```c
/****************************************************************************
 Module
   DiscoBallSM.c

 Revision
   1.0.1

 Description
   This is a template file for implementing flat state machines under the
   Gen2 Events and Services Framework.

****************************************************************************/
/*----------------------------- Include Files -----------------------------*/
/* 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 "ES_Configure.h"
#include "ES_Framework.h"
#include "DiscoBallSM.h"
#include "Constants.h"
#include "Hardware.h"

/*----------------------------- Module Defines ----------------------------*/

/*---------------------------- Module Functions ---------------------------*/
/* prototypes for private functions for this machine.They should be functions
   relevant to the behavior of this state machine
*/

/*---------------------------- Module Variables ---------------------------*/
// everybody needs a state variable, you may need others as well.
// type of state variable should match htat of enum in header file
static DiscoBallState_t CurrentState;
static uint8_t Direction = FORWARD;

// with the introduction of Gen2, we need a module level Priority var as well
static uint8_t MyPriority;


/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitDiscoBallSM

 Parameters
     uint8_t : the priorty of this service

 Returns
     bool, false if error in initialization, true otherwise

 Description
     Saves away the priority, sets up the initial transition and does any
     other required initialization for this state machine
 Notes

 Author
     bag
****************************************************************************/
bool InitDiscoBallSM( uint8_t Priority )
{
  ES_Event ThisEvent;

  MyPriority = Priority;
  // put us into the Initial PseudoState
      CurrentState = SearchingWaiting;
//    SetDirectionDiscoBall(Direction);
      SetDutyIndicator(DISCO_FORWARD_DUTY);
      ES_Timer_InitTimer(DISCO_TIMER, DISCO_SPIN_TIME);


  // post the initial transition event
  ThisEvent.EventType = ES_INIT;
  if (ES_PostToService( MyPriority, ThisEvent) == true)
  {
      return true;
  }else
  {
      return false;
  }
}

/****************************************************************************
 Function
     PostDiscoBallSM

 Parameters
     EF_Event ThisEvent , the event to post to the queue

 Returns
     boolean False if the Enqueue operation failed, True otherwise

 Description
     Posts an event to this state machine's queue
 Notes

 Author
     bag
****************************************************************************/
bool PostDiscoBallSM(ES_Event ThisEvent)
{
  return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunDiscoBallSM

 Parameters
   ES_Event : the event to process

 Returns
   ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise

 Description
   add your description here
 Notes
   uses nested switch/case to implement the machine.
 Author
            bag
****************************************************************************/
ES_Event RunDiscoBallSM( ES_Event ThisEvent )
{
  ES_Event ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

      DiscoBallState_t NextState;
      NextState = CurrentState;
```

```c
    switch (CurrentState)
    {
            case SearchingForward:

                    if(ThisEvent.EventType == ES_TIMEOUT)
                    {
                            //printf("DISCO BALL STOP\r\n");
                            NextState = SearchingWaiting;
                            SetDutyIndicator(DISCO_DUTY_OFF);
                            ES_Timer_InitTimer(DISCO_TIMER, DISCO_WAIT_TIME);
                    }

                    else if(ThisEvent.EventType == ES_PAIR_SUCCESSFUL)
                    {
                            NextState = IndicatingPaired;
                            //Direction = FORWARD;
                            //SetDirectionDiscoBall(Direction);
                            SetDutyIndicator(DISCO_FORWARD_DUTY);
                    }

                    break;

            case SearchingWaiting:


                    //else if((ThisEvent.EventType == ES_TIMEOUT) && (Direction ==
  REVERSE))
                    if(ThisEvent.EventType == ES_TIMEOUT)
                    {
                            //printf("DISCO BALL SPIN\r\n");
                            NextState = SearchingForward;
                            //Direction = FORWARD;
                            //SetDirectionDiscoBall(Direction);
                            SetDutyIndicator(DISCO_FORWARD_DUTY);
                            ES_Timer_InitTimer(DISCO_TIMER, DISCO_SPIN_TIME);
                    }

                    else if(ThisEvent.EventType == ES_PAIR_SUCCESSFUL)
                    {
                            NextState = IndicatingPaired;
                            //Direction = FORWARD;
                            //SetDirectionDiscoBall(Direction);
                            SetDutyIndicator(DISCO_FORWARD_DUTY);
                    }


                    break;

            case SearchingReverse:

                    if(ThisEvent.EventType == ES_TIMEOUT)
                    {
                            NextState = SearchingWaiting;
                            SetDutyIndicator(DISCO_DUTY_OFF);
                    }

                    else if(ThisEvent.EventType == ES_PAIR_SUCCESSFUL)
                    {
                            NextState = IndicatingPaired;
                            Direction = FORWARD;
                            SetDirectionDiscoBall(Direction);
                            SetDutyIndicator(DISCO_FORWARD_DUTY);
                    }
```

```c
                  break;

            case IndicatingPaired:

                  if(ThisEvent.EventType == ES_LOST_CONNECTION)
                  {
                        NextState = SearchingForward;
                        //Direction = FORWARD;
                        //SetDirectionDiscoBall(Direction);
                        SetDutyIndicator(DISCO_FORWARD_DUTY);
                        ES_Timer_InitTimer(DISCO_TIMER, DISCO_SPIN_TIME);
                  }

                  break;
  }  // end switch on Current State

      CurrentState = NextState;
  return ReturnEvent;
}




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