

```

/*****
Module
    Intermission1HSM.c

Description
    This is a template file for implementing state machines.

*****/
/*----- Include Files -----*/
// Basic includes for a program using the Events and Services Framework
#include "ES_Configure.h"
#include "ES_Framework.h"
#include <stdio.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 "Intermission1HSM.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 DuringGoToWall1HSM(ES_Event Event);

/*----- Module Variables -----*/
// everybody needs a state variable, you may need others as well
static Intermission1State_t CurrentState;

/*----- Module Code -----*/
/*****
Function
    RunIntermission1HSM

Parameters
    ES_Event: the event to process

Returns
    ES_Event: an event to return

Description
    add your description here

*****/
ES_Event RunIntermission1HSM(ES_Event CurrentEvent)
{
    unsigned char MakeTransition = false; // are we making a state
transition?
    Intermission1State_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_INTERMISSION1_HSM_CALLS

    printf("RunINTERMISSION1HSM\n\r");

#endif

switch(CurrentState)
{
    case GoingToWall1:

        #ifdef PRINT_INTERMISSION1_HSM_STATES

            printf("GoingToWall1\n\r");

        #endif

        // Execute During function for Charging. ES_ENTRY & ES_EXIT are
processed here allow the lower level state machines to re-map or consume the
event

        CurrentEvent = DuringGoToWall1HSM(CurrentEvent);
        // Process any events

        break;

}

// If we are making a state transition
if (MakeTransition == true)
{
    // Execute exit function for current state
    CurrentEvent.EventType = ES_EXIT;
    RunIntermission1HSM(CurrentEvent);

    CurrentState = NextState; //Modify state variable

    // Execute entry function for new state
    // this defaults to ES_ENTRY
    RunIntermission1HSM(EntryEventKind);
}
return(ReturnEvent);
}
/*****
Function
    StartIntermission1HSM

Parameters
    None

Returns
    None

Description
    Does any required initialization for this state machine

```

Notes

Author

J. Edward Carryer, 2/18/99, 10:38AM

*****/

```
void StartIntermission1HSM(ES_Event CurrentEvent)
{
    // local variable to get debugger to display the value of CurrentEvent
    ES_Event LocalEvent = CurrentEvent;

    #ifdef PRINT_INTERMISSION1_HSM_CALLS
        printf("StartIntermission1HSM\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 = GoingToWall1;
    }
    // call the entry function (if any) for the ENTRY_STATE
    RunIntermission1HSM(CurrentEvent);
}
```

Function

QueryIntermission1HSM

Parameters

None

Returns

Intermission1State_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

*****/

```
Intermission1State_t QueryIntermission1HSM(void)
{
    #ifdef PRINT_INTERMISSION1_HSM_CALLS
        printf("QueryIntermission1HSM\n\r");
    #endif
    return(CurrentState);
}
```

```

/*****
private functions
*****/
static ES_Event DuringGoToWall1HSM(ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assumes no re-mapping or consumption

#ifdef PRINT_Intermission1_HSM_CALLS
    printf("DuringGoingToWall1HSM\n\r");
#endif

    // Process ES_ENTRY, ES_ENTRY_HISTORY, and ES_EXIT events
    if((Event.EventType == ES_ENTRY) || (Event.EventType ==
ES_ENTRY_HISTORY))
    {
        // Implement any entry actions required for this state machine

        //keep driving until we hit the wall if we have not done so
already
        if(get_sonic_fwd() > HOME_FWD_OFFSET)
        {
            Event.EventParam = 3;
            StartDrivingHSM(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
        RunDrivingHSM(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
        RunDrivingHSM(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);  
}
```