```
/*******************************************************************************
 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 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 functionsdefinition
#include "EventCheckers.h"

#include "ADS12.h"
#include "SCI_Send.h"
#include "Coach.h"
#include "SCI_Receive.h"
#include "Ultrason.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



//#include "ES_General.h"

//#include "SCI_Receive.h"

bool Receive_Check(void)
{

      static unsigned char RX_Message;
```

```
    static ES_Event PostEvent;
    bool ReturnVal = false;

    if ((SCI1SR1 & _S12_RDRF) == _S12_RDRF)
        {
            SCI1SR1 &= ~_S12_RDRF; //clear interrupt flag
            RX_Message = SCI1DRL;                //Read data register
            //printf("The message I got is:\n\r");
            //printf("%x\n\r",RX_Message);
            PostEvent.EventType = Received;
            PostEvent.EventParam = RX_Message;
            Post_SCI_Receive (PostEvent);

            ReturnVal = true;
        }

    return ReturnVal;
}



bool PairButton_Check(void)
{

    static ES_Event PostEvent;
    bool ReturnVal = false;

    if ((PTT & BIT3HI) == BIT3HI)      //
        {
            PostEvent.EventType = PairButtonPress;
            Post_Coach (PostEvent);
            //printf("Pairbutton Event posted!!!");
            ReturnVal = true;
        }
    return ReturnVal;
}



bool TagOut_Check(void)
{
    static unsigned char Pos, col, Senddata;
    static unsigned short Read;
    //static unsigned int R;
    static ES_Event PostEvent;
    bool ReturnVal = false;

    col = 0;
    Senddata = 0;

    //if((PTT & BIT1HI) == BIT1HI)
    if ((PTU & BIT6HI) == BIT6HI)
        {

            Read = ADS12_ReadADPin(0) / 4;   // Read from the AD3


            //R = (unsigned int)Read / 4;
            //printf("Tageoutloc:%i\n\r", R);
            if (0 <= Read && Read <= 64)
```

```
                {
                    Pos = 0;
                }

            else if (65 <= Read && Read <= 128)
                {
                    Pos = 1;
                }

            else if (129 <= Read && Read <= 192)
                {
                    Pos = 2;
                }

            else if (193 <= Read && Read <= 256)
                {
                    Pos = 3;
                }


            if ((PTAD & BIT7HI) == BIT7HI)   //Read in color
                {
                    //printf("Green!");
                    col = 1;;     // If COL is set, it is green, otherwise it is red
                }

            Senddata = ((Pos << 1) | col);

            printf("Posis:%x\n\r", Pos);
            printf("MLoca: %x\n\r", Senddata);

            PostEvent.EventType = TagOut;
            PostEvent.EventParam = Senddata;
            Post_Coach (PostEvent);

            ReturnVal = true;
        }

    return ReturnVal;
}



bool Reset_Check(void)
{
    static ES_Event PostEvent;
    bool ReturnVal = false;

    if ((PTT & BIT2HI) == BIT2HI)
        {
            PostEvent.EventType = Reset;
            Post_Coach (PostEvent);
            //printf("PPPPPPPPPost RRRRRRRRRRRRRRRRReset");

            ReturnVal = true;
        }

    return ReturnVal;
}
```

```
/****************************************************************************
 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 TestHarnessService1
 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 ThisEvent;
    ThisEvent.EventType = ES_NEW_KEY;
    ThisEvent.EventParam = GetNewKey();
    // test distribution list functionality by sending the 'L' key out via
    // a distribution list.
    if ( ThisEvent.EventParam == 'L'){
      ES_PostList00( ThisEvent );
    }else{     // otherwise post to Service 0 for processing
      PostTestHarnessService0( ThisEvent );
    }
    return true;
  }
  return false;
}         */
```