```
/*******************************************************************************
 Module
   ShootingSM.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 "ShootingSM.h"

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

static unsigned char Ammo = 5;
```

```
static uint16_t Pitch_Wheel_Target_RPM;


/*---------------------------- 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);
static ES_Event DuringLoaded( ES_Event Event);
static ES_Event DuringFiring( ES_Event Event);

/*---------------------------- Module Variables ----------------------------*/
// everybody needs a state variable, you may need others as well
static TemplateState_t CurrentState;
static unsigned char  Fired_All_Balls_Flag = 0;

/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
    RunShootingSM

 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 RunShootingSM( 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

  #ifdef PRINT_SHOOTING_SM_CALLS
     if(CurrentEvent.EventType != ES_TIMEOUT) //don't spam terminal windows
if this is a routine update/polling event
        {
                printf("RunShootingSM\n\r");
        }

  #endif

   switch ( CurrentState )
   {
       case Out_of_Balls :       // If current state is state one
        { // Execute During function for state one. ES_ENTRY & ES_EXIT are
```

```
          // processed here allow the lowere level state machines to re-map
          // or consume the event
          CurrentEvent = DuringOut_of_Balls(CurrentEvent);
          //process any events

          #ifdef PRINT_SHOOTING_SM_STATES
          if(CurrentEvent.EventType != ES_TIMEOUT) //don't spam terminal
windows if this is a routine update/polling event
              {
             printf("Out_of_Balls\n\r");
                 }

          #endif

          if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
          {
             switch (CurrentEvent.EventType)
             {
                default : //If event is event one
                  { // Execute action function for state one : event one
                   //NextState = Out_of_Balls;//Decide what the next state
will be
                   // for internal transitions, skip changing MakeTransition
                   //MakeTransition = false; //mark that we are taking a
transition
                   // if transitioning to a state with history change kind of
entry
                   //EntryEventKind.EventType = ES_ENTRY_HISTORY;
                                 //EntryEventKind.EventType = ES_ENTRY;
                   // optionally, consume or re-map this event for the upper
                   // level state machine
                   //ReturnEvent = CurrentEvent;
                   }
                   break;
                 // repeat cases as required for relevant events
             }
          }
          }
          break;
       // repeat state pattern as required for other states

           case Loaded :       // If current state is state one
         {  // Execute During function for state one. ES_ENTRY & ES_EXIT are
          // processed here allow the lowere level state machines to re-map
          // or consume the event
          CurrentEvent = DuringLoaded(CurrentEvent);
          //process any events

          #ifdef PRINT_SHOOTING_SM_STATES
              if(CurrentEvent.EventType != ES_TIMEOUT) //don't spam terminal
windows if this is a routine update/polling event
              {
             printf("Loaded\n\r");
                 }

          #endif
```

```
        if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
        {

           switch (CurrentEvent.EventType)
           {

              case Fire_Ball : //If event is event one
               {  // Execute action function for state one : event one

                                 //Keep track of remaining balls to fire
                                 if(Ammo != 0)
                                 {
                                   Ammo = Ammo - 1;

                                  }


                                 NextState = Firing;//Decide what the next
state will be
                  // for internal transitions, skip changing MakeTransition

                                 //Close loading gate and open firing gate
                                 UpdateServo(LOADING_SERVO_CHANNEL,
LOADING_SERVO_CLOSED_POSITION);
                                 UpdateServo(FIRING_SERVO_CHANNEL,
FIRING_SERVO_OPEN_POSITION);

                                 //Set timer
                                 ES_Timer_SetTimer(GATE_TIMER,
GATE_UP_INTERVAL_MS);
                                 ES_Timer_StartTimer(GATE_TIMER);

//ES_Timer_InitTimer(GATE_TIMER,GATE_UP_INTERVAL_MS);


                   MakeTransition = true; //mark that we are taking a
transition
                   // if transitioning to a state with history change kind of
entry
                   //EntryEventKind.EventType = ES_ENTRY_HISTORY;
                                 EntryEventKind.EventType = ES_ENTRY;
                   // optionally, consume or re-map this event for the upper
                   // level state machine
                   ReturnEvent.EventType = ES_NO_EVENT;
                   }
                   break;
                  // repeat cases as required for relevant events

                               case Fire_All_Balls : //If event is event one
                  {  // Execute action function for state one : event one

                                 NextState = Firing;//Decide what the next
state will be
                   // for internal transitions, skip changing MakeTransition

                                 //Open both the loading gate and the firing
gate
```

```
                                    UpdateServo(LOADING_SERVO_CHANNEL,
LOADING_SERVO_OPEN_POSITION);
                                    UpdateServo(FIRING_SERVO_CHANNEL,
FIRING_SERVO_OPEN_POSITION);

                                    //Set timer
                                    ES_Timer_SetTimer(GATE_TIMER,
2*Ammo*GATE_UP_INTERVAL_MS);
                                    ES_Timer_StartTimer(GATE_TIMER);

                                    Fired_All_Balls_Flag = 1;


                    MakeTransition = true; //mark that we are taking a
transition
                    // if transitioning to a state with history change kind of
entry
                    //EntryEventKind.EventType = ES_ENTRY_HISTORY;
                                    EntryEventKind.EventType = ES_ENTRY;
                    // optionally, consume or re-map this event for the upper
                    // level state machine
                    ReturnEvent.EventType = ES_NO_EVENT;
                    }
                    break;
                  // repeat cases as required for relevant events

                                    default : //If event is event one
                   { // Execute action function for state one : event one
                    //NextState = Loaded;//Decide what the next state will be
                    // for internal transitions, skip changing MakeTransition
                    //MakeTransition = false; //mark that we are taking a
transition
                    // if transitioning to a state with history change kind of
entry
                    //EntryEventKind.EventType = ES_ENTRY_HISTORY;
                                    //EntryEventKind.EventType = ES_ENTRY;
                    // optionally, consume or re-map this event for the upper
                    // level state machine
                    //ReturnEvent = CurrentEvent;
                    }
                    break;
                  // repeat cases as required for relevant events
              }
          }
        }
       break;

                case Firing:       // If current state is state one
          {// Execute During function for state one. ES_ENTRY & ES_EXIT are
          // processed here allow the lowere level state machines to re-map
          // or consume the event
          CurrentEvent = DuringFiring(CurrentEvent);

          #ifdef PRINT_SHOOTING_SM_STATES
                if(CurrentEvent.EventType != ES_TIMEOUT) //don't spam terminal
windows if this is a routine update/polling event
              {
```

```
            printf("Firing\n\r");
                }

        #endif

        //process any events
        if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
        {
           switch (CurrentEvent.EventType)
           {
              case ES_TIMEOUT : //If event is event one
                           {
                           if(CurrentEvent.EventParam == GATE_TIMER)
                           {

                                       //if last firing event release all the
balls in one fell swoop...
                                       if(Fired_All_Balls_Flag == 1)
                                       {
                                               Fired_All_Balls_Flag == 0;
//clear flag
                                               Ammo = 0; //update ammo count
                                       }


                  // Execute action function for state one : event one
                                   if(Ammo == 0)
                                   {
                                               NextState = Loaded;//Decide what
the next state will be
                                   }
                                   else
                                   {
                                               NextState = Out_of_Balls;
                                   }

                                   //Close firing gate and open loading gate
                                   UpdateServo(FIRING_SERVO_CHANNEL,
FIRING_SERVO_CLOSED_POSITION);
                                   UpdateServo(LOADING_SERVO_CHANNEL,
LOADING_SERVO_OPEN_POSITION);



                  // for internal transitions, skip changing MakeTransition
                  MakeTransition = true; //mark that we are taking a
transition
                  // if transitioning to a state with history change kind of
entry
                  //EntryEventKind.EventType = ES_ENTRY_HISTORY;
                                   EntryEventKind.EventType = ES_ENTRY;
                  // optionally, consume or re-map this event for the upper
                  // level state machine
                  ReturnEvent.EventType = ES_NO_EVENT;
                                   }

                                   else
```

```
                                {
                                // Execute action function for state one :
event one
                    NextState = Firing;//Decide what the next state will be
                    // for internal transitions, skip changing MakeTransition
                    MakeTransition = false; //mark that we are taking a
transition
                    // if transitioning to a state with history change kind of
entry
                    //EntryEventKind.EventType = ES_ENTRY_HISTORY;
                                EntryEventKind.EventType = ES_ENTRY;
                    // optionally, consume or re-map this event for the upper
                    // level state machine
                    ReturnEvent = CurrentEvent;
                                }
                                }
                    break;
                  // repeat cases as required for relevant events

                              default : //If event is event one
                   { // Execute action function for state one : event one

                    //NextState = Firing;//Decide what the next state will be
                    // for internal transitions, skip changing MakeTransition
                    //MakeTransition = false; //mark that we are taking a
transition
                    // if transitioning to a state with history change kind of
entry
                    //EntryEventKind.EventType = ES_ENTRY_HISTORY;
                                //EntryEventKind.EventType = ES_ENTRY;
                    // optionally, consume or re-map this event for the upper
                    // level state machine
                    //ReturnEvent = CurrentEvent;
                    }
                    break;
                  // repeat cases as required for relevant events
              }

          }
          //break;

    }
    break;

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

       CurrentState = NextState; //Modify state variable

       //   Execute entry function for new state
       // this defaults to ES_ENTRY
       RunShootingSM(EntryEventKind);
```

```
    }

    CurrentState = NextState;

    return(ReturnEvent);
}


/*****************************************************************************
 Function
    StartShootingSM

 Parameters
    None

 Returns
    None

 Description
    Does any required initialization for this state machine
 Notes

 Author
    J. Edward Carryer, 2/18/99, 10:38AM
*****************************************************************************/
void StartShootingSM ( ES_Event CurrentEvent )
{
   // local variable to get debugger to display the value of CurrentEvent
   ES_Event LocalEvent = CurrentEvent;

  #ifdef PRINT_SHOOTING_SM_CALLS
        if(CurrentEvent.EventType != ES_TIMEOUT) //don't spam terminal
windows if this is a routine update/polling event
        {
               printf("StartShootingSM\n\r");
       }

  #endif

   // 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
   if ( ES_ENTRY_HISTORY != CurrentEvent.EventType )
   {
        if(Ammo != 0)
               {
                       CurrentState = Loaded;
               }
               else
               {
                       CurrentState = Out_of_Balls;
               }
   }

   //Start ServoServiceSM now so that we can call servo functions
   //StartServoServiceSM(CurrentEvent);
```

```
   //Ensure that ball servo gates are initialized to proper position:
   //Close firing gate and open loading gate
        UpdateServo(FIRING_SERVO_CHANNEL, FIRING_SERVO_CLOSED_POSITION);
        UpdateServo(LOADING_SERVO_CHANNEL, LOADING_SERVO_OPEN_POSITION);

   //Start spinning the pitching wheel:

   DDRU |= BIT2HI; //set PTU2 as output for pitching wheel PWM
   PTU &= BIT2LO; //initialize with wheel stopped

        //Configure PWM channel 2
        PWMCLK &= ~_S12_PCLK2; //Selected pre-scaled clock B as time base
    PWMPRCLK |= _S12_PCKB2; //Divide system clock by 16
    PWMPRCLK &= (~_S12_PCKB1)&(~_S12_PCKB0);
    PWMCAE &= ~_S12_CAE2;  // left align PWM channel 2
    PWMPOL |= _S12_PPOL2; //set polarity to active high for channel 2
        PWMPER2 = PITCH_MOTOR_PWM_PERIOD; //Establish period for 10 kHZ PWM
frequency
        PWMDTY2 = 0; //Initialize with duty cycle set to 0
        PWME |= _S12_PWME2; //Enable PWM channel 2

        MODRR |= _S12_MODRR2; //Give control of Port U2 to PWM subsystem
        Pitch_Wheel_Target_RPM = DEFAULT_PITCH_WHEEL_RPM; //Initialize PI
control target

        //Enable pitch wheel motor PI loop
        TIM2_TSCR2 |= (_S12_PR2); //Configure clock scaler to divide system
clock by 16
        TIM2_TSCR2 &= (~_S12_PR1)&(~_S12_PR0);
        TIM2_TIOS |= _S12_IOS6; //configure Timer 2, channel 6 as output
compare
        TIM2_TCTL1 &= (~_S12_OM5)&(_S12_OL5); //configure Timer 2, channel 6
to leave pin disconnected
        TIM2_TC6 = TIM2_TCNT + PITCH_MOTOR_CONTROL_INTERVAL; //initialize
output compare register
        TIM2_TFLG1 = _S12_C6F; //clear interrupt flags
        //TIM2_TIE |= _S12_C6I; //enable interrupts for channel 6
        TIM2_TSCR1 |= _S12_TEN; //ensure that timers are enabled
        EnableInterrupts; //ensure that interrupts are enabled

        #ifdef FIRING_TEST_HARNESS

            //Start pitching motor if PI control is not yet implemented
            PWMDTY2 = PITCH_MOTOR_PWM_DTY;


        #endif

   // call the entry function (if any) for the ENTRY_STATE
   RunShootingSM(CurrentEvent);
}

void StopShootingSM ( ES_Event CurrentEvent )
{
   // local variable to get debugger to display the value of CurrentEvent
   ES_Event LocalEvent = CurrentEvent;
```

```
  #ifdef PRINT_SHOOTING_SM_CALLS
            if(CurrentEvent.EventType != ES_TIMEOUT) //don't spam terminal
windows if this is a routine update/polling event
              {
                                printf("StopShootingSM\n\r");
                }

  #endif

    //Ensure that ball servo gates are returned to proper position:
    //Close firing gate and open loading gate
        UpdateServo(FIRING_SERVO_CHANNEL, FIRING_SERVO_CLOSED_POSITION);
        UpdateServo(LOADING_SERVO_CHANNEL, LOADING_SERVO_OPEN_POSITION);

   //Stop spinning the pitching wheel:

        //Configure PWM channel 2
        PWMDTY2 = 0; //Kill duty cycle
        Pitch_Wheel_Target_RPM = 0; //Kill control loop

        //Disable pitch wheel motor PI loop
        TIM2_TFLG1 = _S12_C6F; //clear interrupt flags
        TIM2_TIE &= ~_S12_C6I; //disable interrupts for channel 6

   // call the entry function (if any) for the ENTRY_STATE
   RunShootingSM(CurrentEvent);
}

/****************************************************************************
 Function
     QueryShootingSM

 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 QueryShootingSM ( void )
{
  #ifdef PRINT_SHOOTING_SM_CALLS

    printf("QueryShootingSM\n\r");

  #endif

   return(CurrentState);

}
```

```
unsigned char Querry_Ammo(void)
{

 #ifdef PRINT_SHOOTING_SM_CALLS

    printf("Query_Ammo\n\r");

  #endif

        return Ammo;

}

void Add_Ammo(unsigned char new_balls)
{

        unsigned char updated_ammo_count;

        #ifdef PRINT_SHOOTING_SM_CALLS

    printf("Add_Ammo\n\r");

  #endif

        updated_ammo_count = Ammo + new_balls;

        if(updated_ammo_count > 5)
        {
                updated_ammo_count = 5;
        }

        Ammo = updated_ammo_count;

}

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

static ES_Event DuringOut_of_Balls( ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption
        ES_Event PostEvent;

  #ifdef PRINT_SHOOTING_SM_CALLS
     if(Event.EventType != ES_TIMEOUT) //don't spam terminal windows if this
is a routine update/polling event
         {
    printf("DuringOut_of_Balls\n\r");
        }

  #endif

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if ( (Event.EventType == ES_ENTRY) ||
         (Event.EventType == ES_ENTRY_HISTORY) )
```

```
    {
        //Upon entry into any state within the shooting state machine, notify
the MasterHSM if there are no balls left to shoot
                if(Ammo == 0)
                {
                        PostEvent.EventType = Out_of_Ammo;
                        PostMasterHSM(PostEvent);
                }
                // implement any entry actions required for this state machine
        // after that start any lower level machines that run in this state
        //StartLowerLevelSM( Event );
        // repeat the StartxxxSM() functions for concurrent state machines
        // on the lower level
    }
    else if ( Event.EventType == ES_EXIT )
    {
        // on exit, give the lower levels a chance to clean up first
        //RunLowerLevelSM(Event);
        // repeat for any concurrently running state machines
        // now do any local exit functionality
    }else
    // do the 'during' function for this state
    {
        // run any lower level state machine
        //ReturnEvent = RunLowerLevelSM(Event);
        // repeat for any concurrent lower level machines
        // do any activity that is repeated as long as we are in this state



    }
    // return either Event, if you don't want to allow the lower level
machine
    // to remap the current event, or ReturnEvent if you do want to allow it.
    return(ReturnEvent);
}

static ES_Event DuringLoaded( ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption
       ES_Event PostEvent;

  #ifdef PRINT_SHOOTING_SM_CALLS
     if(Event.EventType != ES_TIMEOUT) //don't spam terminal windows if this
is a routine update/polling event
        {
    printf("DuringLoaded\n\r");
        }

  #endif

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if ( (Event.EventType == ES_ENTRY) ||
         (Event.EventType == ES_ENTRY_HISTORY) )
    {
```

```
                //Upon entry into any state within the shooting state machine,
notify the MasterHSM if there are no balls left to shoot

                if(Ammo == 0)
                {
                        PostEvent.EventType = Out_of_Ammo;
                        PostMasterHSM(PostEvent);
                }
                // implement any entry actions required for this state machine
        // after that start any lower level machines that run in this state
        //StartLowerLevelSM( Event );
        // repeat the StartxxxSM() functions for concurrent state machines
        // on the lower level
    }
    else if ( Event.EventType == ES_EXIT )
    {
        // on exit, give the lower levels a chance to clean up first
        //RunLowerLevelSM(Event);
        // repeat for any concurrently running state machines
        // now do any local exit functionality
    }else
    // do the 'during' function for this state
    {
        // run any lower level state machine
        //ReturnEvent = RunLowerLevelSM(Event);
        // repeat for any concurrent lower level machines
        // do any activity that is repeated as long as we are in this state



    }
    // return either Event, if you don't want to allow the lower level
machine
    // to remap the current event, or ReturnEvent if you do want to allow it.
    return(ReturnEvent);
}

static ES_Event DuringFiring( ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption
    ES_Event PostEvent;

   #ifdef PRINT_SHOOTING_SM_CALLS

             if(Event.EventType != ES_TIMEOUT) //don't spam terminal windows
if this is a routine update/polling event
            {
    printf("DuringFiring\n\r");
        }

  #endif

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if ( (Event.EventType == ES_ENTRY) ||
         (Event.EventType == ES_ENTRY_HISTORY) )
    {
```

```
                //Upon entry into any state within the shooting state machine,
notify the MasterHSM if there are no balls left to shoot
                if(Ammo == 0)
                {
                        PostEvent.EventType = Out_of_Ammo;
                        PostMasterHSM(PostEvent);
                }
                // implement any entry actions required for this state machine
        // after that start any lower level machines that run in this state
        //StartLowerLevelSM( Event );
        // repeat the StartxxxSM() functions for concurrent state machines
        // on the lower level
    }
    else if ( Event.EventType == ES_EXIT )
    {
        // on exit, give the lower levels a chance to clean up first
        //RunLowerLevelSM(Event);
        // repeat for any concurrently running state machines
        // now do any local exit functionality
    }else
    // do the 'during' function for this state
    {
        // run any lower level state machine
        //ReturnEvent = RunLowerLevelSM(Event);
        // repeat for any concurrent lower level machines
        // do any activity that is repeated as long as we are in this state



    }
    // return either Event, if you don't want to allow the lower level
machine
    // to remap the current event, or ReturnEvent if you do want to allow it.
    return(ReturnEvent);
}

/*

                duty_cycle = (unsigned char)( ((float)current_pot_input*
                PITCH_MOTOR_PWM_PERIOD)/1024 );

                if(duty_cycle > PITCH_MOTOR_PWM_PERIOD)
                {
                duty_cycle = PITCH_MOTOR_PWM_PERIOD;
                }
                else if(duty_cycle <
(MIN_DUTY_CYCLE_FRACTION*PITCH_MOTOR_PWM_PERIOD))
                {
                        duty_cycle = (unsigned
char)(MIN_DUTY_CYCLE_FRACTION*PITCH_MOTOR_PWM_PERIOD);
                }


                duty_cycle_percent = (unsigned
char)(((float)duty_cycle)*100/PITCH_MOTOR_PWM_PERIOD);

                printf("Duty Cycle: %d %\n\r", duty_cycle_percent);
```

```
                PWMDTY2 = duty_cycle;
*/
```