```
/*******************************************************************************
 Module
   TemplateService.c

 Revision
   1.0.1

 Description
   This is a template file for implementing a simple service under the
   Gen2 Events and Services Framework.

 Notes

 History
 When           Who     What/Why
 -------------- ---     --------
 01/16/12 09:58 jec      began conversion from TemplateFSM.c
*******************************************************************************/
/*----------------------------- 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 <stdint.h>
#include <stdbool.h>

#include "inc/hw_gpio.h"
#include "inc/hw_types.h"
#include "inc/hw_pwm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"

#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"

#include "BITDEFS.H"

#include "ES_Configure.h"
#include "ES_Framework.h"

#include "GameService.h"
#include "WaterService.h"
#include "LightService.h"
#include "CelebService.h"
#include "HarvestButtonService.h"
#include "LimitSwitchService.h"
#include "NeedleService.h"
#include "LEDWriteService.h"
#include "PWM16Tiva.h"
#include "ADMulti.h"

#define DATA GPIO_PIN_0

#define SCLK GPIO_PIN_1
#define SCLK_HI BIT1HI
#define SCLK_LO BIT1LO

#define RCLK GPIO_PIN_2
#define RCLK_LO BIT2LO
#define RCLK_HI BIT2HI

#define HARVEST_TIME 3000
```

```
#define WELCOME_TIME 1000
#define BTN_WAIT_TIME 10000
#define PULSE_WIDTH 1250
#define PULSE_WIDTH_DISP 2125

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

/*---------------------------- Module Functions ----------------------------*/
/* prototypes for private functions for this service.They should be functions
   relevant to the behavior of this service
*/
static void HardwareInitialize(void);
static void HarvestAndSwitchInit(void);
static void VibeMotorInit(void);
static void HarvestButtonLEDOn(void);
static void HarvestButtonLEDOff(void);
static void DisplayMotorOn(uint8_t channel);

/*---------------------------- Module Variables ----------------------------*/
// with the introduction of Gen2, we need a module level Priority variable
static uint16_t motorPeriod = 25000; // 20 ms
static uint8_t group = 0; // Channels 0 and 1
static uint8_t MyPriority;
static uint8_t HealthBar = 0;
static Game_States_t CurrentState = Inactivity;

/*---------------------------- Module Code ----------------------------*/
/****************************************************************************
 Function
     InitGameService

 Parameters
     uint8_t : the priorty of this service

 Returns
     bool, false if error in initialization, true otherwise

 Description
     Saves away the priority, and does any
     other required initialization for this service
 Notes

 Author
     J. Edward Carryer, 01/16/12, 10:00
****************************************************************************/
bool InitGameService(uint8_t Priority)
{
  ES_Event_t ThisEvent;
  MyPriority = Priority;

  // post the initial transition event
  ThisEvent.EventType = ES_INIT;

      // Initialize the Harvest Button, Harvest LED and Limit Switch
      printf("Init\r\n");
      //HarvestAndSwitchInit();

  // Initialize all the hardware
      //HardwareInitialize();

  // Initialize Port D
  HWREG(SYSCTL_RCGCGPIO) |= BIT3HI;

  // Wait for clock to get ready
```

```
  while ((HWREG(SYSCTL_PRGPIO) & BIT3HI) != BIT3HI)
  {
  }

  // Set data direction and assign digital port on Port D
  HWREG(GPIO_PORTD_BASE+GPIO_O_DEN) |= BIT0HI;
  HWREG(GPIO_PORTD_BASE+GPIO_O_DIR) |= BIT0HI;

  // potato display motors period
  PWM_TIVA_SetPeriod(motorPeriod, group);
  // Initialize display motors on ground
  PWM_TIVA_SetPulseWidth(PULSE_WIDTH, 0);
  PWM_TIVA_SetPulseWidth(PULSE_WIDTH, 1);

  if (ES_PostToService(MyPriority, ThisEvent) == true)
  {
      return true;
  }else
  {
      return false;
  }
}

/****************************************************************************
 Function
     PostGameService

 Parameters
     EF_Event ThisEvent ,the event to post to the queue

 Returns
     bool 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 PostGameService(ES_Event_t ThisEvent)
{
  return ES_PostToService(MyPriority, ThisEvent);
}




bool Check4GameStart(void)
{

bool ReturnVal = false;
static bool LastSensorState = false;
bool CurrentSensorState;
ES_Event_t Event2Post;

// Set CurrentSwitchState to state read from port pin
uint8_t ReadPortA = HWREG(GPIO_PORTA_BASE+(GPIO_O_DATA + ALL_BITS));
CurrentSensorState = !(ReadPortA & BIT4HI);

if (LastSensorState != CurrentSensorState) {
  // Check start of PA4
  if (CurrentSensorState) {
    Event2Post.EventType = GAME_START;
```

```
    PostGameService(Event2Post);
    //printf("TOT detected\r\n");
  }
}

LastSensorState = CurrentSensorState;
return ReturnVal;
}

/****************************************************************************
 Function
    RunGameService

 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

 Author
   J. Edward Carryer, 01/15/12, 15:23
****************************************************************************/
ES_Event_t RunGameService(ES_Event_t ThisEvent)
{
  ES_Event_t ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
  Game States t NextState = CurrentState;
  static int count = 0;

  switch(CurrentState) {

    case Inactivity:
                printf("Inactivity\r\n");

      if (ThisEvent.EventType == GAME_START) {
        HarvestButtonLEDOff();

                        // Initialize Health LEDs to all off
                        ES_Event_t LEDEvent;
                        LEDEvent.EventType = HEALTH_CMD;
                        LEDEvent.EventParam = 0;
                        printf("Health: %u\r\n", LEDEvent.EventParam);
        HealthBar = 0;
                        PostLEDWriteService(LEDEvent);

                        // Turn off water LEDs
                        LEDEvent.EventType = WATER_CMD;
                        LEDEvent.EventParam = 0x00;
                        printf("Water: %u\r\n", LEDEvent.EventParam);
                        PostLEDWriteService(LEDEvent);

                        // Turn off weather LEDs
                        LEDEvent.EventType = LIGHT_CMD;
                        LEDEvent.EventParam = 0x00;
                        printf("Light: %u\r\n", LEDEvent.EventParam);
                        PostLEDWriteService(LEDEvent);

                        // Post to needle service so it moves to leftmost
position
        ES_Event_t Event2Post;
```

```
        Event2Post.EventType = NEEDLE_START;
                        PostNeedleService(Event2Post);

                        // Move to Watering State
                        NextState = Watering;
        ES_Event_t Event2Post2;
        Event2Post2.EventType = WATER_START;
        printf("Water start posted\r\n");
        PostWaterService(Event2Post2);
                  }

                  // If game is not started, stay in welcome mode
                  // LEDs will flash at time intervals of WELCOME_TIMER


                  else if (ThisEvent.EventType == ES_TIMEOUT &&
ThisEvent.EventParam == WELCOME_TIMER) {

                        printf("Welcome timer timeout\r\n");
                        // Start Welcome Timer
                        uint16_t welcomeTime = 500; // ms
                        ES_Timer_InitTimer(WELCOME_TIMER, WELCOME_TIME);
                        printf("Welcome Timer started\r\n");

        // Every other time flash LEDs on
        if (count%2 == 0) {
          ES_Event_t Event2Post;
          Event2Post.EventType = ALL_ON;
          PostLEDWriteService(Event2Post);
        }

        else {
          ES_Event_t Event2Post;
          Event2Post.EventType = ALL_OFF;
          PostLEDWriteService(Event2Post);
        }

                        // Self-transition so stay in current state
                        NextState = Inactivity;
        count++;
                  }

      else {
        // If nothing else has happened, start the timer
        ES_Timer_InitTimer(WELCOME_TIMER, WELCOME_TIME);

        NextState = Inactivity;
        count = 1;
      }
      break;

    case Watering:
                  printf("Watering\r\n");

                  // If the water timer times out
                  if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam
== WATER_TIMER) {

                        // Post to LightService to start
                        ES_Event_t LightEvent;
                        LightEvent.EventType = LIGHT_START;
                        PostLightService(LightEvent);
                        printf("Posted light\r\n");
```

```
                        // Move to Lighting State
                        NextState = Lighting;
                  }
      break;

    case Lighting:
                  printf("Lighting\r\n");

                  // If the light timer times out
                  if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam
== LIGHT_TIMER) {

                        // Turn on Harvest Button LED
                        HarvestButtonLEDOn();

                        // Start the ButtonWait Timer
                        ES_Timer_InitTimer(BTN_WAIT_TIMER, BTN_WAIT_TIME);
                        printf("Btn Wait Timer Started\r\n");

                        // Move to Harvesting State
                        NextState = Harvesting;
                  }
      break;

    case Harvesting:
                  printf("Harvesting\r\n");

                  // If the Button Wait Timer times out
                  if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam
== BTN_WAIT_TIMER) {

                        printf("Btn Wait Timer Ended\r\n");

                        // Init Harvest Timer
                        ES_Timer_InitTimer(HARVEST_TIMER, HARVEST_TIME);
                        printf("Harvest timer started\r\n");

        // Decrease Health -5
                        for (int i = 0; i < 5; i++) {
                              SubtractHealth();
                        }
                        printf("HealthBar: %d", GetHealth());

                        // Switch off the Harvest LED
                        HarvestButtonLEDOff();

        // Bad potato display if health too low
        if (GetHealth() < 6) {

          // Bad Potato Motor at PB7 (Channel 1)
          uint8_t potato_channel = 1;
          //printf("HealthBar: %d\r\n", GetHealth());
          DisplayMotorOn(potato_channel);
          printf("Motor BAD DISPLAY: %d!!!!!!!!!\r\n", potato_channel);
        }

        // Good potato display if health good
        else {

          // Good Potato Motor at PB6 (Channel 0)
          uint8_t potato_channel = 0;
          //printf("HealthBar: %d\r\n", GetHealth());
          DisplayMotorOn(potato_channel);
          printf("Motor GOOD DISPLAY: %d!!!!!!\r\n", potato_channel);
```

```
        }

                        // Move to PotatoDisplay State
                        NextState = PotatoDisplay;

                }

                // If the Harvest Button is pressed
      if (ThisEvent.EventType == BUTTON_PRESSED) {

        // Add 1 to Health Bar
                        AddHealth();
                        printf("HealthBar: %d\r\n", GetHealth());

                        // Init Harvest Timer
                        ES_Timer_InitTimer(HARVEST_TIMER, HARVEST_TIME);
                        printf("Harvest timer started\r\n");

                        // Switch off Harvest LED
                        HarvestButtonLEDOff();

        // Bad potato display if health too low
        if (GetHealth() < 6) {

          // Bad Potato Motor at PB7 (Channel 1)
          uint8_t potato_channel = 1;
          //printf("HealthBar: %d\r\n", GetHealth());
          DisplayMotorOn(potato_channel);
          printf("Motor BAD DISPLAY: %d!!!!!!!!!\r\n", potato_channel);
        }

        // Good potato display if health good
        else {

          // Good Potato Motor at PB6 (Channel 0)
          uint8_t potato_channel = 0;
          //printf("HealthBar: %d\r\n", GetHealth());
          DisplayMotorOn(potato_channel);
          printf("Motor GOOD DISPLAY: %d!!!!!!\r\n", potato_channel);
        }

                        // Move to PotatoDisplay State
                        NextState = PotatoDisplay;
                }
                break;

    case PotatoDisplay:
                printf("PotatoDisplay\r\n");

                // If the harvest timer times out
                if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam
== HARVEST_TIMER) {
                        printf("Harvest timer timed out\r\n");

                        // Post to CelebService
                        ES_Event_t CelebEvent;
                        CelebEvent.EventType = CELEB_START;
                        PostCelebService(CelebEvent);

                        // Move to Celebrating State
                        NextState = Celebrating;
                }

      break;
```

```
    case Celebrating:
                  printf("Celebrating\r\n");

                  // If the celebration timer times out
                  if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam
== CELEB_TIMER) {
                        printf("Celebration timeout\r\n");
                        // Post to Needle Service so it moves to rightmost
position
        ES_Event_t Event2Post;
        Event2Post.EventType = ES_TIMEOUT;
        Event2Post.EventParam = CELEB_TIMER;
                        PostNeedleService(Event2Post);
        printf("Posted Celeb Timeout to NEEDLE\r\n");

                        // Move to Inactivity State
                        NextState = Celebrating;

        // Displace TOT
        // write high on PD0 = Turn solenoid on
        HWREG(GPIO_PORTD_BASE+(GPIO_O_DATA + ALL_BITS)) |= BIT0HI;

        ES_Timer_InitTimer(WELCOME_TIMER, WELCOME_TIME);
        printf("Welcome timer started\r\n");
                  }

      else if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam ==
WELCOME_TIMER) {
        // write low on PD0 = Turn solenoid off
        HWREG(GPIO PORTD BASE+(GPIO O DATA + ALL BITS)) = 0;
        printf("Welcome time out\r\n");
        ES_Timer_InitTimer(WELCOME_TIMER, WELCOME_TIME);

        NextState = Inactivity;
      }
      break;
  }

  // Set CurrentState to NextState
  CurrentState = NextState;

  // Return ES_NO_Event+



  return ReturnEvent;
}

uint8_t GetHealth(void)
{
  return HealthBar;
}

void AddHealth(void)
{
  static uint8_t counter = 0;
  counter++;

  if (counter%2 == 0) {
    HealthBar++;
  }

  uint8_t AllOnes = 0xFF;
```

```
  if (HealthBar > 8) {
    HealthBar = 8;
  }

  ES_Event_t Event2Post;
  Event2Post.EventType = HEALTH_CMD;
  Event2Post.EventParam = AllOnes >> (8 - HealthBar);
  printf("Add Health: %d\r\n", HealthBar);
  PostLEDWriteService(Event2Post);
}

void SubtractHealth(void)
{
  uint8_t AllOnes = 0xFF;

  if (HealthBar > 0) {
    HealthBar -= 1;
  }

  ES_Event_t Event2Post;
  Event2Post.EventType = HEALTH_CMD;
  Event2Post.EventParam = AllOnes >> (8 - HealthBar);
  printf("Healthbar: %d\r\n", HealthBar);
  printf("Sub Health: %d\r\n", HealthBar);
  PostLEDWriteService(Event2Post);
}

/*****************************************************************************
 private functions
 *****************************************************************************/
static void HardwareInitialize(void)
{
  static uint8_t PWM_NumChannels = 5;
  static uint8_t ADC_NumPins = 4;

  // Initialize Vibration Motor
  //VibeMotorInit();
  printf("ADC and PWM Init done\r\n");
}

static void HarvestAndSwitchInit(void)
{
      printf("Harvest and Switch on\r\n");
}

static void VibeMotorInit(void)
{
      // Set data direction and assign digital port on Port B
      HWREG(GPIO_PORTB_BASE+GPIO_O_DEN) |= BIT3HI;
      HWREG(GPIO_PORTB_BASE+GPIO_O_DIR) |= BIT3HI;
      printf("Vibration motor init\r\n");
}

static void HarvestButtonLEDOn(void)
{
      // Turn on HarvestButton LED
      HWREG(GPIO_PORTA_BASE+(GPIO_O_DATA + ALL_BITS)) |= BIT3HI;
      printf("Button LED on\r\n");
}

static void HarvestButtonLEDOff(void)
{
      // Turn off HarvestButton LED
```

```
      HWREG(GPIO_PORTA_BASE+(GPIO_O_DATA + ALL_BITS)) &= BIT3LO;
      printf("Button LED off\r\n");
}

static void DisplayMotorOn(uint8_t potato_channel)
{
//    uint16_t motorPeriod = 25000; // 20 ms
//    uint8_t group = 0; // Channels 0 and 1

      // Set the period and pulse width for potato display
      printf("DISPLAY MOTOR!!!!\r\n");
      PWM_TIVA_SetPulseWidth(PULSE_WIDTH_DISP, potato_channel);
}

/*------------------------------ Footnotes -------------------------------*/
/*------------------------------ End of file -------------------------------*/
```