```
/*******************************************************************************
 Module
   BallRequestSM.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 "BallRequestSM.h"

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

/*---------------------------- 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 DuringPulsing( ES_Event Event);
static ES_Event DuringNot_Pulsing( ES_Event Event);
static ES_Event DuringFull_of_Ammo( ES_Event Event);

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

static unsigned char ball_index = 1;
static unsigned char request_ball_flag = 0;
static unsigned char last_pulse = 0;

/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
    RunBallRequestSM

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

  #endif

   switch ( CurrentState )
   {
       case Pulsing :       // 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 = DuringPulsing(CurrentEvent);
          //process any events

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

          #endif

          if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
          {
             switch (CurrentEvent.EventType)
             {
                case ES_ENTRY :
                            {
                                        if (Querry_Ammo() < 5) //if not fully
loaded
                                        {
                                                request_ball_flag = 1; //signal
ISR to start IR pulse sequence
                                        }
                                        else
                                        {

                                                NextState = Full_of_Ammo;

                                                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;

                                case Request_Sent:
                                {

                                        if(Querry_Ammo() < 5)
                                        {
                                                NextState = Not_Pulsing;
                                        }
                                        else
```

```
                                        {
                                                NextState = Full_of_Ammo;
                                        }

                                    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;

                                case ES_EXIT :
                            {

                                }
                                break;


                                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 Not_Pulsing :       // 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 = DuringNot_Pulsing(CurrentEvent);
            //process any events

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

```
                        printf("Not_Pulsing\n\r");

                }

        #endif

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

           switch (CurrentEvent.EventType)
           {


                           case ES_ENTRY:
                           {
                                //Start_Timer
                                //Set timer for next read of calibration
switches
                                ES_Timer_SetTimer(BALL_REQUEST_TIMER,
BALL_REQUEST_INTERVAL_MS);
                                ES_Timer_StartTimer(BALL_REQUEST_TIMER);
                                }

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

                                  if(CurrentEvent.EventParam ==
BALL_REQUEST_TIMER)
                                  {
                                                NextState = Pulsing;

                                                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 Full_of_Ammo:       // 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 = DuringFull_of_Ammo(CurrentEvent);

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

          #endif

          //process any events
          if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
          {
             switch (CurrentEvent.EventType)
             {

                               default : //If event is 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;
       RunBallRequestSM(CurrentEvent);

       CurrentState = NextState; //Modify state variable

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

     CurrentState = NextState;

     return(ReturnEvent);
}


/*****************************************************************************
 Function
     StartBallRequestSM

 Parameters
     None

 Returns
     None

 Description
     Does any required initialization for this state machine
 Notes

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

  #ifdef PRINT_BALL_REQUEST_SM_CALLS
    if(CurrentEvent.EventType != ES_TIMEOUT) //don't spam terminal windows if
this is a routine update/polling event
        {
                printf("StartBallRequestSM\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(Querry_Ammo() < 5)
               {
                      CurrentState = Pulsing;
               }
               else
               {
                      CurrentState = Full_of_Ammo;
               }
   }

        //Configure ports T4-5 as outputs
        DDRT |= BIT4HI|BIT5HI;

        //Initialize with ports T4-T5 low
        PTT &= BIT4LO&BIT5LO;

        //Turn visual indicator light on before starting pulse sequence
        PTT |= BIT5HI;

        //Configure Timer 1, Channel 4 as an output compare for timing IR
communication pulses with ball depot
        TIM1_TSCR2 |= (_S12_PR2)|(_S12_PR1)|(_S12_PR0); //scale clock to
divide 24 MHz system clock by 128
        TIM1_TIOS |= _S12_IOS4; //configure channel 4 as output compare
        TIM1_TCTL1 &= (~_S12_OM4)&(~_S12_OL4); //configure channel 4 to leave
pin disconnected
        TIM1_TC4 = TIM1_TCNT + LOW_PULSE_INTERVAL; //initialize output compare
register
        TIM1_TFLG1 = _S12_C4F; //clear interrupt flags
        TIM1_TIE |= _S12_C4I; //enable interrupts for channel 4
        TIM1_TSCR1 |= _S12_TEN; //enable timer

        EnableInterrupts; //enable interrupts



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

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

        //Turn visual indicator light off after leaving state
        PTT &= BIT5LO;

  #ifdef PRINT_BALL_REQUEST_SM_CALLS

               printf("StopBallRequestSM\n\r");
  #endif
```

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

/****************************************************************************
 Function
     QueryBallRequestSM

 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 QueryBallRequestSM ( void )
{
  #ifdef PRINT_BALL_REQUEST_SM_CALLS

    printf("QueryBallRequestSM\n\r");

  #endif

   return(CurrentState);
}

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

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

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

  #endif

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if ( (Event.EventType == ES_ENTRY) ||
         (Event.EventType == ES_ENTRY_HISTORY) )
    {
        // 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 DuringNot_Pulsing( ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption

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

  #endif

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if ( (Event.EventType == ES_ENTRY) ||
         (Event.EventType == ES_ENTRY_HISTORY) )
    {
        // 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 DuringFull_of_Ammo( ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption
    ES_Event PostEvent;

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

  #endif

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if ( (Event.EventType == ES_ENTRY) ||
         (Event.EventType == ES_ENTRY_HISTORY) )
    {
        PostEvent.EventType = Reload_Complete;
               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);
}

void interrupt _Vec_tim1ch4 RequestBall(void)
{
                ES_Event PostEvent;

                TIM1_TFLG1 = _S12_C4F; //clear interrupt flags

                EnableInterrupts; //enable interrupts

                if (request_ball_flag == 1) //start pulsing if the request
ball flag is set
                {
                                if(last_pulse == 0) //toggle the IR pulse
                                {
                                        PTT |= BIT4HI; //shine IR light

                                        TIM1_TC4 += HIGH_PULSE_INTERVAL;
//increment timer

                                        last_pulse = 1; //store last pulse

                                }

                                else if (ball_index < 10)
                                {
                                        PTT &= BIT4LO; //turn off IR light

                                        TIM1_TC4 += LOW_PULSE_INTERVAL;
//increment timer

                                        ball_index ++; //increment pulse
counter

                                        last_pulse = 0; //store last pulse

                                }

                                else
                                {
                                        PTT &= BIT4LO; //turn off IR light

                                        TIM1_TC4 += LOW_PULSE_INTERVAL;

                                        ball_index = 1; //reset ball index

                                        request_ball_flag = 0; //clear ball
request flag

                                        last_pulse = 0; //store last pulse

                                        PostEvent.EventType = Request_Sent;
```

```
                                        PostMasterHSM(PostEvent); //Post event
to state machine indicating that ball request transmission is complete

                                        Add_Ammo(1); //make note of the number
of balls that we've added

                                    }

                }

}
```