```
/*******************************************************************************
 Module
   JoustingSM.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 "ServoService.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 "JoustingSM.h"

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

```
#ifdef JOUSTING_TEST_HARNESS

static unsigned char Button_State = 0;

//static long LANCE_SERVO_DOWN_POSITION;
static short last_pot_input;

#endif

/*---------------------------- 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 DuringLance_Up( ES_Event Event);
static ES_Event DuringLance_Down( ES_Event Event);

/*---------------------------- Module Variables ----------------------------*/
// everybody needs a state variable, you may need others as well
static TemplateState_t CurrentState;
static cleared_to_lance_flag = 1;

/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
    RunJoustingSM

 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 RunJoustingSM( 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_JOUSTING_SM_CALLS

    printf("RunJoustingSM\n\r");

  #endif

   switch ( CurrentState )
```

```
   {
       case Lance_Up :       // 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 = DuringLance_Up(CurrentEvent);
         //process any events

         #ifdef PRINT_JOUSTING_SM_STATES

            printf("Lance_Up\n\r");

         #endif

            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 == LANCE_TIMER)
               {


                cleared_to_lance_flag = 1;


                  }


                  }
                               break;



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

                              if(cleared_to_lance_flag == 1)
                              {
                NextState = Lance_Down;//Decide what the next state will be
                // for internal transitions, skip changing MakeTransition

                             if(get_mode() != 1)
                             {
                //Close loading gate and open firing gate
                UpdateServo(LANCE_SERVO_CHANNEL,
LANCE_SERVO_DOWN_POSITION);

                               }

                //Set timer
```

```
                ES_Timer_SetTimer(LANCE_TIMER, LANCE_DOWN_INTERVAL_MS);
                ES_Timer_StartTimer(LANCE_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;

                              cleared_to_lance_flag = 0;
                              }
                }
                break;
              // repeat cases as required for relevant events

              // repeat cases as required for relevant events
           }
        }
        }
        break;
     // repeat state pattern as required for other states
 case Lance_Down :       // 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 = DuringLance_Down(CurrentEvent);
        //process any events

        #ifdef PRINT_JOUSTING_SM_STATES

           printf("Lance_Down\n\r");

        #endif

           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 == LANCE_TIMER)
              {

                      NextState = Lance_Up;

                //Raise Lance
                UpdateServo(LANCE_SERVO_CHANNEL, LANCE_SERVO_UP_POSITION);
```

```
                                ES_Timer_SetTimer(LANCE_TIMER,
LANCE_UP_INTERVAL_MS);
                               ES_Timer_StartTimer(LANCE_TIMER);

                  // 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 = Lance_Down;//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;
                   #ifdef PRINT_JOUSTING_SM_STATES

                   printf("Timer parameter error\n\r");

                   #endif

                  }
                  }
                               break;

                       case RaiseLance : //If event is event one
               {

                        NextState = Lance_Up;

                  //Raise Lance
                  UpdateServo(LANCE_SERVO_CHANNEL, LANCE_SERVO_UP_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;


                    }
                                break;
                    }
                    }
                    }
                    break;

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

       CurrentState = NextState; //Modify state variable

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

     CurrentState = NextState;

     return(ReturnEvent);
}


/*****************************************************************************
 Function
     StartJoustingSM

 Parameters
     None

 Returns
     None

 Description
     Does any required initialization for this state machine
 Notes

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

  #ifdef PRINT_JOUSTING_SM_CALLS

    printf("StartJoustingSM\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 )
   {

            CurrentState = Lance_Up;
                       cleared_to_lance_flag = 1;

   }

   //Ensure that ball servo gates are initialized to proper position:
   //Close firing gate and open loading gate
    UpdateServo(LANCE_SERVO_CHANNEL, LANCE_SERVO_UP_POSITION);

   //Start spinning the pitching wheel:


    #ifdef JOUSTING_TEST_HARNESS
        //init lance servo up position before it has been set by pot

        // Configure Timer 1, Channel 5 to provide a polling interval to
check for
        //new input commands during servo calibration
        TIM1_TSCR2 |= (_S12_PR2)|(_S12_PR1)|(_S12_PR0);

        // Configure Channel 5 as output compare
        TIM1_TIOS |= _S12_IOS5;

        // Configure Channel 5 to leave pin disconnected
        TIM1_TCTL1 &= (~_S12_OM5)&(~_S12_OL5);

        // Initialize output compare register
        TIM1_TC5 = TIM1_TCNT + JOUST_UPDATE_INTERVAL;

        // Clear interrupt flags
        TIM1_TFLG1 = _S12_C5F;

        // Enable interrupts for channel 5
        TIM1_TIE |= _S12_C5I;

        // Enable Timer
        TIM1_TSCR1 |= _S12_TEN;

        //Initialize Port AD0 to be an analog input port
        ADS12_Init("AAAAAAAA");

        //Initialize last_Pot_Value to present value of Port AD0
        last_pot_input = ADS12_ReadADPin(0);

    #endif

   // call the entry function (if any) for the ENTRY_STATE
```

```
   RunJoustingSM(CurrentEvent);
}

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

  #ifdef PRINT_JOUSTING_SM_CALLS

    printf("StopJoustingSM\n\r");

  #endif

    //Ensure that ball servo gates are returned to proper position:
    //Close firing gate and open loading gate
    UpdateServo(LANCE_SERVO_CHANNEL, LANCE_SERVO_UP_POSITION);
        cleared_to_lance_flag = 1;


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

/*****************************************************************************
 Function
     QueryJoustingSM

 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 QueryJoustingSM ( void )
{
  #ifdef PRINT_JOUSTING_SM_CALLS

    printf("QueryJoustingSM\n\r");

  #endif

   return(CurrentState);
}

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

static ES_Event DuringLance_Up( ES_Event Event)
```

```
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption

  #ifdef PRINT_JOUSTING_SM_CALLS

    printf("DuringLanceUp\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 DuringLance_Down( ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption

  #ifdef PRINT_JOUSTING_SM_CALLS

    printf("DuringLanceDown\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);
}



#ifdef JOUSTING_TEST_HARNESS

    //poll pushbutton input for fire ball command
    void interrupt  _Vec_tim1ch5 PollLanceButton(void)
    {

        ES_Event ThisEvent;

        unsigned char Button_Value;

        short current_pot_input;

        // clear flag
        TIM1_TFLG1 = _S12_C5F;

        EnableInterrupts;

        Button_Value = ((PORTE & BIT1HI) == BIT1HI);
        printf("button value: %d\n\r", Button_Value);
        if(Button_Value == 0)
        {
```

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

                 case 1:
                 {
                    Button_State = 2;

                    ThisEvent.EventType = LowerLance;
                    PostJoustingHSM(ThisEvent);

                    printf("Drop Lance!\n\r");
                 }
                 break;

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

    current_pot_input = ADS12_ReadADPin(JOUSTING_TEST_POT_PIN);
   //printf("pot value: %d\n\r", current_pot_input);

    /*if(abs(current_pot_input - last_pot_input) > 10)
    {

         LANCE_SERVO_UP_POSITION = ( ((float)current_pot_input*
        1990)/1024 );
        printf("Lance servo up value: %d\n\r", LANCE_SERVO_UP_POSITION);
    }

    last_pot_input = current_pot_input;
     */
    TIM1_TC5 = TIM1_TCNT + JOUST_UPDATE_INTERVAL;
}

#endif
```