```c
/****************************************************************************
 Module
   Input_Service.c

 Description
   This service is responsible for retrieving the current values of the various
   sensors and buttons on the controller

****************************************************************************/
/*----------------------------- 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 "Outputs_Service.h"
#include "Definitions.h"
#include "Helpers.h"
#include "ADMulti.h"

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

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

void pulseTimerClock(void);

/*---------------------------- Module Variables ---------------------------*/
// with the introduction of Gen2, we need a module level Priority variable
static uint8_t MyPriority;
static bool special_action = false;
static bool pairedFlag = false;

/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitOutputs_Service

 Parameters
     uint8_t : the priorty of this service

 Returns
     bool, false if error in initialization, true otherwise

 Description
     Initializes all Ports and Pins for inputs to Controller
****************************************************************************/
bool InitOutputs_Service ( uint8_t Priority )
{
    ES_Event ThisEvent;

    MyPriority = Priority;

    //just for the first checkpoint
    HWREG(GPIO_PORTC_BASE + GPIO_O_DEN) |= (BIT6HI | BIT7HI);
    HWREG(GPIO_PORTC_BASE + GPIO_O_DIR) |= (BIT6HI | BIT7HI);

    //Initialize Orientation Servos
    InitPWM(LEFT_SERVO_PWM_PARAMATERS);
    InitPWM(RIGHT_SERVO_PWM_PARAMATERS);
    //Set Orientation Servos to Initial Values
    //initialize AD Converter for Badge (PE0)
    ADC_MultiInit(2);


    //Initialize Lift Fan to be off
    GPIO_Init(LIFT_SYSCTL, LIFT_GPIO, LIFT_FAN_PIN, OUTPUT);
    GPIO_Clear(LIFT_GPIO, LIFT_FAN_PIN);

    //Initialize Thrust Fan
    InitPWM(THRUST_LEFT_PWM_PARAMATERS);
    InitPWM(THRUST_RIGHT_PWM_PARAMATERS);

    //Initialize Thrust Fans to Be Zero
    setPWM_value(0, THRUST_LEFT_PWM_PARAMATERS);
    setPWM_value(0, THRUST_RIGHT_PWM_PARAMATERS);

    //Set Direction to Forwards
    GPIO_Init(THRUST_SYSCTL, THRUST_GPIO, THRUST_LEFT_DIRECTION_PIN, OUTPUT);
    GPIO_Init(THRUST_SYSCTL, THRUST_GPIO, THRUST_RIGHT_DIRECTION_PIN, OUTPUT);
    GPIO_Clear(THRUST_GPIO, THRUST_LEFT_DIRECTION_PIN);
    GPIO_Clear(THRUST_GPIO, THRUST_RIGHT_DIRECTION_PIN);

    //Initialize Pin Outputs to DMC
    GPIO_Init(DMC_SYSTCTL, DMC_BASE, COLOR_PIN | PAIRED_PIN | TIMING_PIN, OUTPUT)
              ;

    //Initlaize as Unpaired
    setCurrentPairingStatus(!PAIRED_BIT);

    //Initialize Sound Pin
    GPIO_Init(SOUND_SYSTCTL, SOUND_BASE, SOUND_PIN, OUTPUT);
    GPIO_Set(SOUND_BASE, SOUND_PIN);    //set it to start off so that we do not
             play

    //Start the Timing Timer
    //ES_Timer_InitTimer(DMC_TIMING_TIMER, DMC_TIMING_TIMER_TIME);

    // post the initial transition event
    ThisEvent.EventType = ES_INIT;
    if (ES_PostToService( MyPriority, ThisEvent) == true)
    {
        return true;
    }
    else
    {
        return false;
    }
}

/****************************************************************************
 Function
     PostOutputs_Service

****************************************************************************/
bool PostOutputs_Service( ES_Event ThisEvent )
{
    return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
```

```
 Function
    RunOutputs_Service

 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 RunOutputs_Service( ES_Event ThisEvent )
{
    ES_Event ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

    //If we received a timeout from the timing timer
    if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam ==
        DMC_TIMING_TIMER)
    {
        //if we are paired
        if (pairedFlag)
        {
            //Pulse the Clock
            pulseTimerClock();

            //Reset the timer
            ES_Timer_InitTimer(DMC_TIMING_TIMER, DMC_TIMING_TIMER_TIME);
        }
    }
    //if the special action timer times out, turn off the special action
    if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam ==
        SPECIAL_ACTION_TIMER)
    {
        special_action = false;
        ES_Timer_StopTimer(SPECIAL_ACTION_TIMER);
    }

    return ReturnEvent;
}

/***************************************************************************
 private functions
 ***************************************************************************/
//Function Will Retrieve the Analog value from the team number pin, and convert
//that value to the team number and return it
uint8_t getCurrentTeamNumber(void)
{
    //retrieve the analog value
    uint32_t data[4] = {0,0,0,0};
    ADC_MultiRead(data);
    uint32_t value = data[1];
    uint8_t my_num = 5;
    //cascading if statements for team number
    if (value > 2550)
    {
        my_num = 3;
    }
    else if (value > 2200)
    {
        my_num = 2;
    }
    else if (value > 1800)
    {
        my_num = 1;
    }
    else if (value > 1250 && value < 1500)
    {
        my_num = 0;
    }

    return my_num;
}

//Command the Current Thrust by passing in the integer value
void commandCurrentThrust(int8_t thrust)
{

    //Initialize Thrust Values
    float thrustLeft_f;
    float thrustRight_f;

    //If thrust is greater than 0 then we want to move forward
    if (thrust >= 0)
    {
        //Convert the thrust to duty cycle
        thrustLeft_f = 100.0*(float)thrust / 128.0 + LEFT_THRUST_OFFSET;
        thrustRight_f = 100.0*(float)thrust / 128.0 + RIGHT_THRUST_OFFSET;

        //clamp at Maximum PWM
        if (thrustLeft_f > MAX_PWM_DC)
        {
            thrustLeft_f = MAX_PWM_DC;
        }
        if (thrustRight_f > MAX_PWM_DC)
        {
            thrustRight_f = MAX_PWM_DC;
        }

        //Clear the Pins so that we move forwards
        GPIO_Clear(THRUST_GPIO, THRUST_RIGHT_DIRECTION_PIN);
        GPIO_Clear(THRUST_GPIO, THRUST_LEFT_DIRECTION_PIN);

    }
    else  //Moving backwards
    {

        thrustLeft_f = 100 + (100*(float)thrust / 128.0 - LEFT_THRUST_OFFSET);
        thrustRight_f = 100 + (100*(float)thrust / 128.0 - RIGHT_THRUST_OFFSET);

        //Check if exceeds maximum PWM
        if (thrustLeft_f < (100 - MAX_PWM_DC))
        {
            thrustLeft_f = (100 - MAX_PWM_DC);
        }

        if (thrustRight_f < (100 - MAX_PWM_DC))
        {
            thrustRight_f = (100 - MAX_PWM_DC);
        }

        //Set the direction pins so that we move backwards
        GPIO_Set(THRUST_GPIO, THRUST_RIGHT_DIRECTION_PIN);
        GPIO_Set(THRUST_GPIO, THRUST_LEFT_DIRECTION_PIN);
    }

    //Command PWM
    setPWM_value(thrustLeft_f, THRUST_LEFT_PWM_PARAMATERS);
    setPWM_value(thrustRight_f, THRUST_RIGHT_PWM_PARAMATERS);

}

//Command the current lift by either passing in LIFT_ON or LIFT_OFF
void commandCurrentLift(uint8_t lift)
{
    //set the new PWM Value for the thrust
    if (lift == LIFT_ON)
    {
        GPIO_Set(LIFT_GPIO, LIFT_FAN_PIN);
    }
    else
    {
        GPIO_Clear(LIFT_GPIO, LIFT_FAN_PIN);
    }
}

//Somehow command the change in orientation based on the orientation that was
passed in
void commandCurrentOrientation(int8_t orientation)
{

    //Store the value of the orientation locally
    int8_t value = orientation;

    //Create PWM DC's for the left and right
    float leftPWM;
    float rightPWM;

    if (value < ((-1)*TURN_THRESHOLD)) //turn left
    {
        float left_brake = LEFT_BRAKE_UP + ((float)(-TURN_THRESHOLD - value) * (
                           float)(LEFT_BRAKE_DOWN-LEFT_BRAKE_UP)) /((float)(127.
                           0 - TURN_THRESHOLD));
        //float left_brake = LEFT_BRAKE_UP + ((float)((-1)*value -
                             TURN_THRESHOLD) * abs(LEFT_BRAKE_DOWN-LEFT_BRAKE_UP)
                             ) /(128.0 - TURN_THRESHOLD);
        leftPWM = left_brake;
        rightPWM = RIGHT_BRAKE_UP;

    }
    else if (value > TURN_THRESHOLD) //turn right
    {
        float right_brake = RIGHT_BRAKE_UP + ((float)(value - TURN_THRESHOLD) * (
                            float)(RIGHT_BRAKE_DOWN-RIGHT_BRAKE_UP)) / ((float)(
                            128.0 - TURN_THRESHOLD));
        //float right_brake = RIGHT_BRAKE_UP - ((float)(value - TURN_THRESHOLD)
                              * abs(RIGHT_BRAKE_UP-RIGHT_BRAKE_DOWN)) / (128.0 -
                              TURN_THRESHOLD);

        leftPWM = LEFT_BRAKE_UP;
        rightPWM = right_brake;
    }
    else   //Don't turn
    {
        leftPWM = LEFT_BRAKE_UP;
        rightPWM = RIGHT_BRAKE_UP;
    }

    //Command the PWM values
    setPWM_value(leftPWM, LEFT_SERVO_PWM_PARAMATERS);
    setPWM_value(rightPWM, RIGHT_SERVO_PWM_PARAMATERS);

}

void specialAction(void)
{
    //start playing our song
    startPlayingSong();

    setPWM_value(LEFT_BRAKE_DOWN, LEFT_SERVO_PWM_PARAMATERS);
    setPWM_value(RIGHT_BRAKE_DOWN, RIGHT_SERVO_PWM_PARAMATERS);

    //turn on fan full blast to differentiate from braking
    setPWM_value(85, THRUST_LEFT_PWM_PARAMATERS);
    setPWM_value(85, THRUST_RIGHT_PWM_PARAMATERS);
    special_action = true;
    ES_Timer_InitTimer(SPECIAL_ACTION_TIMER, SPECIAL_ACTION_TIME);
}

//getter function for master
bool isSpecialAction()
{
    return special_action;
}

// Command Breaking based on what was passed in
void commandBreak(void)
{
    setPWM_value(LEFT_BRAKE_DOWN, LEFT_SERVO_PWM_PARAMATERS);
    setPWM_value(RIGHT_BRAKE_DOWN, RIGHT_SERVO_PWM_PARAMATERS);
    //turn fans off
    setPWM_value(10, THRUST_LEFT_PWM_PARAMATERS);
    setPWM_value(10, THRUST_RIGHT_PWM_PARAMATERS);
    GPIO_Clear(THRUST_GPIO, THRUST_RIGHT_DIRECTION_PIN);
    GPIO_Clear(THRUST_GPIO, THRUST_LEFT_DIRECTION_PIN);
}

//moves both brakes up
void disengageBrakes()
{
    setPWM_value(LEFT_BRAKE_UP, LEFT_SERVO_PWM_PARAMATERS);
    setPWM_value(RIGHT_BRAKE_UP, RIGHT_SERVO_PWM_PARAMATERS);
}

/****************************************************************************
 DMC Related Function
 ***************************************************************************/

//Set the Color Pin According to the latest color received (initial color does
                                                            not matter upon
                                                            startup)
void setCurrentColor(uint8_t color)
{
```

```c
    //if the color is blue set the pin and if not clear it
    if (color == COLOR_DEACTIVATE)
    {
        //no way to turn both off
    }
    else
    {
        if (color == COLOR_BLUE)
        {
            GPIO_Set(DMC_BASE, COLOR_PIN);
        }
        else
        {
            GPIO_Clear(DMC_BASE, COLOR_PIN);
        }
    }
}

//setter function for pairing status
void setCurrentPairingStatus(uint8_t paired)
{

    if (paired == PAIRED_BIT)
    {
        GPIO_Set(DMC_BASE, PAIRED_PIN);
        pairedFlag = true;
    }
    else
    {
        GPIO_Clear(DMC_BASE, PAIRED_PIN);
        pairedFlag = false;
    }

}

//Pulse the Timer Clock
void pulseTimerClock(void)
{
    GPIO_Clear(DMC_BASE, TIMING_PIN);
    for (uint16_t i; i < 2000; i++)
    {
    }
    GPIO_Set(DMC_BASE, TIMING_PIN);

    printf("\n\r Timer");
}

/*------------------------------- Sound Related Functions -----------------------
--------*/
void startPlayingSong(void)
{
    GPIO_Clear(SOUND_BASE, SOUND_PIN);
}

void stopPlayingSong(void)
{
    GPIO_Set(SOUND_BASE, SOUND_PIN);
}
```