```
/****************************************************************************
 Module
   ArtilleryFSM.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
 -------------- ---     --------
 02/21/13       DYL      began editing for FAC_FSM
****************************************************************************/
/*----------------------------- 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 <stdio.h>
#include <stdlib.h>
#include <mc9s12e128.h>
#include <S12e128bits.h>
#include <Bin_Const.h>
#include <termio.h>
#include <hidef.h>
#include "S12eVec.h"

#include "E128_PWM.h"         //has all prescale definitions
#include "E128_SPI.h"
#include "E128_Servo.h"
#include "FAC_FSM.h"
#include "NavigationFSM.h"
#include "AlignPPService.h"
#include "DriveTrainService.h"
#include "ArtilleryFSM.h"
#include "StrategyFSM.h"


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


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

/*---------------------------- 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 ArtilleryState_t CurrentState;
static unsigned int NumBalls = 5; //assume we start with 5 balls in hopper
static char ShootingDistance;
static boolean ReceivedFireCommand = False;


// with the introduction of Gen2, we need a module level Priority variable
```

```
static uint8_t MyPriority;

/*---------------------------- Module Code -----------------------------*/
/****************************************************************************
 Function
    InitArtilleryFSM

 Parameters
    uint8_t : the priorty of this service

 Returns
    boolean, 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
    Debbie Li and Ben Sagan 2/20/2013
****************************************************************************/
boolean InitArtilleryFSM ( uint8_t Priority )
{
   ES_Event ThisEvent;

   MyPriority = Priority;
   //servo hardware initialized in BotMain

   // put us into the Off
   CurrentState = Off;

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

/****************************************************************************
 Function
    PostArtilleryFSM

 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
   Debbie Li and Ben Sagan 2/20/2013
```

```
****************************************************************************/
boolean PostArtilleryFSM( ES_Event ThisEvent )
{
   return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
   RunArtilleryFSM

 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
   Debbie Li and Ben Sagan, 2/20/2013
****************************************************************************/
ES_Event RunArtilleryFSM( ES_Event ThisEvent )
{
   ES_Event ReturnEvent;
   ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

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

                         BEGIN STATE MACHINE CODE

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

   switch ( CurrentState )
      {

      case ( Off ) :
         if (ThisEvent.EventType == FIREUP)//StrategyHSM tells you to start up flywheel
            {
               //puts("getting ready to shoot\r\n");
               ShootingDistance = ThisEvent.EventParam;
               //turn on flywheel motor
               RampUpFlywheel(ShootingDistance);

               if (ShootingDistance == LEVIATHAN)
                  {
                     ES_Timer_InitTimer(ARTILLERY_TIMER, FLYWHEEL_TIME);
                  }
               else if (ShootingDistance == SHOOT_PP)
                  {
                     ES_Timer_InitTimer(ARTILLERY_TIMER, PP_TIME);
                  }
               else
                  {
                     ES_Timer_InitTimer(ARTILLERY_TIMER, FLYWHEEL_TIME);
                  }
               CurrentState = RampingUp;
            }
         break; // break Off
```

```
        case ( RampingUp ) :
            if ((ThisEvent.EventType == ES_TIMEOUT) && (ThisEvent.EventParam == ARTILLERY_TIMER))
                //flywheel is up to speed... it takes ~2 second to ramp up from our "idle" state

                    //Set Current state to allow us to fire when necessary
                    CurrentState = WaitingToShoot;
                }

            if ( ThisEvent.EventType == FIRE )
                {
                    puts("Received FIRE Command before ramp up timeout occured \r\n");
                    //ReceivedFireCommand = True;
                    CurrentState = WaitingToShoot;
                }


            break; //break RampingUp

        case ( WaitingToShoot ) :
            if (ThisEvent.EventType == FIRE)
                //get a fire event and Strategy is in the right state to shoot. Note cannon is already up to
speed to be in this state
                {
                    // Got here after the Flywheel was brought up to speed
                    ArtilleryServoShoot();                    //"opens" servo to release ball
                    ES_Timer_InitTimer(ARTILLERY_TIMER, DEPLOY_TIME);     //BALL_DEPLOY timer to
determine when ball deployed, then turn off flywheel
                    //puts("got fire command in waiting2shoot\r\n");
                    CurrentState = Shooting;
                }

            if ((ThisEvent.EventType == ES_TIMEOUT) && (ThisEvent.EventParam == ARTILLERY_TIMER))
                {
                    //Flywheel has completed spinning up, we can now fire.
                    ArtilleryServoShoot();                    //"opens" servo to release ball
                    ES_Timer_InitTimer(ARTILLERY_TIMER, DEPLOY_TIME);     //BALL_DEPLOY timer to
determine when ball deployed, then turn off flywheel
                    puts("got fire command in waiting2shoot\r\n");
                    CurrentState = Shooting;
                }

            if (ThisEvent.EventType == NO_SHOT)
                {
                    RampDownFlywheel();

                    CurrentState = Off;
                }
            break;

        case ( Shooting ) :
            if ((ThisEvent.EventType == ES_TIMEOUT) && (ThisEvent.EventParam == ARTILLERY_TIMER))
                //ball deployed
                {
                    ArtilleryServoLoad();
                    puts("ball deployed\r\n");
                    //turn off flywheel motors
                    RampDownFlywheel();
                    //ES_Timer_InitTimer(ARTILLERY_TIMER, SERVO_TIME);    //SERVO_TIMER to time
when to close servo "door" to stop next ball
```

```
                //tell Strategy that we have fired a ball
                ThisEvent.EventType = BALL_DEPLOYED;
                ThisEvent.EventParam = 0;
                PostStrategyFSM(ThisEvent);

                CurrentState = Off;

            }
        break; //break Shooting
    } // End switch( CurrentState )
  return ReturnEvent;
}
```