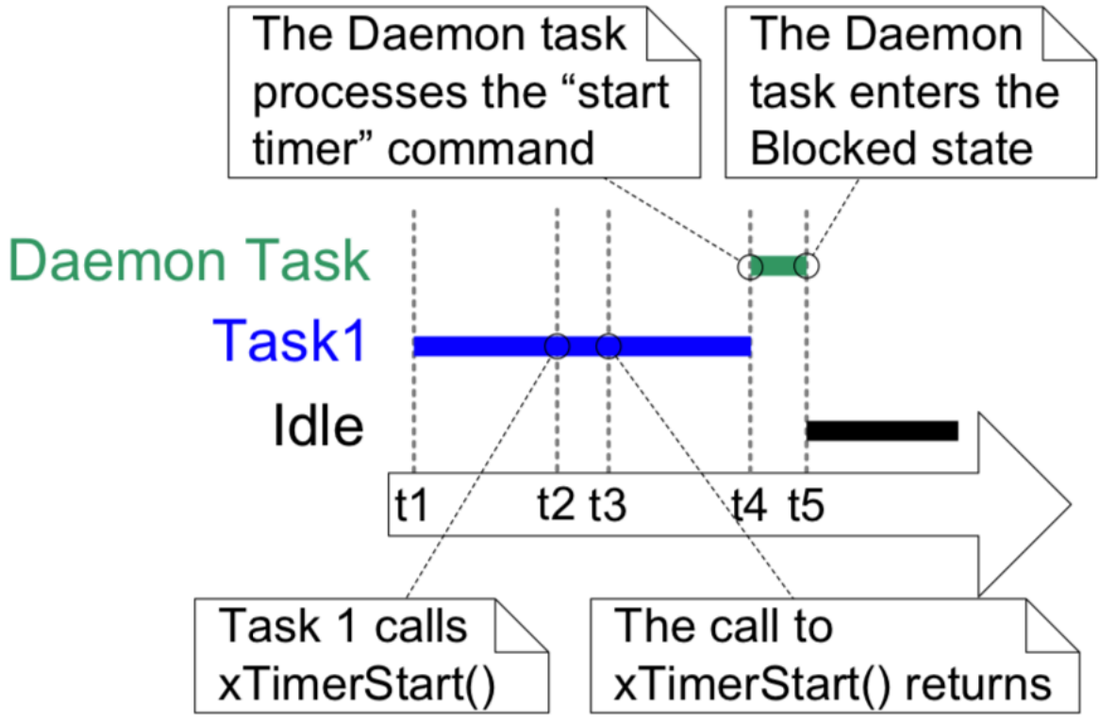
Timer command queue

The RTOS daemon task reads from the timer command queue

Copyright (c) 2019 Servin Corp

Daemon Task Scheduling Example 1 - Task 1 has higher priority than Daemon Task

- Like any other FreeRTOS task, the daemon task only runs when it is the highest priority



Copyright (c) 2019 Servin Corp

5.5 Creating and Starting a Software Timer

- To create a software timer
  - xTimerCreate()
  - xTimerCreateStatic()

Copyright (c) 2019 Servin Corp

xTimerCreate()

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

- Notes
  - Use pdMS_TO_TICKS() to set xTimerPeriodInTicks
  - If uxAutoReload = 0, then one-shot, If uxAutoReload = 1, then auto-reload
  - pvTimerID is a pointer returned and can be used as needed by app developer - useful when same callback is used with more than one timer
  - Returns non-NULL if timer created

Copyright (c) 2019 Servin Corp

xTimerStart()

- BaseType_t
  xTimerStart(
  TimerHandle_t xTimer,
  TickType_t xTicksToWait)

- Notes
  - xTicksToWait is how long to wait if command queue is full - use value of 0 if you want to return immediately if full
  - If INCLUDE_vTaskSuspend is 1, then setting xTicksToWait to portMAX_DELAY will result in calling task staying in blocked state indefinitely until room available in queue
  - Returns pdPASS or pdFALSE
  - Do not call this from an ISR
    - Use xTimerStartFromISR()

Copyright (c) 2019 Servin Corp

Code Demo - Part 1

- #define mainONE_SHOT_TIMER_PERIOD pdMS_TO_TICKS(3333)
- #define mainAUTO_RELOAD_TIMER_PERIOD pdMS_TO_TICKS(500)
- int main(void) {
    TimerHandle_t xOneShotTimer;
    TimerHandle_t xAutoReloadTimer;
    BaseType_t xTimerOneShotStarted;
    BaseType_t xTimerAutoReloadStarted;

    xOneShotTimer = xTimerCreate(“OneShot”, mainONE_SHOT_PERIOD, pdFALSE, 0, prvOneShotTimerCallback);
   xAutoReloadTimer = xTimerCreate(“AutoReload”, MainAUTO_RELOAD_TIMER_PERIOD, pdTRUE, 0, prvAutoReloadTimerCallback);

If ((xOneShotTimer != NULL) && (xAutoReloadTimer != NULL)) {
    xTimerOneShotStarted = xTimerStart(xOneShotTimer, 0);
    xTimerAutoReloadStarted = xTimerStart(xAutoReloadTimer, 0);
    if ((xTimerOneShotStarted) && (xTimerAutoReloadStarted)) {
        vTaskStartScheduler();
    }
}

Copyright (c) 2019 Servin Corp

Code Demo - Part 2

- static void **prvOneShotTimerCallback**(TimerHandle_t xTimer)
  {
    TickType_t xTimeNow;
    xTimeNow = uxTaskGetTickCount();

    vPrintStringAndNumber("One-shot callback\r\n", xTimeNow);
    ulCallCount++;
  }

Copyright (c) 2019 Servin Corp

Code Demo - Part 3

- Static void prvAutoReloadTimerCallback(TimerHandle_t xTimer) {
    TickType_t xTimeNow;
    vPrintStringAndNumber(“Auto-reload timer”, xTimeNow);
    ulCallCount++;
}

Copyright (c) 2019 Servin Corp

5.6 The Timer ID

- Each software timer has an ID
  - ID can be used by app writer for any purpose
    - Stored as a void pointer (void *) so that it can be any value
- Timer ID related functions
  - vTimerSetTimerID()
  - pvTimerGetTimerID()

Copyright (c) 2019 Servin Corp

Code Demo

```
• static void prvTimerCallback(TimerHandle_t *xTimer) {
    TickType_t xTimeNow;
    uint32_t ulExecutionCount;
    ulExecutionCount = (uint32_t) pvTimerGetTimerID(xTimer);
    ulExecutionCount++;
    vTimerSetTimerID(xTimer, (void *)ulExecutionCount);
    xTimeNow = xTaskGetTickCount();

    if (xTimer == xOneShotTimer) {
        vPrintStringAndNumber(“One shot executing”, xTimeNow);
    }
    else {
        vPrintStringAndNumber(“Auto-reload executing”, xTimeNow);
        if (ulExectionCount == 5) {
            xTimerStop(xTimer, 0);
        }
    }
}
```


Copyright (c) 2019 Servin Corp

5.7 Changing the Period of a Timer

- You have the option to change the period of the timer
- Example:
  - Use one period of the timer to flash LED at normal period
  - Use another period of the timer to flash LED at faster period to alert to errors

Copyright (c) 2019 Servin Corp

xTimerChangePeriod()

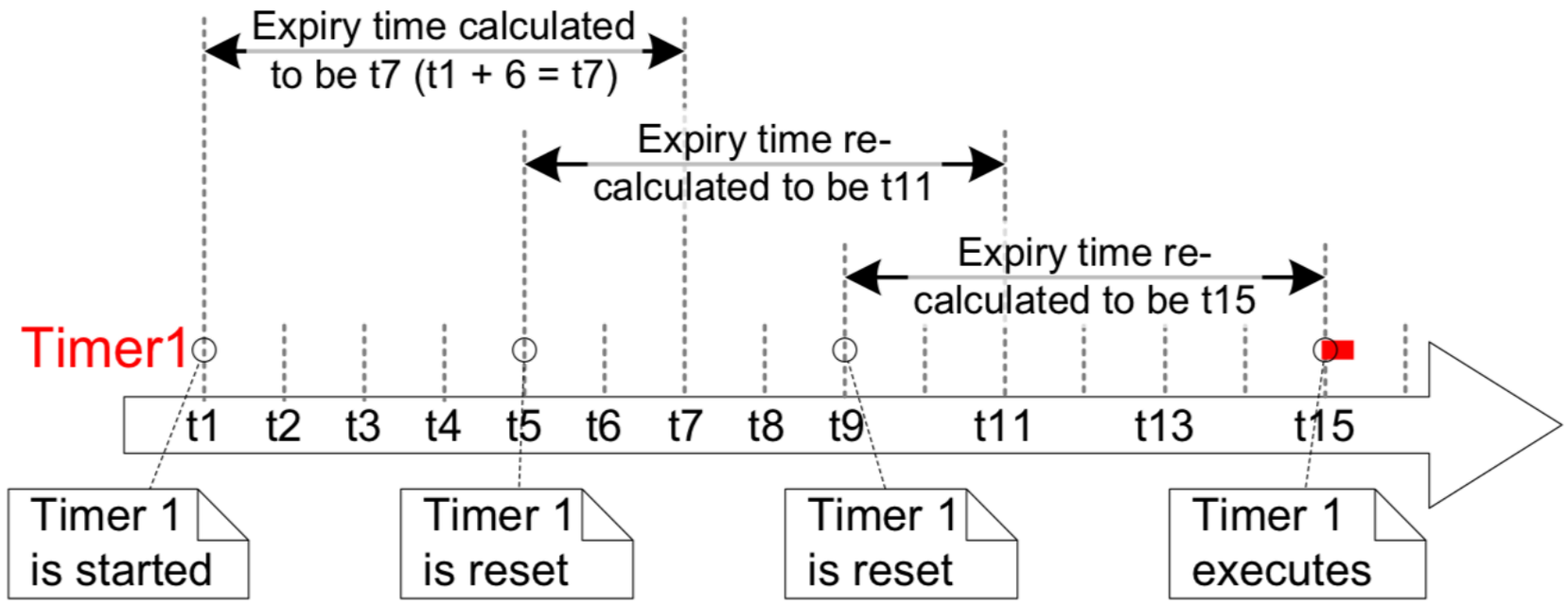
- BaseType_t
  xTimerChangePeriod(
  TimerHandle_t xTimer,
  TickType_t xNewTimerPeriodInTicks,
  TickType_t xTicksToWait)

- Notes
  - Set xNewImterPeriodInTicks with pdMS_TO_TICKS() macro
  - Set xTicksToWait to 0 if you do not want to wait if Timer command queue is full
    - If INCLUDE_vTaskSuspend is set to 1, then portMAX_DELAY will result in task being blocked until Queue has space
  - Returns pdPASS or pdFALSE
  - Never call xTimerChangePeriod() in an ISR - call xTimerChangePeriodFromISR() instead

Copyright (c) 2019 Servin Corp

5.8 Resetting a Software Timer

- Resetting a software timer means to restart the timer



Copyright (c) 2019 Servin Corp

xTimerReset()

- BaseType_t
  xTimerReset(
      TimerHandle_t xTimer,
      TickType_t xTicksToWait)

- Notes

  - Set xTicksToWait to 0 if you do not want to wait if Timer command queue is full

    - If INCLUDE_vTaskSuspend is set to 1, then portMAX_DELAY will result in task being blocked until Queue has space

  - Returns pdPASS or pdFALSE

Copyright (c) 2019 Servin Corp

Summary

- 5.1 Intro and Scope
- 5.2 Software Timer Callback Functions
- 5.3 Attributes and States of a Software Timer
- 5.4 The Context of a Software Timer
- 5.5 Creating and Starting a Software Timer
- 5.6 The Timer ID
- 5.7 Changing the Period of a Timer
- 5.8 Resetting a Software Timer

Copyright (c) 2019 Servin Corp