ES_Configure.h header file for Servo PIC configuration

```
/****************************************************************************
 Module
     ES_Configure.h
 Description
     This file contains macro definitions that are edited by the user to
     adapt the Events and Services framework to a particular application.
 Notes

 History
 When            Who     What/Why
 -------------- ---     --------
 10/21/13 20:54 jec      lots of added entries to bring the number of timers
                         and services up to 16 each
 08/06/13 14:10 jec      removed PostKeyFunc stuff since we are moving that
                         functionality out of the framework and putting it
                         explicitly into the event checking functions
 01/15/12 10:03 jec      started coding
****************************************************************************/

#ifndef CONFIGURE_H
#define CONFIGURE_H

/****************************************************************************/
// The maximum number of services sets an upper bound on the number of
// services that the framework will handle. Reasonable values are 8 and 16
// corresponding to an 8-bit(uint8_t) and 16-bit(uint16_t) Ready variable size
#define MAX_NUM_SERVICES 8

/****************************************************************************/
// This macro determines that nuber of services that are *actually* used in
// a particular application. It will vary in value from 1 to MAX_NUM_SERVICES
#define NUM_SERVICES 2

/****************************************************************************/
// These are the definitions for Service 0, the lowest priority service.
// Every Events and Services application must have a Service 0. Further
// services are added in numeric sequence (1,2,3,...) with increasing
// priorities
// the header file with the public fuction prototypes
#define SERV_0_HEADER "SPISlaveService.h"
// the name of the Init function
#define SERV_0_INIT InitSPISlaveService
// the name of the run function
#define SERV_0_RUN RunSPISlaveService
// How big should this services Queue be?
#define SERV_0_QUEUE_SIZE 3

/****************************************************************************/
// The following sections are used to define the parameters for each of the
// services. You only need to fill out as many as the number of services
// defined by NUM_SERVICES
/****************************************************************************/
// These are the definitions for Service 1
#if NUM_SERVICES > 1
// the header file with the public fuction prototypes
#define SERV_1_HEADER "ServoService.h"
// the name of the Init function
#define SERV_1_INIT InitServoService
```

```
// the name of the run function
#define SERV_1_RUN RunServoService
// How big should this services Queue be?
#define SERV_1_QUEUE_SIZE 3
#endif

/*****************************************************************************/
// These are the definitions for Service 2
#if NUM_SERVICES > 2
// the header file with the public fuction prototypes
#define SERV_2_HEADER "SPIService.h"
// the name of the Init function
#define SERV_2_INIT Init_SPIService
// the name of the run function
#define SERV_2_RUN Run_SPIService
// How big should this services Queue be?
#define SERV_2_QUEUE_SIZE 3
#endif

/*****************************************************************************/
// These are the definitions for Service 3
#if NUM_SERVICES > 3
// the header file with the public fuction prototypes
#define SERV_3_HEADER "TestHarnessService3.h"
// the name of the Init function
#define SERV_3_INIT InitTestHarnessService3
// the name of the run function
#define SERV_3_RUN RunTestHarnessService3
// How big should this services Queue be?
#define SERV_3_QUEUE_SIZE 3
#endif

/*****************************************************************************/
// These are the definitions for Service 4
#if NUM_SERVICES > 4
// the header file with the public fuction prototypes
#define SERV_4_HEADER "TestHarnessService4.h"
// the name of the Init function
#define SERV_4_INIT InitTestHarnessService4
// the name of the run function
#define SERV_4_RUN RunTestHarnessService4
// How big should this services Queue be?
#define SERV_4_QUEUE_SIZE 3
#endif

/*****************************************************************************/
// These are the definitions for Service 5
#if NUM_SERVICES > 5
// the header file with the public fuction prototypes
#define SERV_5_HEADER "TestHarnessService5.h"
// the name of the Init function
#define SERV_5_INIT InitTestHarnessService5
// the name of the run function
#define SERV_5_RUN RunTestHarnessService5
// How big should this services Queue be?
#define SERV_5_QUEUE_SIZE 3
#endif

/*****************************************************************************/
// These are the definitions for Service 6
```

```
#if NUM_SERVICES > 6
// the header file with the public fuction prototypes
#define SERV_6_HEADER "TestHarnessService6.h"
// the name of the Init function
#define SERV_6_INIT InitTestHarnessService6
// the name of the run function
#define SERV_6_RUN RunTestHarnessService6
// How big should this services Queue be?
#define SERV_6_QUEUE_SIZE 3
#endif

/****************************************************************************/
// These are the definitions for Service 7
#if NUM_SERVICES > 7
// the header file with the public fuction prototypes
#define SERV_7_HEADER "TestHarnessService7.h"
// the name of the Init function
#define SERV_7_INIT InitTestHarnessService7
// the name of the run function
#define SERV_7_RUN RunTestHarnessService7
// How big should this services Queue be?
#define SERV_7_QUEUE_SIZE 3
#endif

/****************************************************************************/
// These are the definitions for Service 8
#if NUM_SERVICES > 8
// the header file with the public fuction prototypes
#define SERV_8_HEADER "TestHarnessService8.h"
// the name of the Init function
#define SERV_8_INIT InitTestHarnessService8
// the name of the run function
#define SERV_8_RUN RunTestHarnessService8
// How big should this services Queue be?
#define SERV_8_QUEUE_SIZE 3
#endif

/****************************************************************************/
// These are the definitions for Service 9
#if NUM_SERVICES > 9
// the header file with the public fuction prototypes
#define SERV_9_HEADER "TestHarnessService9.h"
// the name of the Init function
#define SERV_9_INIT InitTestHarnessService9
// the name of the run function
#define SERV_9_RUN RunTestHarnessService9
// How big should this services Queue be?
#define SERV_9_QUEUE_SIZE 3
#endif

/****************************************************************************/
// These are the definitions for Service 10
#if NUM_SERVICES > 10
// the header file with the public fuction prototypes
#define SERV_10_HEADER "TestHarnessService10.h"
// the name of the Init function
#define SERV_10_INIT InitTestHarnessService10
// the name of the run function
#define SERV_10_RUN RunTestHarnessService10
// How big should this services Queue be?
```

```
#define SERV_10_QUEUE_SIZE 3
#endif

/****************************************************************************/
// These are the definitions for Service 11
#if NUM_SERVICES > 11
// the header file with the public fuction prototypes
#define SERV_11_HEADER "TestHarnessService11.h"
// the name of the Init function
#define SERV_11_INIT InitTestHarnessService11
// the name of the run function
#define SERV_11_RUN RunTestHarnessService11
// How big should this services Queue be?
#define SERV_11_QUEUE_SIZE 3
#endif

/****************************************************************************/
// These are the definitions for Service 12
#if NUM_SERVICES > 12
// the header file with the public fuction prototypes
#define SERV_12_HEADER "TestHarnessService12.h"
// the name of the Init function
#define SERV_12_INIT InitTestHarnessService12
// the name of the run function
#define SERV_12_RUN RunTestHarnessService12
// How big should this services Queue be?
#define SERV_12_QUEUE_SIZE 3
#endif

/****************************************************************************/
// These are the definitions for Service 13
#if NUM_SERVICES > 13
// the header file with the public fuction prototypes
#define SERV_13_HEADER "TestHarnessService13.h"
// the name of the Init function
#define SERV_13_INIT InitTestHarnessService13
// the name of the run function
#define SERV_13_RUN RunTestHarnessService13
// How big should this services Queue be?
#define SERV_13_QUEUE_SIZE 3
#endif

/****************************************************************************/
// These are the definitions for Service 14
#if NUM_SERVICES > 14
// the header file with the public fuction prototypes
#define SERV_14_HEADER "TestHarnessService14.h"
// the name of the Init function
#define SERV_14_INIT InitTestHarnessService14
// the name of the run function
#define SERV_14_RUN RunTestHarnessService14
// How big should this services Queue be?
#define SERV_14_QUEUE_SIZE 3
#endif

/****************************************************************************/
// These are the definitions for Service 15
#if NUM_SERVICES > 15
// the header file with the public fuction prototypes
#define SERV_15_HEADER "TestHarnessService15.h"
```

```
// the name of the Init function
#define SERV_15_INIT InitTestHarnessService15
// the name of the run function
#define SERV_15_RUN RunTestHarnessService15
// How big should this services Queue be?
#define SERV_15_QUEUE_SIZE 3
#endif


/****************************************************************************/
// Name/define the events of interest
// Universal events occupy the lowest entries, followed by user-defined events
typedef enum {   ES_NO_EVENT = 0,
                 ES_ERROR,   /* used to indicate an error from the service */
                 ES_INIT,    /* used to transition from initial pseudo-state */
                 ES_TIMEOUT, /* signals that the timer has expired */
                 New_SPI_Byte,
                 TMR2_Timeout,
                 Go_to_Safe_Mode,
               } ES_EventTyp_t ;

/****************************************************************************/
// These are the definitions for the Distribution lists. Each definition
// should be a comma seperated list of post functions to indicate which
// services are on that distribution list.
#define NUM_DIST_LISTS 0
#if NUM_DIST_LISTS > 0
#define DIST_LIST0 PostTestHarnessService0, PostTestHarnessService0
#endif
#if NUM_DIST_LISTS > 1
#define DIST_LIST1 PostTestHarnessService1, PostTestHarnessService1
#endif
#if NUM_DIST_LISTS > 2
#define DIST_LIST2 PostTemplateFSM
#endif
#if NUM_DIST_LISTS > 3
#define DIST_LIST3 PostTemplateFSM
#endif
#if NUM_DIST_LISTS > 4
#define DIST_LIST4 PostTemplateFSM
#endif
#if NUM_DIST_LISTS > 5
#define DIST_LIST5 PostTemplateFSM
#endif
#if NUM_DIST_LISTS > 6
#define DIST_LIST6 PostTemplateFSM
#endif
#if NUM_DIST_LISTS > 7
#define DIST_LIST7 PostTemplateFSM
#endif

/****************************************************************************/
// This are the name of the Event checking funcion header file.
#define EVENT_CHECK_HEADER "EventCheckers.h"

/****************************************************************************/
// This is the list of event checking functions
#define EVENT_CHECK_LIST SPI_Buffer_Check, TMR2_Timeout_Check

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

```
// These are the definitions for the post functions to be executed when the
// correspnding timer expires. All 16 must be defined. If you are not using
// a timer, then you should use TIMER_UNUSED
// Unlike services, any combination of timers may be used and there is no
// priority in servicing them
#define TIMER_UNUSED ((pPostFunc)0)
#define TIMER0_RESP_FUNC PostSPISlaveService
#define TIMER1_RESP_FUNC TIMER_UNUSED
#define TIMER2_RESP_FUNC TIMER_UNUSED
#define TIMER3_RESP_FUNC TIMER_UNUSED
#define TIMER4_RESP_FUNC TIMER_UNUSED
#define TIMER5_RESP_FUNC TIMER_UNUSED
#define TIMER6_RESP_FUNC TIMER_UNUSED
#define TIMER7_RESP_FUNC TIMER_UNUSED
#define TIMER8_RESP_FUNC TIMER_UNUSED
#define TIMER9_RESP_FUNC TIMER_UNUSED
#define TIMER10_RESP_FUNC TIMER_UNUSED
#define TIMER11_RESP_FUNC TIMER_UNUSED
#define TIMER12_RESP_FUNC TIMER_UNUSED
#define TIMER13_RESP_FUNC TIMER_UNUSED
#define TIMER14_RESP_FUNC TIMER_UNUSED
#define TIMER15_RESP_FUNC TIMER_UNUSED

/****************************************************************************/
// Give the timer numbers symbolc names to make it easier to move them
// to different timers if the need arises. Keep these definitons close to the
// definitions for the response functions to make it easier to check that
// the timer number matches where the timer event will be routed
// These symbolic names should be changed to be relevant to your application

#define SERVICE0_TIMER 15
#define SERVICE1_TIMER 14
#define SERVICE2_TIMER 13
#define SERVICE3_TIMER 12
#define SERVICE4_TIMER 11
#define SERVICE5_TIMER 10
#define SERVICE6_TIMER 9
#define SERVICE7_TIMER 8
#define SERVICE8_TIMER 7
#define SERVICE9_TIMER 6
#define SERVICE10_TIMER 5
#define SERVICE11_TIMER 4
#define SERVICE12_TIMER 3
#define SERVICE13_TIMER 2
#define SERVICE14_TIMER 1
#define SERVICE15_TIMER 0

#endif /* CONFIGURE_H */
```

EventCheckers.h header file for Servo PIC event checker routine

```
/****************************************************************************
 Module
     EventCheckers.h
 Description
     header file for the event checking functions
 Notes
```

```
 History
 When           Who     What/Why
 -------------- ---     --------
*****************************************************************************/

#ifndef EventCheckers_H
#define EventCheckers_H

#include "bitdefs.h"
#include "SPISlaveService.h"

//note that Timer 2 is sent to generate an interrupt every 13.06 ms
#define IDLE_TIMEOUT_COUNT          8       //~100 ms
#define SAFE_MODE_TIMEOUT_COUNT 153     //~2 s


// prototypes for event checkers

bool SPI_Buffer_Check(void);
bool TMR2_Timeout_Check(void);

// prototypes for other public functions
void Set_TMR2_Flag(void);


#endif /* EventCheckers_H */
```

EventCheckers.c source file for Servo PIC event checker routine

```
/*****************************************************************************
 Module
   EventCheckers.c

 Revision
   1.0.1

 Description
   This is the sample for writing event checkers along with the event
   checkers used in the basic framework test harness.

 Notes

 History
 When           Who     What/Why
 -------------- ---     --------

*****************************************************************************/

// this will pull in the symbolic definitions for events, which we will want
// to post in response to detecting events
#include "ES_Configure.h"
// this will get us the structure definition for events, which we will need
// in order to post events in response to detecting events
#include "ES_Events.h"
// if you want to use distribution lists then you need those function
// definitions too.
#include "ES_PostList.h"
// This include will pull in all of the headers from the service modules
```

```
// providing the prototypes for all of the post functions
#include "ES_ServiceHeaders.h"
// this test harness for the framework references the serial routines that
// are defined in ES_Port.c
#include "ES_Port.h"
// include our own prototypes to insure consistency between header &
// actual functionsdefinition
#include "EventCheckers.h"

#include "SPISlaveService.h"

static unsigned char TMR2_Timeout_Count = 0;
static unsigned char TMR2_Timeout_Flag = 0;


bool SPI_Buffer_Check(void)
{
    bool ReturnVal = false;

    ES_Event ThisEvent;

    if (BF == 1)
        {

            ThisEvent.EventType = New_SPI_Byte;

            PostSPISlaveService(ThisEvent);

            TMR2_Timeout_Count = 0; //Reset Counter
        }

    return ReturnVal;
}

bool TMR2_Timeout_Check(void)
{
    bool ReturnVal = false;

    ES_Event ThisEvent;

    if (TMR2_Timeout_Flag == 1)
        {

            TMR2_Timeout_Flag == 0;

            //If a small delay has occurred since last byte transfer, return to idle state
            //If a large delay has occurred that indicates loss of communication with controller, enter safe
mode
            if ( TMR2_Timeout_Count >= SAFE_MODE_TIMEOUT_COUNT)
                {
                    ThisEvent.EventType = Go_to_Safe_Mode;
                    PostSPISlaveService(ThisEvent);
                }
            else if ( TMR2_Timeout_Count >= IDLE_TIMEOUT_COUNT) //We have to use multiple
interrupts from the timer 2 counter
                {

                    ThisEvent.EventType = TMR2_Timeout;
                    PostSPISlaveService(ThisEvent);
```

```
            }


        }

    return ReturnVal;
}


//Other Functions
void Set_TMR2_Flag( void )
{
    TMR2_Timeout_Flag = 1; //Set flag that can be read by event checker on the PIC

    TMR2_Timeout_Count ++; //Track consecutive timeouts so that we can identify a communication failure that should put the servo driver into safe moe

}
```

SPISlaveService.h header file for Servo PIC SPI communications service

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

   Header file for SPISlaveService
   based on the Gen 2 Events and Services Framework

 ****************************************************************************/
//Ported over from ME218B project for use on a PIC16F690 by M. Hoffman, 05/10/2014
//Added A/D converter functionality for testing purposes with up to three A/D input channels

#ifndef SPISlaveService_H
#define SPISlaveService_H

#include "ES_Types.h"
#include "ES_Configure.h"
#include "ES_Timers.h"
#include "Bin_Const.h"
#include "htc.h"

#include "ServoService.h"

#define LED0    RC5
#define LED1    RC4
#define LED2    RC3
#define FIRE_PORT RB7

#define CONTROL_HEADER      0x01
#define DISPLAY_COMMAND     0x0A

#define CONTROL_BYTES 0x06   //Number of bytes following the control header

#define KICK_THRESHOLD                  0x08

//Designate servo channels
#define THROTTLE_SERVO 0
#define STEERING_SERVO 1
#define KICKING_SERVO   2
```

```
#define DISPLAY_SERVO   3

//Define neutral servo positions
#define THROTTLE_NEUTRAL    1000
#define STEERING_NEUTRAL    1540
#define KICKING_NEUTRAL     900
#define DISPLAY_NEUTRAL     2000

#define KICKING_ACTIVE      2100

#define MAX_THROTTLE        2000
#define MIN_THROTTLE        1000

#define MAX_STEERING        1750
#define MIN_STEERING 1325

#define MAX_DISPLAY   2000
#define MIN_DISPLAY   1000


// typedefs for the states
// State definitions for use with the query function
typedef enum { InitPState, Safe_Mode, Idle, Wait_for_Control_Bytes, Wait_for_Energy_Status}
TemplateState_t ;


// Public Function Prototypes

bool InitSPISlaveService ( uint8_t Priority );
bool PostSPISlaveService( ES_Event ThisEvent );
ES_Event RunSPISlaveService( ES_Event ThisEvent );

#endif /* SPISlaveService_H */
```

SPISlaveService.c source file for Servo PIC SPI communications service

```
/*******************************************************************************
 Module
   SPISlaveService.c

 Revision
   1.0.1

 Description

 Notes

 History
 When            Who     What/Why
 -------------- ---     --------

*******************************************************************************/

//Ported over from ME218B project for use on a PIC16F690 by M. Hoffman, 05/10/2014

/*----------------------------- 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 "SPISlaveService.h"

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

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

TemplateState_t Decode_Command_Header(unsigned char Header_Byte);

void Decode_Throttle_Command(signed char Throttle_Command);
void Decode_Steering_Command(signed char Steering_Command);
void Decode_AUX_Command(unsigned char AUX_Command);
void Decode_Whimsy_Command(unsigned char Whimsy_Command);
void Decode_Display_Command(unsigned char Display_Command);

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

static TemplateState_t CurrentState;

static unsigned char Energy_Level = 7; //Initialize energy level to 7 so that,
//in the event of a communications failure or error, we can still kick


/*---------------------------- Module Code ----------------------------*/
/****************************************************************************
 Function
     InitSPISlaveService

 Parameters
     uint8_t : the priorty of this service

 Returns
     boolean, 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 InitSPISlaveService ( uint8_t Priority )
{
    ES_Event ThisEvent;

    CurrentState = InitPState;

    MyPriority = Priority;

    unsigned char utility_variable; //use for reading SSPBUF so that we can initialize with BF flag cleared

//Configure hardware
```

```
//Configure Timer 2 to handle byte timeout (return to idle) and communication failure (return to safe mode)
events
    T2CKPS0 = 1;        //T2CKPS<1:0> = 11 in T2CON for 1/16 prescaler
    T2CKPS1 = 1;
    TOUTPS0 = 1;        //TOUTPS<3:0> = 1111 in T2CON for 1/16 postscaler
    TOUTPS1 = 1;
    TOUTPS2 = 1;
    TOUTPS3 = 1;
    PR2 = 255; //max value we can use for 8-bit timer period
    TMR2ON = 1;         //turn on Timer 2

//Clear interrupt flags
    TMR2IF = 0;

//Enable interrupts for Timer 2
    TMR2IE = 1;
    PEIE = 1;

//Configure SPI system
    SMP = 0;
    CKE = 0; //SPI Mode 3
    CKP = 1;
    SSPIF = 0; //Clear SPI interrupt flag
    utility_variable = SSPBUF; //Clear BF flag
//SSPM<3:0> = 0010 for master mode, sck = f_osc/64
//SSPM<3:0> = 0100 for slave mode, slave select enabled
    SSPM3 = 0;
    SSPM2 = 1;
    SSPM1 = 0;
    SSPM0 = 0;
    SSPEN = 1; //enable SPI system
//SSPIE = 1; //enable SPI interrupts
    PEIE = 1; //enable peripheral interrupts
    ANS8 = 0; //set slave select to digital input
    TRISC6 = 1;
    ANS9 = 0; //set SDO to digital input since we are not wiring the servo PIC to return a response
    TRISC7 = 1;
    TRISB6 = 1; //set SCK to output for master, input for slave
    ANS10 = 0; //set SDI to digital input
    TRISB4 = 1;

//Enable display of energy status LEDs for initial debugging purposes
    ANS7 = 0;
    TRISC3 = 0;
    TRISC4 = 0;
    TRISC5 = 0;

//Enable turret gun firing for whimsy function
    TRISB7 = 0;
    RB7 = 0;

    GIE = 1; //Enable global interrupts, INTCON register

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

```
        {
            return false;
        }
}

/****************************************************************************
 Function
     PostSPISlaveService

 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 PostSPISlaveService( ES_Event ThisEvent )
{
    return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunSPISlaveService

 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 RunSPISlaveService( ES_Event ThisEvent )
{

    ES_Event ReturnEvent;

    unsigned char dummy_variable;

    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

    static unsigned char Control_Byte = 0; //keep track of which control byte we are in;

    switch ( CurrentState )
        {
        case InitPState :          // If current state is initial Psedudo State
            if ( ThisEvent.EventType == ES_INIT )// only respond to ES_Init
                {
```

```
                    // this is where you would put any actions associated with the
                    // transition from the initial pseudo-state into the actual
                    // initial state

                    //Initialize servos to default/neutral positions
                    UpdateServo(THROTTLE_SERVO, THROTTLE_NEUTRAL);
                    UpdateServo(STEERING_SERVO, STEERING_NEUTRAL);
                    UpdateServo(KICKING_SERVO,   KICKING_NEUTRAL);
                    UpdateServo(DISPLAY_SERVO,   DISPLAY_NEUTRAL);

                    // now put the machine into the actual initial state
                    CurrentState = Safe_Mode;
                }
            break;

        case Safe_Mode :          // If current state is state one

            UpdateServo(THROTTLE_SERVO, THROTTLE_NEUTRAL);
            UpdateServo(STEERING_SERVO, STEERING_NEUTRAL);
            UpdateServo(KICKING_SERVO,   KICKING_NEUTRAL);
            UpdateServo(DISPLAY_SERVO,   DISPLAY_NEUTRAL);

            switch ( ThisEvent.EventType )
                {

                case New_SPI_Byte :
                {
                    CurrentState = Decode_Command_Header(SSPBUF); //Decode the header byte to
determine next state

                }
                break;

                }
            break;


        case Idle :

            switch ( ThisEvent.EventType )
                {

                case New_SPI_Byte :
                {
                    CurrentState = Decode_Command_Header(SSPBUF);

                }
                break;

                case Go_to_Safe_Mode :
                {

                    CurrentState = Safe_Mode; //We haven't been communicated with for a fairly long
interval, enter safe mode

                }
                break;

                }
            break;
```

```
        case Wait_for_Control_Bytes :

            switch ( ThisEvent.EventType )
                {

                case New_SPI_Byte :
                {
                    switch (Control_Byte)
                        {

                        case 0 :
                        {
                            Decode_Throttle_Command(SSPBUF); //Decode command for throttle
                        }
                        break;

                        case 1 :
                        {
                            Decode_Steering_Command(SSPBUF); //Decode command for steering
                        }
                        break;

                        case 2 :
                        {
                            Decode_AUX_Command(SSPBUF); //Decode auxilary command (kicking/braking)
                        }
                        break;

                        case 3 :
                        {
                            Decode_Whimsy_Command(SSPBUF); //Decode whimsy byte
                        }
                        break;

                        case 4 :
                        {
                            dummy_variable = SSPBUF;//Do nothing for now (other than read SSPBUF), this
is the translation byte
                        }
                        break;

                        case 5 :
                        {
                            Decode_Display_Command(SSPBUF);          //Decode energy display command
                        }
                        break;

                        }

                    Control_Byte ++; //Increment byte counter

                    if (Control_Byte < CONTROL_BYTES)
                        {
                            CurrentState = Wait_for_Control_Bytes; //Wait for next control byte
                        }
                    else
                        {
                            Control_Byte = 0; //Reset byte counter
```

```
                        CurrentState = Idle; //Wait for next command header

                    }


                }
                break;

                case TMR2_Timeout :
                {
                    Control_Byte = 0; //Reset byte counter

                    CurrentState = Idle; //We haven't received a byte for a while, return to idle state and
wait for next command header

                }
                break;

                case Go_to_Safe_Mode :
                {
                    Control_Byte = 0; //Reset byte counter

                    CurrentState = Safe_Mode; //We haven't been communicated with for a fairly long
interval, enter safe mode

                }
                break;

                }
            break;



        case Wait_for_Energy_Status :

            switch ( ThisEvent.EventType )
                {

                case New_SPI_Byte :
                {
                    Decode_Display_Command(SSPBUF); //Decode command for axillary functions

                    CurrentState = Idle;

                }
                break;

                case TMR2_Timeout :
                {

                    CurrentState = Idle; //We haven't received a byte for a while, return to idle state and
wait for next command header

                }
                break;

                case Go_to_Safe_Mode :
                {

                    CurrentState = Safe_Mode; //We haven't been communicated with for a fairly long
```

```
interval, enter safe mode

                }
                break;

                }
            break;



        }


    return ReturnEvent;

}


//Module-Private Functions:

TemplateState_t Decode_Command_Header(unsigned char Header_Byte)
{
    TemplateState_t Next_State = Idle; //Default return mode is Idle state, a timeout will pull us back to
Safe Mode

    switch (Header_Byte)
        {
        case CONTROL_HEADER :

            Next_State = Wait_for_Control_Bytes;

            break;

        case DISPLAY_COMMAND :

            Next_State = Wait_for_Energy_Status;

            break;

        }

    return Next_State;

}

void Decode_Throttle_Command(signed char Throttle_Command)
{
    uint16_t Scaled_Throttle;

    if (Throttle_Command <= 0)
        {
            Scaled_Throttle = MIN_THROTTLE;

        }
    else
        {
            Scaled_Throttle = MIN_THROTTLE + (((long)(Throttle_Command))*(MAX_THROTTLE -
MIN_THROTTLE))/127;
        }
```

```
    if (Scaled_Throttle < MIN_THROTTLE)
        {
            UpdateServo(THROTTLE_SERVO, MIN_THROTTLE);
        }
    else if (Scaled_Throttle > MAX_THROTTLE)
        {
            UpdateServo(THROTTLE_SERVO, MAX_THROTTLE);
        }
    else
        {
            UpdateServo(THROTTLE_SERVO, Scaled_Throttle);
        }

}

void Decode_Steering_Command(signed char Steering_Command)
{
    uint16_t Scaled_Steering;

    volatile static signed char Mirror_Steering_Command;

    Mirror_Steering_Command = Steering_Command; //for viewing in PIC debugger

    if (Steering_Command <= -118)
        {
            Scaled_Steering = MIN_STEERING;
        }
    else if (Steering_Command >= 117)
        {
            Scaled_Steering = MAX_STEERING;
        }
    else
        {

            if (Steering_Command >= 0)
                {
                    Scaled_Steering = (uint16_t)(STEERING_NEUTRAL +
((((long)(Steering_Command))*(MAX_STEERING - STEERING_NEUTRAL))/127));
                }
            else
                {
                    Scaled_Steering = (uint16_t)((((long)(-Steering_Command))*(STEERING_NEUTRAL -
MIN_STEERING))/128);

                    if (Scaled_Steering > STEERING_NEUTRAL)
                        {
                            Scaled_Steering = MIN_STEERING;
                        }
                    else
                        {
                            Scaled_Steering = STEERING_NEUTRAL - Scaled_Steering;
                        }

                }

        }

    if (Scaled_Steering < MIN_STEERING)
        {
            UpdateServo(STEERING_SERVO, MIN_STEERING);
```

```
        }
    else if (Scaled_Steering > MAX_STEERING)
        {
            UpdateServo(STEERING_SERVO, MAX_STEERING);

        }
    else
        {
            UpdateServo(STEERING_SERVO, Scaled_Steering);
        }

}


void Decode_AUX_Command(unsigned char AUX_Command)
{

    if ((AUX_Command >= KICK_THRESHOLD)&&(Energy_Level > 1))
        {
            UpdateServo(KICKING_SERVO, KICKING_ACTIVE);

        }
    else
        {
            UpdateServo(KICKING_SERVO, KICKING_NEUTRAL);

        }

}

void Decode_Whimsy_Command(unsigned char Whimsy_Command)
{

    if (Whimsy_Command >= 0xAA)
        {
            FIRE_PORT = 1;
        }
    else
        {
            FIRE_PORT = 0;
        }

}


void Decode_Display_Command(unsigned char Display_Command)
{

    uint16_t Scaled_Display;

    unsigned char Display_Command_Mirror = Display_Command;

    if (Display_Command <= 0x07) //Only process this command if it represents a valid energy status
update
        {
            Energy_Level = Display_Command; //Update module-level variable so that the kicking routine
knows when the eneryg is too low to kick

            Scaled_Display = (uint16_t)(MAX_DISPLAY - (((long)Display_Command)*(MAX_DISPLAY -
```

```
MIN_DISPLAY))/7);

            if (Scaled_Display < MIN_DISPLAY)
                {
                    Scaled_Display = MIN_DISPLAY;
                }
            else if (Scaled_Display > MAX_DISPLAY)
                {
                    Scaled_Display = MAX_DISPLAY;
                }

            UpdateServo(DISPLAY_SERVO, Scaled_Display);

            //Update LEDS
            if ((Display_Command & BIT0HI) == BIT0HI)
                {
                    LED2 = 1;
                }
            else
                {
                    LED2 = 0;
                }

            if ((Display_Command & BIT1HI) == BIT1HI)
                {
                    LED1 = 1;
                }
            else
                {
                    LED1 = 0;
                }

            if ((Display_Command & BIT2HI) == BIT2HI)
                {
                    LED0 = 1;
                }
            else
                {
                    LED0 = 0;
                }

        }

}


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

ServoService.h header file for Servo PIC servo signal generation service

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

   Header file for ServoService
   based on the Gen 2 Events and Services Framework

  ****************************************************************************/
//Ported over from ME218B project for use on a PIC16F690 by M. Hoffman, 05/10/2014
```

```
//Added A/D converter functionality for testing purposes with up to three A/D input channels

#ifndef ServoService_H
#define ServoService_H

#include "ES_Types.h"
#include "ES_Configure.h"
#include "ES_Timers.h"
#include "Bin_Const.h"
#include "htc.h"

#include "SPISlaveService.h"
#include "EventCheckers.h"

//#define SERVO_TEST_HARNESS

//define number of servo channels to activate
#define NUMBER_OF_SERVOS 4

//map servo channel ports and data direction registers
//Servo channel 0:
#define SERVO_0_PT RA0
#define SERVO_0_ANSEL ANS0
#define SERVO_0_DDR TRISA
#define SERVO_0_BITHI BIT0HI
//Servo channel 1:
#define SERVO_1_PT RA1
#define SERVO_1_ANSEL ANS1
#define SERVO_1_DDR TRISA
#define SERVO_1_BITHI BIT1HI
//Servo channel 2:
#define SERVO_2_PT RA2
#define SERVO_2_ANSEL ANS2
#define SERVO_2_DDR TRISA
#define SERVO_2_BITHI BIT2HI
//Servo channel 3:
#define SERVO_3_PT RC0
#define SERVO_3_ANSEL ANS4
#define SERVO_3_DDR TRISC
#define SERVO_3_BITHI BIT0HI
//Servo channel 4:
#define SERVO_4_PT RA0
#define SERVO_4_ANSEL ANS0
#define SERVO_4_DDR TRISA
#define SERVO_4_BITHI BIT0HI

//define default servo pulse widths in units of microseconds
#define SERVO_PERIOD_MICROSECONDS 20000 //Period of pulses
#define SERVO_0_DEFAULT 1000
#define SERVO_1_DEFAULT 1000
#define SERVO_2_DEFAULT 1000
#define SERVO_3_DEFAULT 1000
#define SERVO_4_DEFAULT 1000

#define MIN_PULSE_WIDTH 900
#define MAX_PULSE_WIDTH 2100

#define TICKS_PER_MICROSECOND 0.625//assumes F_oscillator = 20 MHz, F_clock = F_oscillator/4, divide
F_clock by 8
```

```
// Public Function Prototypes

bool InitServoService ( uint8_t Priority );
bool PostServoService( ES_Event ThisEvent );
ES_Event RunServoService( ES_Event ThisEvent );
void UpdateServo(unsigned char ServoChannel, uint16_t PulseWidthMicroseconds);

#endif /* ServoService_H */
```

ServoService.c source file for Servo PIC servo signal generation service

```
/*******************************************************************************
 Module
   ServoService.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
 -------------- ---     --------
*******************************************************************************/

//Ported over from ME218B project for use on a PIC16F690 by M. Hoffman, 05/10/2014

/*----------------------------- 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 "ServoService.h"

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

static uint16_t Servo_Pulse_Widths[5];


static uint16_t Servo_Period;


/*------------------------------ Module Code ------------------------------*/
/*******************************************************************************
 Function
     InitServoService

 Parameters
     uint8_t : the priorty of this service

 Returns
```

```
     boolean, 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 InitServoService ( uint8_t Priority )
{
    ES_Event ThisEvent;

    MyPriority = Priority;

//Set default servo pulse widths
    Servo_Pulse_Widths[0] = (uint16_t)(((float)SERVO_0_DEFAULT)*TICKS_PER_MICROSECOND);
    Servo_Pulse_Widths[1] = (uint16_t)(((float)SERVO_1_DEFAULT)*TICKS_PER_MICROSECOND);
    Servo_Pulse_Widths[2] = (uint16_t)(((float)SERVO_2_DEFAULT)*TICKS_PER_MICROSECOND);
    Servo_Pulse_Widths[3] = (uint16_t)(((float)SERVO_3_DEFAULT)*TICKS_PER_MICROSECOND);
    Servo_Pulse_Widths[4] = (uint16_t)(((float)SERVO_4_DEFAULT)*TICKS_PER_MICROSECOND);

//Configure servo period
    Servo_Period = (uint16_t)(((float)(20000 - 1000*NUMBER_OF_SERVOS))*TICKS_PER_MICROSECOND);

//Configure servo channel data direction registers
//and initialize output as low
    if (NUMBER_OF_SERVOS > 0)
        {
            SERVO_0_ANSEL = 0;
            SERVO_0_DDR &= ~SERVO_0_BITHI;
            SERVO_0_PT = 0;
        }

    if (NUMBER_OF_SERVOS > 1)
        {
            SERVO_1_ANSEL = 0;
            SERVO_1_DDR &= ~SERVO_1_BITHI;
            SERVO_1_PT = 0;
        }

    if (NUMBER_OF_SERVOS > 2)
        {
            SERVO_2_ANSEL = 0;
            SERVO_2_DDR &= ~SERVO_2_BITHI;
            SERVO_2_PT = 0;
        }

    if (NUMBER_OF_SERVOS > 3)
        {
            SERVO_3_ANSEL = 0;
            SERVO_3_DDR &= ~SERVO_3_BITHI;
            SERVO_3_PT = 0;
        }

    if (NUMBER_OF_SERVOS > 4)
        {
            SERVO_4_ANSEL = 0;
            SERVO_4_DDR &= ~SERVO_4_BITHI;
```

```
            SERVO_4_PT = 0;
        }

// Configure Timer 1 to time up to 5 servo channels
    SCS = 0;    //Configure clock to use external oscillator, OSCCON register
    T1CON = 0x31;      //Configure clock scaler to divide system clock by 8
    CCP1CON = 0x0A; //Configure Timer 1 for output compare mode, generate software interrupt, leave pin
alone
    CCPR = TMR1 + Servo_Period; //Initialize output compare register

// Clear interrupt flags
    CCP1IF = 0; //Clear CCP1 interrupt flag in PIR1 register

// Enable interrupts for Timer 1
    CCP1IE = 1; //Enable interrupt for CCP1 system in PIE1 register
    PEIE = 1; //Enable interrupts for peripherals in INTCON register
    GIE = 1; //Enable global interrupts, INTCON register

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

/****************************************************************************
 Function
      PostServoService

 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 PostServoService( ES_Event ThisEvent )
{
    return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunServoService

 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 RunServoService( ES_Event ThisEvent )
{
    ES_Event ReturnEvent;

    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

    return ReturnEvent;

}

void UpdateServo(unsigned char ServoChannel, uint16_t PulseWidthMicroseconds)
{

    //check for valid values
    if (PulseWidthMicroseconds < MIN_PULSE_WIDTH)
        {
            PulseWidthMicroseconds = MIN_PULSE_WIDTH;
        }
    else if (PulseWidthMicroseconds > MAX_PULSE_WIDTH)
        {
            PulseWidthMicroseconds = MAX_PULSE_WIDTH;
        }

    //update servo channel pulse with
    Servo_Pulse_Widths[ServoChannel] =
(uint16_t)(((float)PulseWidthMicroseconds)*TICKS_PER_MICROSECOND);

}


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

//loop through output compare ISR to fire pulses for five separate servo channels
void interrupt ActuateServos(void)
{

    //track current servo channel
    static unsigned char i = 0;

    //service pending interrupts

    if (CCP1IF == 1) //CCP1 System (Timer 1) Interrupt Flag
        {

            // clear flag
            CCP1IF = 0;

            //sequentially pulse servo channels
            switch (i)
```

```
{
case 0:
{
    SERVO_0_PT = 1;
}
break;

case 1:
{
    SERVO_0_PT = 0;

    if (NUMBER_OF_SERVOS > 1)
        {
            SERVO_1_PT = 1;
        }
}
break;

case 2:
{
    SERVO_1_PT = 0;

    if (NUMBER_OF_SERVOS > 2)
        {
            SERVO_2_PT = 1;
        }
}
break;

case 3:
{
    SERVO_2_PT = 0;

    if (NUMBER_OF_SERVOS > 3)
        {
            SERVO_3_PT = 1;
        }
}
break;

case 4:
{
    SERVO_3_PT = 0;

    if (NUMBER_OF_SERVOS > 4)
        {
            SERVO_4_PT = 1;
        }
}
break;

case 5:
{
    SERVO_4_PT = 0;
}
break;

default:
{
    //Error if we ever get here
```

```
                }

                }


            if (i < NUMBER_OF_SERVOS)
                {

                    CCPR += Servo_Pulse_Widths[i];

                    i++;

                }
            else
                {
                    CCPR += Servo_Period;

                    i = 0;
                }

        }

    if (TMR2IF == 1) //Timer 2 Interrupt Flag
        {
            TMR2IF = 0; //Clear Timer 2 Interrupt Flag

            Set_TMR2_Flag(); //Set flag in event checker (we can't post to services from an interrupt on the
PIC)

        }

}


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