```c
/****************************************************************************
 Module
   TemplateFSM.c

 Revision
   1.0.1

 Description
   This is a template file for implementing flat state machines under the
   Gen2 Events and Services Framework.

 Notes

 History
 When           Who     What/Why
 -------------- ---     --------
 01/15/12 11:12 jec      revisions for Gen2 framework
 11/07/11 11:26 jec      made the queue static
 10/30/11 17:59 jec      fixed references to CurrentEvent in RunTemplateSM()
 10/23/11 18:20 jec      began conversion from SMTemplate.c (02/20/07 rev)
****************************************************************************/
/*----------------------------- Include Files -----------------------------*/
/* include header files for this state machine as well as any machines at the
   next lower level in the hierarchy that are sub-machines to this machine
*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "LEDBlinkSM.h"
#include "FarmerMasterSM.h"
#include "Constants.h"

#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_nvic.h"
#include "inc/hw_uart.h"
#include "inc/hw_sysctl.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"      // Define PART_TM4C123GH6PM in project
#include "driverlib/gpio.h"
#include "driverlib/uart.h"

/*----------------------------- Module Defines ----------------------------*/

/*---------------------------- Module Functions ---------------------------*/
/* prototypes for private functions for this machine.They should be functions
   relevant to the behavior of this state machine
*/

/*---------------------------- Module Variables ---------------------------*/
// everybody needs a state variable, you may need others as well.
// type of state variable should match htat of enum in header file
static LEDBlinkState_t CurrentState;
static uint8_t LED_num;

// with the introduction of Gen2, we need a module level Priority var as well
static uint8_t MyPriority;


/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitLEDBlinkSM
 Parameters
     uint8_t : the priorty of this service

 Returns
     bool, false if error in initialization, true otherwise

 Description
     Saves away the priority, sets up the initial transition and does any
     other required initialization for this state machine
 Notes

 Author
     J. Edward Carryer, 10/23/11, 18:55
****************************************************************************/
bool InitLEDBlinkSM ( uint8_t Priority )
{
  ES_Event ThisEvent;

  MyPriority = Priority;
  // put us into the Initial PseudoState
  CurrentState = InitPS;
  // post the initial transition event
  ThisEvent.EventType = ES_INIT;
  if (ES_PostToService( MyPriority, ThisEvent) == true)
  {
      return true;
  }else
  {
      return false;
  }
}

/****************************************************************************
 Function
     PostLEDBlinkSM

 Parameters
     EF_Event ThisEvent , the event to post to the queue

 Returns
     boolean False if the Enqueue operation failed, True otherwise

 Description
     Posts an event to this state machine's queue
 Notes

 Author
     J. Edward Carryer, 10/23/11, 19:25
****************************************************************************/
bool PostLEDBlinkSM(ES_Event ThisEvent)
{
  return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunLEDBlinkSM

 Parameters
   ES_Event : the event to process

 Returns
   ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise
 Description
   add your description here
 Notes
   uses nested switch/case to implement the machine.
 Author
   J. Edward Carryer, 01/15/12, 15:23
****************************************************************************/
ES_Event RunLEDBlinkSM( ES_Event ThisEvent )
{
  ES_Event ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

      LEDBlinkState_t NextState;
      NextState = CurrentState;

  switch (CurrentState)
  {
            case InitPS:
                  if(ThisEvent.EventType == ES_INIT)
                  {
                        NextState = BlinkOn;
                        LED_num = getDogSelect();
                        printf("Initializing LED associated with with %i\r\n",
LED_num);

                        if(LED_num == 0)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(Y_LED_1_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_2_B | G_LED_2_B | Y_LED_3_B | G_LED_3_B);
                        }

                        else if(LED_num == 1)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(Y_LED_2_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_1_B | G_LED_2_B | Y_LED_3_B | G_LED_3_B);
                        }

                        else if(LED_num == 2)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(Y_LED_3_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_1_B | G_LED_2_B | Y_LED_2_B | G_LED_3_B);
                        }

                        ES_Timer_InitTimer(LED_TIMER, BLINK_TIME);
                  }

                  break;

            case BlinkOn:

                  if(ThisEvent.EventType == ES_INCREMENT_LED)
                  {
                        LED_num = getDogSelect();

                        printf("Turning on LED associated with with %i\r\n",
LED_num);

                        if(LED_num == 0)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(Y_LED_1_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_2_B | G_LED_2_B | Y_LED_3_B | G_LED_3_B);
                        }

                        else if(LED_num == 1)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(Y_LED_2_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_1_B | G_LED_2_B | Y_LED_3_B | G_LED_3_B);
                        }

                        else if(LED_num == 2)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(Y_LED_3_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_1_B | G_LED_2_B | Y_LED_2_B | G_LED_3_B);
                        }
                  }

                  if(ThisEvent.EventType == ES_TIMEOUT)
                  {
                        NextState = BlinkOff;
                        HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_1_B | G_LED_2_B | Y_LED_2_B | G_LED_3_B | Y_LED_3_B);
                        ES_Timer_InitTimer(LED_TIMER, BLINK_TIME);
                  }

                  if(ThisEvent.EventType == ES_PAIR_SUCCESSFUL)
                  {
                        NextState = PairedSolid;

                        if(LED_num == 0)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(G_LED_1_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(Y_LED_1_B | Y_LED_2_B | G_LED_2_B | Y_LED_3_B | G_LED_3_B);
                        }

                        else if(LED_num == 1)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(G_LED_2_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_1_B | Y_LED_2_B | Y_LED_3_B | G_LED_3_B);
                        }

                        else if(LED_num == 2)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(G_LED_3_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_1_B | G_LED_2_B | Y_LED_2_B | Y_LED_3_B);
                        }
                  }

                  break;

            case BlinkOff:
                  if(ThisEvent.EventType == ES_INCREMENT_LED)
                  {
                        LED_num = getDogSelect();
                  }

                  if(ThisEvent.EventType == ES_TIMEOUT)
                  {
                        //printf("Turning on LED associated with with %i\r\n",
LED_num);

                        NextState = BlinkOn;
                        if(LED_num == 0)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(Y_LED_1_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_2_B | G_LED_2_B | Y_LED_3_B | G_LED_3_B);
                        }

                        else if(LED_num == 1)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(Y_LED_2_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_1_B | G_LED_2_B | Y_LED_3_B | G_LED_3_B);
                        }

                        else if(LED_num == 2)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(Y_LED_3_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_1_B | G_LED_2_B | Y_LED_2_B | G_LED_3_B);
                        }
                        ES_Timer_InitTimer(LED_TIMER, BLINK_TIME);

                  }

                  if(ThisEvent.EventType == ES_PAIR_SUCCESSFUL)
                  {
                        NextState = PairedSolid;

                        if(LED_num == 0)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(G_LED_1_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(Y_LED_1_B | Y_LED_2_B | G_LED_2_B | Y_LED_3_B | G_LED_3_B);
                        }

                        else if(LED_num == 1)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(G_LED_2_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_1_B | Y_LED_2_B | Y_LED_3_B | G_LED_3_B);
                        }

                        else if(LED_num == 2)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(G_LED_3_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_1_B | G_LED_2_B | Y_LED_2_B | Y_LED_3_B);
```

```
                        }
                  }

                  break;

            case PairedSolid:

                  if(ThisEvent.EventType == ES_LOST_CONNECTION)
                  {
                        NextState = BlinkOn;
                        if(LED_num == 0)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(Y_LED_1_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_2_B | G_LED_2_B | Y_LED_3_B | G_LED_3_B);
                        }

                        else if(LED_num == 1)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(Y_LED_2_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_1_B | G_LED_2_B | Y_LED_3_B | G_LED_3_B);
                        }

                        else if(LED_num == 2)
                        {
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) |=
(Y_LED_3_B);
                              HWREG(GPIO_PORTB_BASE + (ALL_BITS + GPIO_O_DATA)) &=
~(G_LED_1_B | Y_LED_1_B | G_LED_2_B | Y_LED_2_B | G_LED_3_B);
                        }
                        ES_Timer_InitTimer(LED_TIMER, BLINK_TIME);

                  }

                  break;
  } // end switch on Current State

      CurrentState = NextState;
  return ReturnEvent;
}


/***************************************************************************
 private functions
 ***************************************************************************/
```