

```

/*****
Module
    ByterSM.c

Revision
    1.0.1

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

Notes

History
When          Who          What/Why
-----
01/15/12 11:12 jec        revisions for Gen2 framework
11/07/11 11:26 jec        made the queue static
10/30/11 17:59 jec        fixed references to CurrentEvent in RunTemplateSM()
10/23/11 18:20 jec        began conversion from SMTemplate.c (02/20/07 rev)
*****/
/*----- 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 "ByterSM.h"
#include <sys/attrs.h>
#include "dbprintf.h"

#include "ServoService.h"

/*----- Module Defines -----*/
#define BITES_IN_24S 3
#define BITING_TIME 4000
#define COOLDOWN_TIME 3000
#define NUM_TRACKED_BYTE_TIMES 3

#define T4_PERIOD 0xFFFF
#define _24_SECONDS 1875000 //24s as a number of ticks
#define _1_SECOND 78125
#define ROLLOVER_COUNT_INDEX 1
#define CURRENT_TIME_INDEX 0
typedef union{
    uint32_t FullCount;
    uint16_t Counters[2];
    //1 is the rollover counter
    //0 is the current count;
}ByterTracker_t;

/*----- Module Functions -----*/
/* prototypes for private functions for this machine.They should be functions
   relevant to the behavior of this state machine
*/
static void configTimer4(void);

/*----- Module Variables -----*/
// everybody needs a state variable, you may need others as well.

```

```

// type of state variable should match that of enum in header file
static ByteState_t CurrentState;
static bool SingleByteCooldownFlag = false;
static bool validTimingFlag = false;

volatile static ByteTracker_t ByteTimer;

static uint8_t currentByteIndex = 0;
static uint32_t ByteTracker[NUM_TRACKED_BYTE_TIMES];
static bool biteInit = false;

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

/*----- Module Code -----*/
/*****
Function
    InitByteSM

Parameters
    uint8_t : the priority 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
    J. Edward Carryer, 10/23/11, 18:55
*****/
bool InitByteSM(uint8_t Priority)
{
    ES_Event_t ThisEvent;

    MyPriority = Priority;

    configTimer4();
    ByteTimer.Counters[ROLLOVER_COUNT_INDEX] = 100; //set timer to arbitrary high number
    initially

    // put us into the Initial PseudoState
    CurrentState = InitPState2;
    // post the initial transition event
    ThisEvent.EventType = ES_INIT;
    if (ES_PostToService(MyPriority, ThisEvent) == true)
    {
        return true;
    }
    else
    {
        return false;
    }
}

```

```

/*****
Function
    PostByteSM

Parameters
    EF_Event_t 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
    J. Edward Carryer, 10/23/11, 19:25
*****/
bool PostByteSM(EF_Event_t ThisEvent)
{
    return ES_PostToService(MyPriority, ThisEvent);
}

/*****
Function
    RunByteSM

Parameters
    ES_Event_t : the event to process

Returns
    ES_Event_t, ES_NO_EVENT if no error ES_ERROR otherwise

Description
    add your description here
Notes
    uses nested switch/case to implement the machine.
Author
    J. Edward Carryer, 01/15/12, 15:23
*****/
ES_Event_t RunByteSM(ES_Event_t ThisEvent)
{
    ES_Event_t ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

    switch (CurrentState)
    {
        case InitPState2:          // If current state is initial Psedudo State
        {
            if (ThisEvent.EventType == ES_INIT)    // only respond to ES_Init
            {
                CurrentState = ReadyToBite;
            }
        }
        break;

        case ReadyToBite:
        {
            switch (ThisEvent.EventType)
            {
                case ES_BITE_CMD:

```

```

{
    //----- deploy biter -----
    ES_Event_t BiteEvent = {ES_BITE, 0};
    PostServoService(BiteEvent);
    // -----

    ByteTimer.Counters[CURRENT_TIME_INDEX] = TMR4; //get lower 16 bits of timer
    ByteTracker[currentByteIndex] = ByteTimer.FullCount; //save full time
    if (currentByteIndex + 1 <= 2){ //increment or loop index
        currentByteIndex++; //increment bite Index
    } else {
        currentByteIndex = 0; //loop index back to 0
    }

    biteInit = true;
    CurrentState = Biting; //change to biting state

    ES_Timer_InitTimer(BitingTimer, 4000);
}
break;

default:
    ;
}
}
break;

case Biting:
{
    switch (ThisEvent.EventType)
    {
        case ES_TIMEOUT:
        {
            if (ThisEvent.EventParam == BitingTimer){ //Byter has been deployed for max
time
                // ----- retract biter -----
                ES_Event_t PrimeEvent = {ES_PRIME_BYTER, 0};
                PostServoService(PrimeEvent);

                // -----

                SingleByteCooldownFlag = false; //set cooldown flag to false
                ES_Timer_InitTimer(CooldownTimer, COOLDOWN_TIME); //begin single byte
cooldown timer
                CurrentState = Cooldown; //change to the cooldown state
            }
        }
        break;

        default:
            ;
    }
}
break;

case Cooldown:
{
    switch (ThisEvent.EventType)
    {

```

```

        case ES_TIMEOUT:
        {
            if (ThisEvent.EventParam == CooldownTimer){
                SingleByteCooldownFlag = true; //set cooldown flag to true
            }
        }
        break;

        case ES_PRIME_BYTER:
        {
            CurrentState = ReadyToBite;
        }
        break;

        default:
            ;
    }
}
break;

default:
    ;
}
// end switch on Current State
return ReturnEvent;
}

```

Function

QueryByterSM

Parameters

None

Returns

ByteState_t The current state of the Template state machine

Description

returns the current state of the Template state machine

Notes

Author

J. Edward Carryer, 10/23/11, 19:21

*****/

ByteState_t QueryByterSM(void)

```

{
    return CurrentState;
}

```

bool checkByterTiming(void){

bool retVal = false;

validTimingFlag = false; //set timing flag to false

uint32_t currentTime = ByteTimer.FullCount; //get the current time

for(int i = 0; i < NUM_TRACKED_BYTE_TIMES; i++){ //check all the byte times

if (currentTime - ByteTracker[i] >= _24_SECONDS){ //if any of the times happened more than 24s ago

validTimingFlag = true; //set timing flag to true

}else{

}

```

    }

    if (CurrentState == Cooldown){ //if we are in cooldown
        if (SingleByteCooldownFlag){ //if byte has cooled down
            if (validTimingFlag){ //if we have < 3 bytes in 24 secs
                ES_Event_t NewEvent;
                NewEvent.EventType = ES_PRIME_BYTER; //setup new event
                PostByteSM(NewEvent); //post event to self
            }
        }
    }

    return retVal;
}

bool getBiteInitFlag(void){
    return biteInit;
}

bool clearBiteInitFlag(void){
    biteInit = false;
}

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

static void configTimer4(void){
    //-----CONFIG TIMER4 SETTINGS-----//
    T4CONbits.ON = 0; //disable timer 4
    T4CONbits.TGATE = 0; //disable gated time accumulation mode
    T4CONbits.TCS = 0; //select PBCLK as source
    T4CONbits.TCKPS = 0b111; //prescale of 256
    TMR4 = 0; //Clear timer register
    PR4 = T4_PERIOD; //set timer period

    //-----CONFIG TIMER 4 INTERRUPTS-----//
    __builtin_disable_interrupts();
    INTCONbits.MVEC = 1; // enable multivector mode
    IPC4bits.T4IP = 7; //set Timer4Priority

    IFS0CLR = _IFS0_T4IF_MASK; //clear T4 interrupt flag

    IEC0SET = _IEC0_T4IE_MASK; //enable T4 Interrupts
    __builtin_enable_interrupts(); //enable global interrupts

    T4CONbits.ON = 1; //enable timer 4
}

void __ISR(_TIMER_4_VECTOR, IPL7SOFT) TMR4_ROLLOVER(void){
    //DB_printf("TIMER ROLLOVER\r\n");

    __builtin_disable_interrupts();
    if (IFS0bits.T4IF == 1){
        ByteTimer.Counters[ROLLOVER_COUNT_INDEX]++; //increment rollovers
        IFS0CLR = _IFS0_T4IF_MASK; //clear interrupt flag
    }
}

```

```
    __builtin_enable_interrupts();  
}
```