```
/*******************************************************************************
 Module
   DrivingHSM.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 "DrivingHSM.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 DuringChargingFieldFwdState( ES_Event Event);
static ES_Event DuringChargingFieldRvsState( ES_Event Event);
static ES_Event DuringFindingWallFwdState( ES_Event Event);
static ES_Event DuringFindingWallRvsState( ES_Event Event);
static ES_Event DuringStopped( ES_Event Event);
//static ES_Event DuringGoingToStartFwdState( ES_Event Event);
//static ES_Event DuringGoingToStartRvsState( ES_Event Event);

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

/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
    RunDrivingHSM

 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 RunDrivingHSM( 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
   ES_Event PostEvent;

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

  #endif

   switch ( CurrentState )
   {
       case ChargingFieldFwdState :       // 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 = DuringChargingFieldFwdState(CurrentEvent);
         //process any events

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

         #endif

         if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
         {
            switch (CurrentEvent.EventType)
            {
               /*case Crossed_Offset_Nominal :
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS


        printf("Crossed_Offset_Nominal\n\r");

                                        #endif

                                        PostEvent.EventType = ChargingFieldFwd;
                                        PostMotorService(PostEvent);
                                }
                                break;

                                case Crossed_Offset_Min :
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS


        printf("Crossed_Offset_Min\n\r");

                                        #endif

                                        //PostEvent.EventType = VeerRightFwd;
                                        //PostMotorService(PostEvent);
                                }
                                break;

                                case Crossed_Offset_Max :
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS


        printf("Crossed_Offset_Max\n\r");

                                        #endif

                                        //PostEvent.EventType = VeerLeftFwd;
```

```
        //PostMotorService(PostEvent);
}
break;*/

case found_goal :
{
        #ifdef PRINT_DRIVING_HSM_EVENTS

                printf("found_goal\n\r");

        #endif

        PostEvent.EventType = LowerLance;
        PostMasterHSM(PostEvent);

}
break;

case found_knight_fwd :
{
        #ifdef PRINT_DRIVING_HSM_EVENTS

                printf("found_knight_fwd\n\r");

        #endif

        PostEvent.EventType = LowerLance;
        PostMasterHSM(PostEvent);

}
break;

case lost_knight_fwd :
{
        #ifdef PRINT_DRIVING_HSM_EVENTS

                printf("lost_knight_fwd\n\r");

        #endif

        PostEvent.EventType = LowerLance;
        PostMasterHSM(PostEvent);

}
break;

case found_knight_rvs :
{
        #ifdef PRINT_DRIVING_HSM_EVENTS

                printf("found_knight_rvs\n\r");

        #endif

        PostEvent.EventType = LowerLance;
        PostMasterHSM(PostEvent);
```

```
                                }
                                break;

                                case Crossed_Slow_Down_Fwd_Offset :
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS


        printf("Crossed_Slow_Down_Fwd_Offset\n\r");

                                        #endif

                                        PostEvent.EventType = FindingWallFwd;
                                        PostMotorService(PostEvent);

                                        NextState = FindingWallFwdState;
                                        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;
      // repeat state pattern as required for other states

            case ChargingFieldRvsState :       // 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 = DuringChargingFieldRvsState(CurrentEvent);
         //process any events

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

         #endif

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

```
        /*case Crossed_Offset_Nominal :
                        {
                                #ifdef PRINT_DRIVING_HSM_EVENTS


printf("Crossed_Offset_Nominal\n\r");

                                #endif

                                PostEvent.EventType = ChargingFieldRvs;
                                PostMotorService(PostEvent);
                        }
                        break;

                        case Crossed_Offset_Min :
                        {
                                #ifdef PRINT_DRIVING_HSM_EVENTS


printf("Crossed_Offset_Min\n\r");

                                #endif

                                //PostEvent.EventType = VeerRightRvs;
                                //PostMotorService(PostEvent);
                        }
                        break;

                        case Crossed_Offset_Max :
                        {
                                #ifdef PRINT_DRIVING_HSM_EVENTS


printf("Crossed_Offset_Max\n\r");

                                #endif

                                //PostEvent.EventType = VeerLeftRvs;
                                //PostMotorService(PostEvent);
                        }
                        break;*/

                        case found_goal :
                        {
                                #ifdef PRINT_DRIVING_HSM_EVENTS

                                        printf("found_goal\n\r");

                                #endif

                                PostEvent.EventType = LowerLance;
                                PostMasterHSM(PostEvent);

                        }
                        break;

                        case found_knight_rvs :
```

```
                {
                        #ifdef PRINT_DRIVING_HSM_EVENTS

                                printf("found_knight_fwd\n\r");

                        #endif

                        PostEvent.EventType = LowerLance;
                        PostMasterHSM(PostEvent);

                }
                break;

                case lost_knight_rvs :
                {
                        #ifdef PRINT_DRIVING_HSM_EVENTS

                                printf("lost_knight_fwd\n\r");

                        #endif

                        PostEvent.EventType = LowerLance;
                        PostMasterHSM(PostEvent);

                }
                break;

                case found_knight_fwd :
                {
                        #ifdef PRINT_DRIVING_HSM_EVENTS

                                printf("found_knight_rvs\n\r");

                        #endif

                        PostEvent.EventType = LowerLance;
                        PostMasterHSM(PostEvent);

                }
                break;

                case Crossed_Slow_Down_Rvs_Offset:
                {
                        #ifdef PRINT_DRIVING_HSM_EVENTS


        printf("Crossed_Slow_Down_Rvs_Offset\n\r");

                        #endif

                        PostEvent.EventType = FindingWallRvs;
                        PostMotorService(PostEvent);

                        NextState = FindingWallRvsState;
                        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;

           case FindingWallFwdState :       // 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 = DuringFindingWallFwdState(CurrentEvent);
         //process any events

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

         #endif

         if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
         {
            switch (CurrentEvent.EventType)
            {
               /*case Crossed_Offset_Nominal :
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS


        printf("Crossed_Offset_Nominal\n\r");

                                        #endif

                                        PostEvent.EventType = FindingWallFwd;
                                        PostMotorService(PostEvent);
                                }
                                break;

                                case Crossed_Offset_Min :
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS


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

```
                                        #endif

                                        //PostEvent.EventType = VeerLeftFwd;
                                        //PostMotorService(PostEvent);
                                }
                                break;

                                case Crossed_Offset_Max :
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS


        printf("Crossed_Offset_Max\n\r");

                                        #endif

                                        //PostEvent.EventType = VeerRightFwd;
                                        //PostMotorService(PostEvent);
                                }
                                break;*/

                                case Crossed_Home_Fwd_Offset :
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS


        printf("Crossed_Home_Fwd_Offset\n\r");

                                        #endif

                                        PostEvent.EventType = STOP;
                                        PostMotorService(PostEvent);


                                        NextState = Stopped;
                                        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;

                case FindingWallRvsState :       // 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 = DuringFindingWallRvsState(CurrentEvent);
          //process any events

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

          #endif

          if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
          {
             switch (CurrentEvent.EventType)
             {
                /*case Crossed_Offset_Nominal :
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS


        printf("Crossed_Offset_Nominal\n\r");

                                        #endif

                                        PostEvent.EventType = FindingWallRvs;
                                        PostMotorService(PostEvent);
                                }
                                break;

                                case Crossed_Offset_Min :
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS


        printf("Crossed_Offset_Min\n\r");

                                        #endif

                                        //PostEvent.EventType = VeerRightRvs;
                                        //PostMotorService(PostEvent);
                                }
                                break;

                                case Crossed_Offset_Max :
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS


        printf("Crossed_Offset_Max\n\r");

                                        #endif

                                        //PostEvent.EventType = VeerLeftRvs;
                                        //PostMotorService(PostEvent);
```

```
                                }
                                break;*/

                                case Crossed_Home_Rvs_Offset:
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS


        printf("Crossed_Home_Rvs_Offset\n\r");

                                        #endif

                                        PostEvent.EventType = STOP;
                                        PostMotorService(PostEvent);


                                        NextState = Stopped;
                                        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;

                case Stopped :       // 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 = DuringStopped(CurrentEvent);
         //process any events

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

         #endif

        /* if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is
active
         {
            switch (CurrentEvent.EventType)
            {
               case ChargeFieldFwd :
```

```
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS

                                                printf("ChargeFieldFwd\n\r");

                                        #endif

                                        NextState = ChargingFieldFwdState;
                                        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 ChargeFieldRvs :
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS

                                                printf("ChargeFieldRvs\n\r");

                                        #endif

                                        NextState = ChargingFieldRvsState;
                                        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 FindWallFwd :
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS

                                                printf("FindWallFwd\n\r");

                                        #endif

                                        NextState = FindingWallFwdState;
                                        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 FindWallRvs :
                                {
                                        #ifdef PRINT_DRIVING_HSM_EVENTS

                                                printf("FindWallRvs\n\r");

                                        #endif

                                        NextState = FindingWallRvsState;
                                        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;
       RunDrivingHSM(CurrentEvent);

       CurrentState = NextState; //Modify state variable

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

     CurrentState = NextState;

     return(ReturnEvent);
}
```

```
/*****************************************************************************
 Function
     StartDrivingHSM

 Parameters
     None

 Returns
     None

 Description
     Does any required initialization for this state machine
 Notes

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

  #ifdef PRINT_DRIVING_SM_CALLS
         if(CurrentEvent.EventType != ES_TIMEOUT) //don't spam terminal
windows if this is a routine update/polling event
         {
                printf("StartDrivingHSM\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 )
   {
               switch(CurrentEvent.EventParam)
               {
                       case 0:
                              CurrentState = Stopped;
                       break;

                       case 1:
                              CurrentState = ChargingFieldFwdState;
                       break;

                       case 2:
                              CurrentState = ChargingFieldRvsState;
                       break;

                       case 3:
                              CurrentState = FindingWallFwdState;
                       break;

                       case 4:
```

```
                                CurrentState = FindingWallRvsState;
                        break;

                        default:
                                CurrentState = Stopped;
                        break;
                }

    }

    RunDrivingHSM(CurrentEvent);

}


/*****************************************************************************
 Function
     QueryDrivingHSM

 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 QueryDrivingHSM ( void )
{
  #ifdef PRINT_DRIVING_SM_CALLS

    printf("QueryDrivingHSM\n\r");

  #endif

   return(CurrentState);

}


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

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

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

```
        {
    printf("DuringChargingFieldFwdState\n\r");
        }

  #endif

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

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

  #endif
```

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

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

  #endif

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

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

  #endif

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

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