Introduction

If you have gone through the Event-Driven Programming tutorial, you would probably have either wished for or developed some way to accurately measure the passage of a predetermined amount of time, without resorting to the delay() function. In this section, we will introduce a helper class, CTimeout, to facilitate the timing for us.

CTimeout Class

The CTimeout class is very simple. It consists mainly of a function to check whether "enough time" has elapsed, and a variable to track the "starting time." It would be similar to the following code snippet:

```cpp
class CTimeout
{
public:
    CTimeout(unsigned long timeoutMS);
    bool Expired();

private:
    unsigned long m_startTime;
    unsigned long m_timeout;
};
```

One could then use the class to do something after, say, 5 seconds, by:

```cpp
CTimeout g_Timeout(5000);
```

```cpp
void loop()
{
    if (g_Timeout.Expired())
    {
        // Do something interesting here!
    }

    // Do other repetitive tasks here, which
    // cannot wait for delay() to return.
}
```

Bells and Whistles

While the CTimeout::Expired() function above works, what if we want a repetitive event, such as a blinking LED? In this case, it would be nice if the Expired() method could reset the internal timer state whenever it returned true. Since that might not always be the desired behavior, the caller can indicate which behavior is desired with a flag.

The consumer of the class might also want to manually reset the timer to detect when a normally recurring event stops happening. For example, if we have a button which expects to be pressed once per minute, we could set a CTimeout object to expire in 60,000 ms, and reset it whenever a button is pressed. That way, if the timeout ever does expire, we can know that the button has not been pressed in the last minute.

Modifying the class definition above, we get the following:
class CTimeout
{
public:
    CTimeout(unsigned long timeoutMS);
    bool Expired(bool fAutoReset = false);
    void Reset();
private:
    static unsigned long s_currentTime;
    unsigned long m_startTime;
    unsigned long m_timeout;
};

Modifying the Blink Example
Let’s revisit the blink example from the Digital Output tutorial. Rather than use the delay() function to control the blinking rate, we can use the CTimeout class to control when to change the LED state.

#include "Timeout.h"

#define LED_PIN                     13
#define LED_BLINK_HALF_PERIOD_MS    500

// Global Variables
bool g_fLedOn = true;
CTimeout g_LedTimeout(LED_BLINK_HALF_PERIOD_MS);

void setup()
{
    // Initialize LED pin as an output
    pinMode(LED_PIN, OUTPUT);
}

void loop()
{
    if (g_LedTimeout.Expired(true))
    {
        if (g_fLedOn)
        {
            // Turn LED on
            digitalWrite(LED_PIN, HIGH);
        }
        else
        {
            // Turn LED off
            digitalWrite(LED_PIN, LOW);
        }
    }
// Flip LED state
    g_fLedOn = !g_fLedOn;
}

**Code Files**

CTimeout class files:

- [Timeout.h](#)
- [Timeout.cpp](#)

Modified Blink Example:

- [TimeoutExample.ino](#)