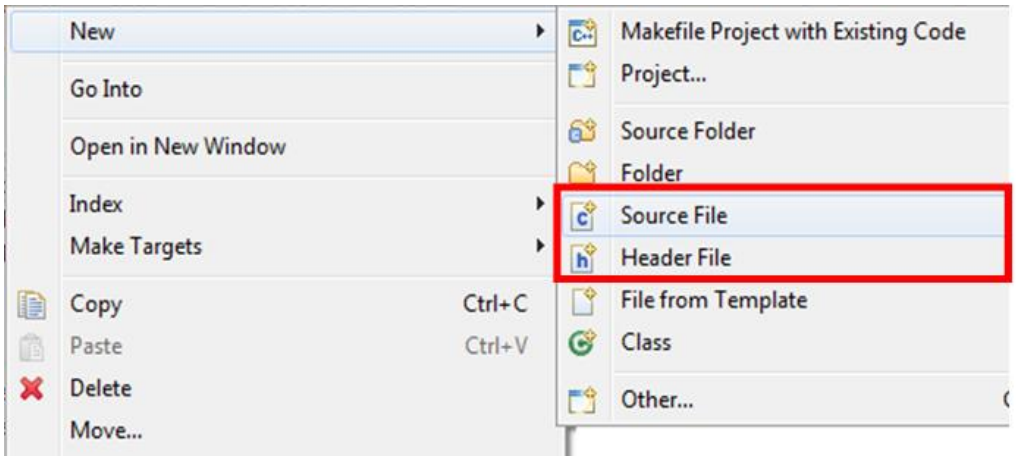
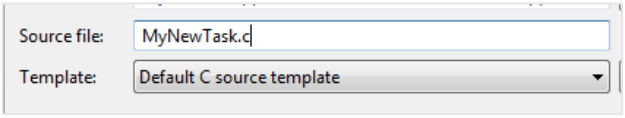
Creating a new Task and Timer

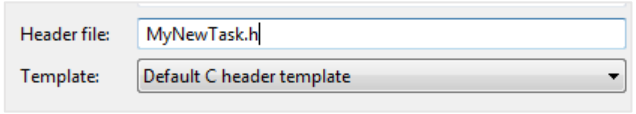
Step by step instructions to create a new Task that will receive an event to toggle the LEDs. The Event will be triggered when a Timer expires. This timer will run indefinitely, so an Interval timer will be used.

"MyWirelessApp Demo - Non Beacon (Coordinator)" will be used as the base of this example lab.

1. **Import a MyWirelessApp Demo Coordinator (mwa_coordinator_freertos)**
2. **Create new 'MyNewTask.c' and 'MyNewTask.h' files in "Application → Source"**







3. **Open MyNewTask.c**

a. **Include "MyNewTask.h"**

b. **Declare required variables and prototypes**

```
osaEventId_t          mMyEvents;
/* Global Variable to store our TimerID */
tmrTimerID_t myTimerID = gTmrInvalidTimerID_c;

/* Handler ID for task */
osaTaskId_t gMyTaskHandler_ID;

/* Local variable to store the current state of the LEDs */
static uint8_t ledsState = 0;
```

c. **Define OSA Task**

```
/* OSA Task Definition*/
OSA_TASK_DEFINE(My_Task, gMyTaskPriority_c, 1, gMyTaskStackSize_c, FALSE );
```

d. Define the new Task function that will handle the events and trigger an Interval Timer.

```
/* Main custom task */
void My_Task(osaTaskParam_t argument)
{
    osaEventFlags_t customEvent;
    myTimerID = TMR_AllocateTimer();

    while(1)
    {
          OSA_EventWait(mMyEvents, osaEventFlagsAll_c, FALSE, osaWaitForever_c,
&customEvent);

          if( !gUseRtos_c && !customEvent)
          {
                break;
          }

          /* Depending on the received event */
          switch(customEvent){
          case gMyNewTaskEvent1_c:
                TMR_StartIntervalTimer(myTimerID, /*myTimerID*/
                            1000, /* Timer's Timeout */
                            myTaskTimerCallback, /* pointer to
myTaskTimerCallback function */
                            NULL
                );
                TurnOffLeds(); /* Ensure all LEDs are turned off */
                break;
          case gMyNewTaskEvent2_c: /* Event called from myTaskTimerCallback */
                if(!ledsState) {
                      TurnOnLeds();
                      ledsState = 1;
                }
                else {
                      TurnOffLeds();
                      ledsState = 0;
                }
                break;
          case gMyNewTaskEvent3_c: /* Event to stop the timer */
                ledsState = 0;
                TurnOffLeds();
                TMR_StopTimer(myTimerID);
                break;
          default:
                break;
          }
    }
}
```

e. Define the function that will be called to send an Event to your new Task.

```
/* Function to init the task */
void MyTask_Init(void)
{
```

```
    mMyEvents = OSA_EventCreate(TRUE);
    /* The instance of the MAC is passed at task creaton */
    gMyTaskHandler_ID = OSA_TaskCreate(OSA_TASK(My_Task), NULL);
}
```

f. Define the callback that will be called each time the timer expires

```
/* This is the function called by the Timer each time it expires */
static void myTaskTimerCallback(void *param)
{
    OSA_EventSet(mMyEvents, gMyNewTaskEvent2_c);
}
```

g. Define a function to send an event to stop the timer

```
/* Public function to send an event to stop the timer */
void MyTaskTimer_Stop(void)
{
    OSA_EventSet(mMyEvents, gMyNewTaskEvent3_c);
}
```

4. Open MyNewTask.h

a. Include the required dependencies

```
/* Fwk */
#include "TimersManager.h"
#include "FunctionLib.h"
#include "LED.h"
/* KSDK */
#include "fsl_common.h"
#include "EmbeddedTypes.h"
#include "fsl_os_abstraction.h"
```

b. Define the EVENTs that the new task will support.

```
/* Define the available Task's Events */
#define gMyNewTaskEvent1_c (1 << 0)
#define gMyNewTaskEvent2_c (1 << 1)
#define gMyNewTaskEvent3_c (1 << 2)
```

c. Define a TASK PRIORITY and TASK SIZE for your new task

```
#define gMyTaskPriority_c 3
#define gMyTaskStackSize_c 400
```

d. Prototype definition

```
void MyTaskTimer_Start(void);
void MyTaskTimer_Stop(void);
void MyTask_Init(void);
```

5. **Open mwa_coordinator.c**

a. **Include your new header file**

```
#include "MyNewTask.h"
```

6. **Initialize your task in main_task()**

```
MyTask_Init();    /* INIT MY NEW TASK */
```

7. **Call function to initialize the timer in your task. Do this under "stateStartCoordinatorWaitConfirm", after the coordinator initialized the network.**

```
MyTaskTimer_Start(); /*Start LED flashing with your task*/
```

8. **Call a function to stop the timer from your task (after Network Creation). Call this when the End device joined the network.**

```
MyTaskTimer_Stop();   /* STOP Timer from MY NEW TASK*/
```