```
/*******************************************************************************
 Module
   CalibrationServiceSM.c

 Revision
   2.0.1

 Description
   This is a template file for implementing state machines.

 Notes

 History
 When           Who     What/Why
 -------------- ---     --------
 02/07/13 21:00 jec      corrections to return variable (should have been
                         ReturnEvent, not CurrentEvent) and several EV_xxx
                         event names that were left over from the old version
 02/08/12 09:56 jec      revisions for the Events and Services Framework Gen2
 02/13/10 14:29 jec      revised Start and run to add new kind of entry
function
                         to make implementing history entry cleaner
 02/13/10 12:29 jec      added NewEvent local variable to During function and
                         comments about using either it or Event as the
return
 02/11/10 15:54 jec      more revised comments, removing last comment in
during
                         function that belongs in the run function
 02/09/10 17:21 jec      updated comments about internal transitions on
During funtion
 02/18/09 10:14 jec      removed redundant call to RunLowerlevelSM in
EV_Entry
                         processing in During function
 02/20/07 21:37 jec      converted to use enumerated type for events & states
 02/13/05 19:38 jec      added support for self-transitions, reworked
                         to eliminate repeated transition code
 02/11/05 16:54 jec      converted to implment hierarchy explicitly
 02/25/03 10:32 jec      converted to take a passed event parameter
 02/18/99 10:19 jec      built template from MasterMachine.c
 02/14/99 10:34 jec      Began Coding
*******************************************************************************/
/*----------------------------- Include Files -----------------------------*/
// Basic includes for a program using the Events and Services Framework
#include "ES_Configure.h"
#include "ES_Framework.h"

/* 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 "CalibrationServiceSM.h"

/*----------------------------- Module Defines ----------------------------*/
// define constants for the states for this machine
// and any other local defines

static short last_pot_input;
```

```
static unsigned char pot_value;

static unsigned char Button_State = 0;

static unsigned char last_dipswitch_value;

/*---------------------------- Module Functions ----------------------------*/
/* prototypes for private functions for this machine, things like during
   functions, entry & exit functions.They should be functions relevant to the
   behavior of this state machine
*/
//static ES_Event DuringOut_of_Balls( ES_Event Event);

/*---------------------------- Module Variables ----------------------------*/
// everybody needs a state variable, you may need others as well
static TemplateState_t CurrentState;

/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
    RunCalibrationServiceSM

 Parameters
   ES_Event: the event to process

 Returns
   ES_Event: an event to return

 Description
   add your description here
 Notes
   uses nested switch/case to implement the machine.
 Author
   J. Edward Carryer, 2/11/05, 10:45AM
****************************************************************************/
ES_Event RunCalibrationServiceSM( ES_Event CurrentEvent )
{
   unsigned char MakeTransition = false;/* are we making a state transition?
*/
   TemplateState_t NextState = CurrentState;
   ES_Event EntryEventKind = { ES_ENTRY, 0 };// default to normal entry to
new state
   ES_Event ReturnEvent = CurrentEvent; // assume we are not consuming event
   ES_Event PostEvent;

   short current_pot_input;

   unsigned char current_dipswitch_value;

   unsigned char Button_Value;

  #ifdef PRINT_CALIBRATION_SM_CALLS

    printf("RunCalibrationServiceSM\n\r");

  #endif
```

```
        current_pot_input = ADS12_ReadADPin(POT_PIN);

        current_dipswitch_value = get_dipswitch();

        Button_Value = ((PTIAD & PUSHBUTTON) == PUSHBUTTON);

        #ifdef PRINT_CALIBRATION_READS

                printf("Current Pot Input: %d\n\r", ADS12_ReadADPin(POT_PIN));
                printf("Dipswitch Value: %d\n\r", get_dipswitch());
                printf("Dipswitch 1: %d\n\r", get_dipswitch_1());
                printf("Dipswitch 2: %d\n\r", get_dipswitch_2());
                printf("Dipswitch 3: %d\n\r", get_dipswitch_3());
                printf("Pushbutton: %d\n\r", ((PTIAD & PUSHBUTTON) ==
PUSHBUTTON));

        #endif

        if(abs(current_pot_input - last_pot_input) > 10)
        {
                pot_value = (unsigned
char)(((float)(current_pot_input))*100/1024);

                PostEvent.EventType = New_Pot;

                PostEvent.EventParam = pot_value;

                PostMasterHSM(PostEvent);

        }

        if(last_dipswitch_value != current_dipswitch_value)
        {
                PostEvent.EventType = New_Dipswitch;

                PostEvent.EventParam = current_dipswitch_value;

                PostMasterHSM(PostEvent);

        }

        if(Button_Value == 0)
        {
            Button_State = 0;
        }
        else
        {
                switch(Button_State)
                {
                    case 0:
                    {
                        Button_State = 1;
                     }
                     break;

                     case 1:
```

```
                        {
                           Button_State = 2;

                           PostEvent.EventType = Button_Pushed;

                                        PostMasterHSM(PostEvent);

                        }
                        break;

                        case 2:
                        {
                        //do nothing now that button is debounced
                        }
                        break;
                     }
        }

        last_pot_input = current_pot_input;

        last_dipswitch_value = current_dipswitch_value;

        //Set timer for next read of calibration switches
        ES_Timer_SetTimer(CALIBRATION_TIMER, CALIBRATION_INTERVAL_MS);
        ES_Timer_StartTimer(CALIBRATION_TIMER);


    //   If we are making a state transition
    if (MakeTransition == true)
    {
       //   Execute exit function for current state
       CurrentEvent.EventType = ES_EXIT;
       RunCalibrationServiceSM(CurrentEvent);

       CurrentState = NextState; //Modify state variable

       //   Execute entry function for new state
       // this defaults to ES_ENTRY
       RunCalibrationServiceSM(EntryEventKind);
     }
     return(ReturnEvent);
}


/*****************************************************************************
 Function
     StartCalibrationServiceSM

 Parameters
     None

 Returns
     None

 Description
     Does any required initialization for this state machine
 Notes
```

```
 Author
     J. Edward Carryer, 2/18/99, 10:38AM
******************************************************************************/
void StartCalibrationServiceSM ( ES_Event CurrentEvent )
{
   // local variable to get debugger to display the value of CurrentEvent
   ES_Event LocalEvent = CurrentEvent;
   // to implement entry to a history state or directly to a substate
   // you can modify the initialization of the CurrentState variable
   // otherwise just start in the entry state every time the state machine
   // is started

  #ifdef PRINT_CALIBRATION_SM_CALLS

    printf("StartCalibrationServiceSM\n\r");

  #endif

    //Initialize analog port
    //ADS12_Init("OOOAIIII"); //Documentation suggests that first/leftmost
letter in string corresponds to MSB
        ADS12_Init("IIIIAOOO");

        //Initialize last_Pot_Value to present value of Port AD0
    last_pot_input = ADS12_ReadADPin(POT_PIN);
        pot_value = (unsigned char)(((float)(last_pot_input))*100/1024);

        //Initialize dip switch value
        last_dipswitch_value = get_dipswitch();

        //Set timer for next read of calibration switches
        ES_Timer_SetTimer(CALIBRATION_TIMER, CALIBRATION_INTERVAL_MS);
        ES_Timer_StartTimer(CALIBRATION_TIMER);

   //set initial state
   CurrentState = CalibrationState;
   // call the entry function (if any) for the ENTRY_STATE
   RunCalibrationServiceSM(CurrentEvent);

}


/*****************************************************************************
 Function
     QueryCalibrationServiceSM

 Parameters
     None

 Returns
     TemplateState_t The current state of the Template state machine

 Description
     returns the current state of the Template state machine
 Notes
```

```
 Author
     J. Edward Carryer, 2/11/05, 10:38AM
********************************************************************************/
TemplateState_t QueryCalibrationServiceSM ( void )
{
    #ifdef PRINT_SERVO_SM_CALLS

    printf("QueryCalibrationServiceSM\n\r");

  #endif

   return(CurrentState);
}


unsigned char get_cal_pot(void)
{
        return pot_value;
}

unsigned char get_pushbutton(void)
{
        unsigned char pushbutton = 0;

        if((PTIAD&PUSHBUTTON) == PUSHBUTTON)
        {
                pushbutton = 1;
        }

        return pushbutton;

}

unsigned char get_dipswitch(void)
{
        unsigned char dipswitch_1 = 0;
        unsigned char dipswitch_2 = 0;
        unsigned char dipswitch_3 = 0;
        unsigned char dipswitch_value;

        if((PTIAD&DIPSWITCH_1) == DIPSWITCH_1)
        {
                dipswitch_1 = 1;
        }

        if((PTIAD&DIPSWITCH_2) == DIPSWITCH_2)
        {
                dipswitch_2 = 1;
        }

        if((PTIAD&DIPSWITCH_3) == DIPSWITCH_3)
        {
                dipswitch_3 = 1;
        }

        dipswitch_value = 1*dipswitch_1 + 2*dipswitch_2 + 4*dipswitch_3;
```

```
        return dipswitch_value;

}

unsigned char get_dipswitch_1(void)
{
        unsigned char dipswitch_1 = 0;

        if((PTIAD&DIPSWITCH_1) == DIPSWITCH_1)
        {
                dipswitch_1 = 1;
        }

        return dipswitch_1;

}

unsigned char get_dipswitch_2(void)
{
        unsigned char dipswitch_2 = 0;

        if((PTIAD&DIPSWITCH_2) == DIPSWITCH_2)
        {
                dipswitch_2 = 1;
        }

        return dipswitch_2;

}

unsigned char get_dipswitch_3(void)
{
        unsigned char dipswitch_3 = 0;

        if((PTIAD&DIPSWITCH_3) == DIPSWITCH_3)
        {
                dipswitch_3 = 1;
        }

        return dipswitch_3;

}

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