

```

/*****
Module
    SocceringSM.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
02/13/10 12:29 jec          to make implementing history entry cleaner
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
02/09/10 17:21 jec          function that belongs in the run function
updated comments about internal transitions on
During funtion
02/18/09 10:14 jec          removed redundant call to RunLowerlevelSM in
EV_Entry
02/20/07 21:37 jec          processing in During function
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 "SocceringSM.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 DuringSocceringSM( ES_Event Event);

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

/*----- Module Code -----*/
/*****
Function
    RunSocceringSM

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

#endif

    switch ( CurrentState )
    {
        case Soccering : // 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 = DuringSocceringSM(CurrentEvent);
                //process any events

```

```

    #ifdef PRINT_SOCCERING_SM_STATES
    if(CurrentEvent.EventType != ES_TIMEOUT) //don't spam terminal
windows if this is a routine update/polling event
    {
        printf("Soccering\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
transition
                //MakeTransition = false; //mark that we are taking a
entry
                // if transitioning to a state with history change kind of
                //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
}

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

    CurrentState = NextState; //Modify state variable

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

CurrentState = NextState;

return(ReturnEvent);
}

```

```

/*****
Function
    StartSocceringSM

Parameters
    None

Returns
    None

Description
    Does any required initialization for this state machine
Notes

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

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

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

/*****
Function
    QuerySocceringSM

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 QuerySocceringSM (void)

{

 #ifdef PRINT_SOCCKERING_SM_CALLS

 printf("QuerySocceringSM\n\r");

 #endif

 return(CurrentState);

}

/******

 private functions

*****/

static ES_Event DuringSocceringSM(ES_Event Event)

{

 ES_Event ReturnEvent = Event; // assumes no re-mapping or consumption

 ES_Event PostEvent;

 #ifdef PRINT_SOCCKERING_SM_CALLS

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

 {
 printf("DuringSocceringSM\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

 PostEvent.EventType = STOP;

 PostMotorService(PostEvent);

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

 StartShootingSM(Event);

 PostEvent.EventType = Fire_All_Balls;

 PostMasterHSM(PostEvent);

 // 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 )
    {

        StopShootingSM(Event);
        // 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
    {
        RunShootingSM(Event);
        // 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);
}

```