```
/*******************************************************************************
 Module
   EventCheckers.c

 Revision
   1.0.1

 Description
   This is the sample for writing event checkers along with the event
   checkers used in the basic framework test harness.

 Notes
   Note the use of static variables in sample event checker to detect
   ONLY transitions.

 History
 When           Who     What/Why
 -------------- ---    --------
 08/06/13 13:36 jec     initial version
*******************************************************************************/

// this will pull in the symbolic definitions for events, which we will want
// to post in response to detecting events
#include "ES_Configure.h"
// This gets us the prototype for ES_PostAll
```

```
#include "ES_Framework.h"
// this will get us the structure definition for events, which we will need
// in order to post events in response to detecting events
#include "ES_Events.h"
// if you want to use distribution lists then you need those function
// definitions too.
#include "ES_PostList.h"
// This include will pull in all of the headers from the service modules
// providing the prototypes for all of the post functions
#include "ES_ServiceHeaders.h"
// this test harness for the framework references the serial routines that
// are defined in ES_Port.c
#include "ES_Port.h"
// include our own prototypes to insure consistency between header &
// actual functions definition
#include "EventCheckers.h"
#include "dbprintf.h"

// This is the event checking function sample. It is not intended to be
// included in the module. It is only here as a sample to guide you in writing
// your own event checkers
#if 0
/****************************************************************************
 Function
   Check4Lock
 Parameters
```

```
  None
 Returns
  bool: true if a new event was detected
 Description
  Sample event checker grabbed from the simple lock state machine example
 Notes
  will not compile, sample only
 Author
  J. Edward Carryer, 08/06/13, 13:48
****************************************************************************/
bool Check4Lock(void)
{
  static uint8_t  LastPinState = 0;
  uint8_t         CurrentPinState;
  bool            ReturnVal = false;

  CurrentPinState = LOCK_PIN;
  // check for pin high AND different from last time
  // do the check for difference first so that you don't bother with a test
  // of a port/variable that is not going to matter, since it hasn't changed
  if ((CurrentPinState != LastPinState) &&
          (CurrentPinState == LOCK_PIN_HI)) // event detected, so post detected event
  {
    ES_Event ThisEvent;
    ThisEvent.EventType   = ES_LOCK;
    ThisEvent.EventParam  = 1;
```

```
            // this could be any of the service post functions, ES_PostListx or
            // ES_PostAll functions
            ES_PostAll(ThisEvent);
    ReturnVal = true;
  }
  LastPinState = CurrentPinState; // update the state for next time

  return ReturnVal;
}

#endif

/****************************************************************************
 Function
   Check4Keystroke
 Parameters
   None
 Returns
   bool: true if a new key was detected & posted
 Description
   checks to see if a new key from the keyboard is detected and, if so,
   retrieves the key and posts an ES_NewKey event to TestHarnessService0
 Notes
   The functions that actually check the serial hardware for characters
   and retrieve them are assumed to be in ES_Port.c
   Since we always retrieve the keystroke when we detect it, thus clearing the
```

```
   hardware flag that indicates that a new key is ready this event checker
   will only generate events on the arrival of new characters, even though we
   do not internally keep track of the last keystroke that we retrieved.
 Author
   J. Edward Carryer, 08/06/13, 13:48
*****************************************************************************/
bool Check4Keystroke(void)
{
  if (IsNewKeyReady())   // new key waiting?
  {
    ES_Event_t ThisEvent;
    ThisEvent.EventType   = ES_NEW_KEY;
    ThisEvent.EventParam  = GetNewKey();
            ES_PostAll(ThisEvent);
            return true;
  }
  return false;
}

static bool ACORN_Proximity_Event_bool = false;

void EnableACORNProximitySensing(void){
    TRISBbits.TRISB15 = 1;
    ANSELBbits.ANSB15 = 0;
    ACORN_Proximity_Event_bool = true;
}
```

```
void DisableACORNProximitySensing(void){
    ACORN_Proximity_Event_bool = false;
}

bool Check4ACORN(void)
{
            bool ObjectReturnVal = false;
            if(true == ACORN_Proximity_Event_bool){
            if(0 == PORTBbits.RB15){
           ES_Event_t ObjectDetected;
           ObjectDetected.EventType = ES_STRIKING_DISTANCE;
           ObjectDetected.EventParam = 0;
            PostMasterSM_Leader(ObjectDetected);
           ACORN_Proximity_Event_bool = false;
           ObjectReturnVal = true;
            }
            }
            return ObjectReturnVal;
}

static bool Spoon_Hi_Event_bool = false;

void EnableSpoonHiSensing(void){
    TRISBbits.TRISB5 = 1;
    Spoon_Hi_Event_bool = true;
```

```
}

bool Check4SpoonHi(void){
        bool SpoonReturnVal = false;
        if(true == Spoon_Hi_Event_bool){
        if(1 == PORTBbits.RB5){
        ES_Event_t SpoonEvent;
        SpoonEvent.EventType = ES_SPOON_STOPPED;
        PostMasterSM_Leader(SpoonEvent);
        Spoon_Hi_Event_bool = false;
        SpoonReturnVal = true;
        }
        }
        return SpoonReturnVal;
}

static bool Tape_Event_bool = false;

void EnableTapeSensing(void){
   Tape_Event_bool = true;
}

bool Check4TapeHi(void){
        bool TapeReturnVal = false;
        if(true == Tape_Event_bool){
        if(1 == PORTBbits.RB5){
```

```
            ES_Event_t TapeEvent;
            TapeEvent.EventType = ES_TAPE_DETECTED;
             PostMasterSM_Leader(TapeEvent);
            Tape_Event_bool = false;
            TapeReturnVal = true;
             }
             }
             return TapeReturnVal;
}

bool CheckGameplaySwitch(void){
             //return false;
             bool HaveSwitched = false;
             static bool HaveSwitchPreset = false;
             static uint8_t PrevVal;
             if(false == HaveSwitchPreset){
        TRISBbits.TRISB12 = 1;
        ANSELBbits.ANSB12 = 0;
        HaveSwitchPreset = true;
        PrevVal = PORTBbits.RB12;
             }
             uint8_t CurrentVal = PORTBbits.RB12;
             if((CurrentVal != PrevVal) && (1 == CurrentVal)){
        HaveSwitched = true;
        ES_Event_t SwitchEvent;
        SwitchEvent.EventType = ES_GAME_SWITCH;
```

```
            PostMasterSM_Leader(SwitchEvent);
            }
    PrevVal = CurrentVal;
            return HaveSwitched;
}
```