```
/*******************************************************************************
 Module
   BalanceFSM.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 "GameFSM.h"
#include <stdio.h>
#include <math.h>
#include "VibrationControl.h"
#include "BalanceFSM.h"
#include "CommFSM.h"

#define SPACE_INDEX 47
/*----------------------------- Module Defines ----------------------------*/

#define BALANCED 'n'
#define OFF_BALANCE  'm'

//#define KEYPRESS 1
/*---------------------------- Module Functions ---------------------------*/
/* prototypes for private functions for this machine.They should be functions
   relevant to the behavior of this state machine
*/
static bool initOffBalanceState(void);

/*---------------------------- 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 BalanceState_t CurrentState;
//static GameState_t NextState;
```

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


/*---------------------------- Module Code ----------------------------*/
/****************************************************************************
 Function
     InitBalanceFSM

 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 InitBalanceFSM ( uint8_t Priority )
{
  ES_Event ThisEvent;

  MyPriority = Priority;
  // put us into the Initial PseudoState
  CurrentState = FirstBInit;


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

/****************************************************************************
 Function
     PostBalanceFSM

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

/*****************************************************************************
 Function
    RunBalanceFSM

 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 RunBalanceFSM( ES_Event ThisEvent )
{
  BalanceState_t NextState = CurrentState;
  //GameState_t MainState;

  ES_Event ReturnEvent;
  ES_Event CommEvent;

  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

  switch ( CurrentState )
  {
   // If current state is initial
   case FirstBInit:
          if ( ThisEvent.EventType == ES_INIT){ // only respond to EF_Init
            NextState = InitBState;
          }
          break;

   case InitBState :
          if ( ThisEvent.EventType == ES_Begin) //wait for the MainSM to move into checking
          balance.
          {
               //If B is currently Off Balance
```

```
            bool offBalance = initOffBalanceState();
            if(offBalance == True){
                NextState = BOffBalance;
                puts("Init Balance FSM. Move to: B Off Balance \r");

                CommEvent.EventType =  ES_SEND_REQUEST;
                CommEvent.EventParam = BUnbalancedMessage;
                PostCommFSM(CommEvent);

            }else{
                NextState = BOnBalance;
                puts("Init Balance FSM. Move to: B On Balance \r");

                CommEvent.EventType =  ES_SEND_REQUEST;
                CommEvent.EventParam = BRebalancedMessage;
                PostCommFSM(CommEvent);
            }

        }
         break;

case (BOffBalance):
    if (ThisEvent.EventType == ES_AccessCardRemoved){     //Internal event from comm
        NextState = InitBState;

        puts("Access Card Removed. Move To: Init Balance State \r");

        //game timeout
    }else if  (ThisEvent.EventType == ES_GameEnd) {               //Internal event from comm

         NextState = InitBState;
         puts("Game timeout.  Move to Init Balance State \r");

        // If B goes back on Balance

    }
    #ifdef KEYPRESS
    else if(ThisEvent.EventType == ES_NEW_KEY && ThisEvent.EventParam == BALANCED){
    #endif

    #ifndef KEYPRESS
    else if( ThisEvent.EventType == ES_WentOnBalance){
    #endif

        NextState = BOnBalance;

        CommEvent.EventType =  ES_SEND_REQUEST;
        CommEvent.EventParam = BRebalancedMessage;
        PostCommFSM(CommEvent);

        puts("B on Balance. Move to: B On Balance \r");
    }
    break;
```

```
  case (BOnBalance):

    if (ThisEvent.EventType == ES_AccessCardRemoved){      //Internal event from comm
        NextState = InitBState;

        puts("Access Card Removed. Move To: Init Balance State \r");

    }else if (ThisEvent.EventType == ES_GameEnd) {            //Internal event from comm

        NextState = InitBState;
        puts("Game timeout.  Move to Init Balance State \r");
    }

    #ifdef KEYPRESS
    else if(ThisEvent.EventType == ES_NEW_KEY && ThisEvent.EventParam == OFF_BALANCE){
    #endif

    #ifndef KEYPRESS
    else if( ThisEvent.EventType == ES_WentOffBalance){
    #endif
        //B goes off balance. Tell A
        NextState = BOffBalance;

        CommEvent.EventType =  ES_SEND_REQUEST;
        CommEvent.EventParam = BUnbalancedMessage;
        PostCommFSM(CommEvent);

        puts("B off balance. Move to: B Off Balance \r");

    }

    else if (ThisEvent.EventType == ES_BalanceComplete){               //Internal event
    from comm

        //Post ES_ES_BothRebalanced to AMainSM

        NextState = BSynchroOnBalance;

        puts(" B On balance. Move to: synchro on balance \r");
    }

    break;

  case (BSynchroOnBalance):
    if (ThisEvent.EventType == ES_AccessCardRemoved){           //Internal event from comm
        NextState = InitBState;

        puts("Access Card Removed. Move To: Init Balance State \r");

    }else if (ThisEvent.EventType == ES_GameEnd) {               //Internal event from comm

        NextState = InitBState;
```

```
            puts("Game timeout.  Move to Init Balance State \r");
        }

        #ifdef KEYPRESS
        else if(ThisEvent.EventType == ES_NEW_KEY && ThisEvent.EventParam == OFF_BALANCE){
        #endif

        #ifndef KEYPRESS
        else if( ThisEvent.EventType == ES_WentOffBalance){
        #endif

            // B goes off balance
            NextState = BSynchroOffBalance;

            CommEvent.EventType =  ES_SEND_REQUEST;
            CommEvent.EventParam = BUnbalancedMessage;
            PostCommFSM(CommEvent);

            TurnVibrationOn();

            puts("B off balance. Move to: Synchro Off Balance \r");
        }

        break;

    case (BSynchroOffBalance):
        if (ThisEvent.EventType == ES_AccessCardRemoved){          //Internal event from comm
            NextState = InitBState;

            puts("Access Card Removed. Move To: Init Balance State \r");

        //game timeout
        }else if  (ThisEvent.EventType == ES_GameEnd) {         //Internal event from comm

            NextState = InitBState;
            puts("Game timeout.  Move to Init Balance State \r");

        // If B goes back on Balance
        }

        #ifdef KEYPRESS
        else if(ThisEvent.EventType == ES_NEW_KEY && ThisEvent.EventParam == BALANCED){
        #endif

        #ifndef KEYPRESS
        else if( ThisEvent.EventType == ES_WentOnBalance){
        #endif

            NextState = BSynchroOnBalance;

            CommEvent.EventType =  ES_SEND_REQUEST;
            CommEvent.EventParam = BRebalancedMessage;
            PostCommFSM(CommEvent);
```

```
            TurnVibrationOff();

            puts("B on Balance. Move to: B On Balance \r");
         }
         break;

  }

  CurrentState = NextState;

  return ReturnEvent;
}

/*****************************************************************************
 Function
     QueryTemplateSM

 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, 10/23/11, 19:21
*****************************************************************************/
BalanceState_t QueryBalanceFSM ( void )
{
   return(CurrentState);
}

/*****************************************************************************
 private functions
 *****************************************************************************/
 static bool initOffBalanceState(void){
       unsigned char currentBalanceValue = BOARD_SWITCH_PORT & BOARD_SWITCH_INPUT_PIN;
       ES_Event CommEvent;

       // If Off Balance
       if(currentBalanceValue == BOARD_SWITCH_INPUT_PIN){
            CommEvent.EventType =  ES_SEND_REQUEST;
            CommEvent.EventParam = BUnbalancedMessage;
            PostCommFSM(CommEvent);
            return True;
       }else{
            CommEvent.EventType =  ES_SEND_REQUEST;
            CommEvent.EventParam = BRebalancedMessage;
            PostCommFSM(CommEvent);
```

```
            return False;
        }
    }
```