```
//
// begin license header
//
// This file is part of Pixy CMUcam5 or "Pixy" for short
//
// All Pixy source code is provided under the terms of the
// GNU General Public License v2 (http://www.gnu.org/licenses/gpl-2.0.html).
// Those wishing to use Pixy source code, software and/or
// technologies under different licensing terms should contact us at
// cmucam@cs.cmu.edu. Such licensing terms are available for
// all portions of the Pixy codebase presented here.
//
// end license header
//
// This file is for defining the Block struct and the Pixy template class version 2.
// (TPixy2).  TPixy takes a communication link as a template parameter so that
// all communication modes (SPI, I2C and UART) can share the same code.
//

#ifndef _PIXY2CCC_H
#define _PIXY2CCC_H

#define CCC_MAX_SIGNATURE                   7

#define CCC_RESPONSE_BLOCKS                 0x21
#define CCC_REQUEST_BLOCKS                  0x20

// Defines for sigmap:
// You can bitwise "or" these together to make a custom sigmap.
// For example if you're only interested in receiving blocks
// with signatures 1 and 5, you could use a sigmap of
// PIXY_SIG1 | PIXY_SIG5
#define CCC_SIG1                    1
#define CCC_SIG2                    2
#define CCC_SIG3                    4
#define CCC_SIG4                    8
#define CCC_SIG5                    16
#define CCC_SIG6                    32
#define CCC_SIG7                    64
#define CCC_COLOR_CODES             128

#define CCC_SIG_ALL                 0xff // all bits or'ed together

struct Block
{
  // print block structure!
  void print()
  {
    int i, j;
    char buf[128], sig[6], d;
    bool flag;
    if (m_signature>CCC_MAX_SIGNATURE) // color code! (CC)
    {
      // convert signature number to an octal string
      for (i=12, j=0, flag=false; i>=0; i-=3)
      {
        d = (m_signature>>i)&0x07;
        if (d>0 && !flag)
          flag = true;
        if (flag)
          sig[j++] = d + '0';
      }
      sig[j] = '\0';
      sprintf(buf, "CC block sig: %s (%d decimal) x: %d y: %d width: %d height: %d angle:
%d index: %d age: %d", sig, m_signature, m_x, m_y, m_width, m_height, m_angle, m_index,
m_age);
    }
    else // regular block.  Note, angle is always zero, so no need to print
      sprintf(buf, "sig: %d x: %d y: %d width: %d height: %d index: %d age: %d",
```

```
m_signature, m_x, m_y, m_width, m_height, m_index, m_age);
    Serial.println(buf);
  }

  uint16_t m_signature;
  uint16_t m_x;
  uint16_t m_y;
  uint16_t m_width;
  uint16_t m_height;
  int16_t m_angle;
  uint8_t m_index;
  uint8_t m_age;
};

template <class LinkType> class TPixy2;

template <class LinkType> class Pixy2CCC
{
public:
  Pixy2CCC(TPixy2<LinkType> *pixy)
  {
    m_pixy = pixy;
  }

  int8_t getBlocks(bool wait=true, uint8_t sigmap=CCC_SIG_ALL, uint8_t maxBlocks=0xff);

  uint8_t numBlocks;
  Block *blocks;

private:
  TPixy2<LinkType> *m_pixy;
};

template <class LinkType> int8_t Pixy2CCC<LinkType>::getBlocks(bool wait, uint8_t sigmap,
uint8_t maxBlocks)
{
  blocks = NULL;
  numBlocks = 0;

  while(1)
  {
    // fill in request data
    m_pixy->m_bufPayload[0] = sigmap;
    m_pixy->m_bufPayload[1] = maxBlocks;
    m_pixy->m_length = 2;
    m_pixy->m_type = CCC_REQUEST_BLOCKS;

    // send request
    m_pixy->sendPacket();
    if (m_pixy->recvPacket()==0)
    {
      if (m_pixy->m_type==CCC_RESPONSE_BLOCKS)
      {
        blocks = (Block *)m_pixy->m_buf;
        numBlocks = m_pixy->m_length/sizeof(Block);
        return numBlocks;
      }
          // deal with busy and program changing states from Pixy (we'll wait)
      else if (m_pixy->m_type==PIXY_TYPE_RESPONSE_ERROR)
      {
        if ((int8_t)m_pixy->m_buf[0]==PIXY_RESULT_BUSY)
        {
          if(!wait)
            return PIXY_RESULT_BUSY; // new data not available yet
                }
          else if ((int8_t)m_pixy->m_buf[0]!=PIXY_RESULT_PROG_CHANGING)
          return m_pixy->m_buf[0];
      }
    }
```

```
    else
      return PIXY_RESULT_ERROR;  // some kind of bitstream error

    // If we're waiting for frame data, don't thrash Pixy with requests.
    // We can give up half a millisecond of latency (worst case)
    delayMicroseconds(500);
  }
}

#endif
```