```c
/*
  libxbee - a C library to aid the use of Digi's Series 1 XBee modules
  running in API mode (AP=2).

  Copyright (C) 2009  Attie Grande (attie@attie.co.uk)

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <stdio.h>
#include <stdlib.h>

#include <stdarg.h>

#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>

#ifdef __GNUC__ /* ---- */
#include <unistd.h>
#include <termios.h>
#include <pthread.h>
#include <sys/time.h>
#else /* -------------- */
#include <Windows.h>
#include <io.h>
#include <time.h>
#endif /* ------------- */

#include "xbee.h"
#include "api.h"

#ifdef __GNUC__ /* ---- */
#include "xsys/linux.c"
#else /* -------------- */
#include "xsys/win32.c"
#endif /* ------------- */


#ifdef __UMAKEFILE
/* for embedded compiling */
const char *xbee_svn_version(void) {
  return "Embedded";
}
#endif

/* ################################################################# */
/* ### Memory Handling ############################################# */
/* ################################################################# */

/* malloc wrapper function */
static void *Xmalloc(size_t size) {
  void *t;
  t = malloc(size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    perror("libxbee:malloc()");
    exit(1);
  }
  return t;
}

/* calloc wrapper function */
static void *Xcalloc(size_t size) {
  void *t;
  t = calloc(1, size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    perror("libxbee:calloc()");
    exit(1);
  }
  return t;
}
```

```c

/* realloc wrapper function */
static void *Xrealloc(void *ptr, size_t size) {
  void *t;
  t = realloc(ptr,size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    perror("libxbee:realloc()");
    exit(1);
  }
  return t;
}

/* free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
static void Xfree2(void **ptr) {
  if (!*ptr) return;
  free(*ptr);
  *ptr = NULL;
}

/* ################################################################# */
/* ### Helper Functions ############################################ */
/* ################################################################# */

/* #################################################################
   returns 1 if the packet has data for the digital input else 0 */
int xbee_hasdigital(xbee_pkt *pkt, int sample, int input) {
  int mask = 0x0001;
  if (input < 0 || input > 7) return 0;
  if (sample >= pkt->samples) return 0;

  mask <<= input;
  return !!(pkt->IOdata[sample].IOmask & mask);
}

/* #################################################################
   returns 1 if the digital input is high else 0 (or 0 if no digital data present) */
int xbee_getdigital(xbee_pkt *pkt, int sample, int input) {
  int mask = 0x0001;
  if (!xbee_hasdigital(pkt,sample,input)) return 0;

  mask <<= input;
  return !!(pkt->IOdata[sample].IOdigital & mask);
}

/* #################################################################
   returns 1 if the packet has data for the analog input else 0 */
int xbee_hasanalog(xbee_pkt *pkt, int sample, int input) {
  int mask = 0x0200;
  if (input < 0 || input > 5) return 0;
  if (sample >= pkt->samples) return 0;

  mask <<= input;
  return !!(pkt->IOdata[sample].IOmask & mask);
}

/* #################################################################
   returns analog input as a voltage if vRef is non-zero, else raw value (or 0 if no analog data present) */
double xbee_getanalog(xbee_pkt *pkt, int sample, int input, double Vref) {
  if (!xbee_hasanalog(pkt,sample,input)) return 0;

  if (Vref) return (Vref / 1023) * pkt->IOdata[sample].IOanalog[input];
  return pkt->IOdata[sample].IOanalog[input];
}

/* ################################################################# */
/* ### XBee Functions ############################################## */
/* ################################################################# */

static void xbee_logf(const char *logformat, const char *function, char *format, ...) {
  char buf[128];
  va_list ap;
  FILE *log;
  va_start(ap,format);
  vsnprintf(buf,127,format,ap);
  va_end(ap);
  if (xbee.log) {
    log = xbee.log;
  } else {
    log = stderr;
  }
  fprintf(log,logformat,function,buf);
}

/* #################################################################
```

```c
   xbee_sendAT - INTERNAL
   allows for an at command to be send, and the reply to be captured */
static int xbee_sendAT(char *command, char *retBuf, int retBuflen) {
  return xbee_sendATdelay(0,0,command,retBuf, retBuflen);
}
static int xbee_sendATdelay(int preDelay, int postDelay, char *command, char *retBuf, int retBuflen) {
  struct timeval to;

  int ret;
  int bufi = 0;

  /* if there is a preDelay given, then use it and a bit more */
  if (preDelay) usleep(preDelay * 1200);

  /* get rid of any pre-command sludge... */
  memset(&to, 0, sizeof(to));
  ret = xbee_select(&to);
  if (ret > 0) {
    char t[128];
    while (xbee_read(t,127));
  }

  /* send the requested command */
  if (xbee.log) xbee_log("sendATdelay: Sending '%s'", command);
  xbee_write(command, strlen(command));

  /* if there is a postDelay, then use it */
  if (postDelay) {
    usleep(postDelay * 900);

    /* get rid of any post-command sludge... */
    memset(&to, 0, sizeof(to));
    ret = xbee_select(&to);
    if (ret > 0) {
      char t[128];
      while (xbee_read(t,127));
    }
  }

  /* retrieve the data */
  memset(retBuf, 0, retBuflen);
  memset(&to, 0, sizeof(to));
  /* select on the xbee fd... wait at most 200ms for the response */
  to.tv_usec = 200000;
  if ((ret = xbee_select(&to)) == -1) {
    perror("libxbee:xbee_sendATdelay()");
    exit(1);
  }

  if (!ret) {
    /* timed out, and there is nothing to be read */
    if (xbee.log) xbee_log("sendATdelay: No Data to read - Timeout...");
    return 1;
  }

  /* check for any dribble... */
  do {
    /* if there is actually no space in the retBuf then break out */
    if (bufi >= retBuflen - 1) {
      break;
    }

    /* read as much data as is possible into retBuf */
    if ((ret = xbee_read(&retBuf[bufi], retBuflen - bufi - 1)) == 0) {
      break;
    }

    /* advance the 'end of string' pointer */
    bufi += ret;

    /* wait at most 100ms for any more data */
    memset(&to, 0, sizeof(to));
    to.tv_usec = 100000;
    if ((ret = xbee_select(&to)) == -1) {
      perror("libxbee:xbee_sendATdelay()");
      exit(1);
    }

    /* loop while data was read */
  } while (ret);

  if (!bufi) {
    if (xbee.log) xbee_log("sendATdelay: No response...");
    return 1;
  }

  /* terminate the string */
  retBuf[bufi] = '\0';

  if (xbee.log) xbee_log("sendATdelay: Recieved '%s'",retBuf);
  return 0;
}


/* #################################################################
   xbee_start
   sets up the correct API mode for the xbee
   cmdSeq  = CC
   cmdTime = GT */
static int xbee_startAPI(void) {
  char buf[256];

  if (xbee.cmdSeq == 0 || xbee.cmdTime == 0) return 1;

  /* setup the command sequence string */
  memset(buf,xbee.cmdSeq,3);
  buf[3] = '\0';

  /* try the command sequence */
  if (xbee_sendATdelay(xbee.cmdTime, xbee.cmdTime, buf, buf, sizeof(buf))) {
    /* if it failed... try just entering 'AT' which should return OK */
    if (xbee_sendAT("AT\r\n", buf, sizeof(buf)) || strncmp(buf,"OK\r",3)) return 1;
  } else if (strncmp(&buf[strlen(buf)-3],"OK\r",3)) {
    /* if data was returned, but it wasn't OK... then something went wrong! */
    return 1;
  }

  /* get the current API mode */
  if (xbee_sendAT("ATAP\r\n", buf, sizeof(buf))) return 1;
  buf[1] = '\0';
  xbee.oldAPI = atoi(buf);

  if (xbee.oldAPI != 2) {
    /* if it wasnt set to mode 2 already, then set it to mode 2 */
    if (xbee_sendAT("ATAP2\r\n", buf, sizeof(buf)) || strncmp(buf,"OK\r",3)) return 1;
  }

  /* quit from command mode, ready for some packets! :) */
  if (xbee_sendAT("ATCN\r\n", buf, 4) || strncmp(buf,"OK\r",3)) return 1;

  return 0;
}

/* #################################################################
   xbee_end
   resets the API mode to the saved value - you must have called xbee_setup[log]API */
int xbee_end(void) {
  int ret = 1;
  xbee_con *con, *ncon;
  xbee_pkt *pkt, *npkt;

  ISREADY;
  if (xbee.log) fprintf(xbee.log,"libxbee: Stopping...\n");

  /* if the api mode was not 2 to begin with then put it back */
  if (xbee.oldAPI == 2) {
    ret = 0;
  } else {
    int to = 5;

    con = xbee_newcon('I',xbee_localAT);
    xbee_senddata(con,"AP%c",xbee.oldAPI);

    pkt = NULL;

    while (!pkt && to--) {
      pkt = xbee_getpacketwait(con);
    }
    if (pkt) {
      ret = pkt->status;
      Xfree(pkt);
    }
    xbee_endcon(con);
  }

  /* stop listening for data... either after timeout or next char read which ever is first */
  xbee.listenrun = 0;
  xbee_thread_kill(xbee.listent,0);
  /* xbee_* functions may no longer run... */
  xbee_ready = 0;

  if (xbee.log) fflush(xbee.log);

  /* nullify everything */

  /* free all connections */
  con = xbee.conlist;
  xbee.conlist = NULL;
  while (con) {
    ncon = con->next;
    Xfree(con);
    con = ncon;
  }

  /* free all packets */
  xbee.pktlast = NULL;
  pkt = xbee.pktlist;
  xbee.pktlist = NULL;
  while (pkt) {
    npkt = pkt->next;
    Xfree(pkt);
    pkt = npkt;
  }

  /* destroy mutexes */
  xbee_mutex_destroy(xbee.conmutex);
  xbee_mutex_destroy(xbee.pktmutex);
  xbee_mutex_destroy(xbee.sendmutex);

  /* close the serial port */
  Xfree(xbee.path);
#ifdef __GNUC__ /* ---- */
  if (xbee.tty) fclose(xbee.tty);
  if (xbee.ttyfd) close(xbee.ttyfd);
#else /* -------------- */
  if (xbee.tty) CloseHandle(xbee.tty);
#endif /* ------------- */

  /* close log and tty */
  if (xbee.log) {
    fprintf(xbee.log,"libxbee: Stopped! (%s)\n",xbee_svn_version());
    fflush(xbee.log);
    fclose(xbee.log);
  }

  /* wipe everything else... */
  memset(&xbee,0,sizeof(xbee));

  return ret;
}

/* #################################################################
   xbee_setup
   opens xbee serial port & creates xbee listen thread
   the xbee must be configured for API mode 2
   THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
int xbee_setup(char *path, int baudrate) {
  return xbee_setuplogAPI(path,baudrate,0,0,0);
}
int xbee_setuplog(char *path, int baudrate, int logfd) {
  return xbee_setuplogAPI(path,baudrate,logfd,0,0);
}
int xbee_setupAPI(char *path, int baudrate, char cmdSeq, int cmdTime) {
  return xbee_setuplogAPI(path,baudrate,0,cmdSeq,cmdTime);
}
int xbee_setuplogAPI(char *path, int baudrate, int logfd, char cmdSeq, int cmdTime) {
#ifdef __GNUC__ /* ---- */
  struct flock fl;
  struct termios tc;
  speed_t chosenbaud;
#else /* -------------- */
  int chosenbaud;
  DCB tc;
  int evtMask;
  COMMTIMEOUTS timeouts;
#endif /* ------------- */
  t_info info;

  memset(&xbee,0,sizeof(xbee));

#ifdef DEBUG
  /* logfd or stderr */
  xbee.logfd = ((logfd)?logfd:2);
#else
  xbee.logfd = logfd;
```

```c
#endif
  if (xbee.logfd) {
    xbee.log = fdopen(xbee.logfd,"w");
    if (!xbee.log) {
      /* errno == 9 is bad file descriptor (probrably not provided) */
      if (errno != 9) perror("xbee_setup(): Failed opening logfile");
      xbee.logfd = 0;
    } else {
      /* set to line buffer - ensure lines are written to file when complete */
#ifdef __GNUC__ /* ---- */
      setvbuf(xbee.log,NULL,_IOLBF,BUFSIZ);
#else /* -------------- */
      /* Win32 is rubbish... so we have to completely disable buffering... */
      setvbuf(xbee.log,NULL,_IONBF,BUFSIZ);
#endif /* ------------- */
    }
  }

  if (xbee.log) fprintf(xbee.log,"libxbee: Starting (%s)...\n",xbee_svn_version());

#ifdef __GNUC__ /* ---- */
  /* select the baud rate */
  switch (baudrate) {
  case 1200:  chosenbaud = B1200;   break;
  case 2400:  chosenbaud = B2400;   break;
  case 4800:  chosenbaud = B4800;   break;
  case 9600:  chosenbaud = B9600;   break;
  case 19200: chosenbaud = B19200;  break;
  case 38400: chosenbaud = B38400;  break;
  case 57600: chosenbaud = B57600;  break;
  case 115200:chosenbaud = B115200; break;
  default:
    fprintf(stderr,"%s(): Unknown or incompatiable baud rate specified... (%d)\n",__FUNCTION__,baudrate);
    return -1;
  };
#endif /* ------------- */

  /* setup the connection stuff */
  xbee.conlist = NULL;

  /* setup the packet stuff */
  xbee.pktlist = NULL;
  xbee.pktlast = NULL;
  xbee.pktcount = 0;
  xbee.listenrun = 1;

  /* setup the mutexes */
  if (xbee_mutex_init(xbee.conmutex)) {
    perror("xbee_setup():xbee_mutex_init(conmutex)");
    return -1;
  }
  if (xbee_mutex_init(xbee.pktmutex)) {
    perror("xbee_setup():xbee_mutex_init(pktmutex)");
    xbee_mutex_destroy(xbee.conmutex);
    return -1;
  }
  if (xbee_mutex_init(xbee.sendmutex)) {
    perror("xbee_setup():xbee_mutex_init(sendmutex)");
    xbee_mutex_destroy(xbee.conmutex);
    xbee_mutex_destroy(xbee.pktmutex);
    return -1;
  }

  /* take a copy of the XBee device path */
  if ((xbee.path = Xmalloc(sizeof(char) * (strlen(path) + 1))) == NULL) {
    perror("xbee_setup():Xmalloc(path)");
    xbee_mutex_destroy(xbee.conmutex);
    xbee_mutex_destroy(xbee.pktmutex);
    xbee_mutex_destroy(xbee.sendmutex);
    return -1;
  }
  strcpy(xbee.path,path);

#ifdef __GNUC__ /* ---- */
  /* open the serial port as a file descriptor */
  if ((xbee.ttyfd = open(path,O_RDWR | O_NOCTTY | O_NONBLOCK)) == -1) {
    perror("xbee_setup():open()");
    xbee_mutex_destroy(xbee.conmutex);
    xbee_mutex_destroy(xbee.pktmutex);
    xbee_mutex_destroy(xbee.sendmutex);
    Xfree(xbee.path);
    return -1;
  }

  /* lock the file */
  fl.l_type = F_WRLCK | F_RDLCK;
  fl.l_whence = SEEK_SET;
  fl.l_start = 0;
  fl.l_len = 0;
  fl.l_pid = getpid();
  if (fcntl(xbee.ttyfd, F_SETLK, &fl) == -1) {
    perror("xbee_setup():fcntl()");
    xbee_mutex_destroy(xbee.conmutex);
    xbee_mutex_destroy(xbee.pktmutex);
    xbee_mutex_destroy(xbee.sendmutex);
    Xfree(xbee.path);
    close(xbee.ttyfd);
    return -1;
  }

  /* open the serial port as a FILE* */
  if ((xbee.tty = fdopen(xbee.ttyfd,"r+")) == NULL) {
    perror("xbee_setup():fdopen()");
    xbee_mutex_destroy(xbee.conmutex);
    xbee_mutex_destroy(xbee.pktmutex);
    xbee_mutex_destroy(xbee.sendmutex);
    Xfree(xbee.path);
    close(xbee.ttyfd);
    return -1;
  }

  /* flush the serial port */
  fflush(xbee.tty);

  /* disable buffering */
  setvbuf(xbee.tty,NULL,_IONBF,BUFSIZ);

  /* setup the baud rate and other io attributes */
  tcgetattr(xbee.ttyfd, &tc);
  /* input flags */
  tc.c_iflag &= ~ IGNBRK;           /* enable ignoring break */
  tc.c_iflag &= ~(IGNPAR | PARMRK); /* disable parity checks */
  tc.c_iflag &= ~ INPCK;            /* disable parity checking */
  tc.c_iflag &= ~ ISTRIP;           /* disable stripping 8th bit */
  tc.c_iflag &= ~(INLCR | ICRNL);   /* disable translating NL <-> CR */
  tc.c_iflag &= ~ IGNCR;            /* disable ignoring CR */
  tc.c_iflag &= ~(IXON | IXOFF);    /* disable XON/XOFF flow control */
  /* output flags */
  tc.c_oflag &= ~ OPOST;            /* disable output processing */
  tc.c_oflag &= ~(ONLCR | OCRNL);   /* disable translating NL <-> CR */
  tc.c_oflag &= ~ OFILL;            /* disable fill characters */
  /* control flags */
  tc.c_cflag |=   CREAD;            /* enable reciever */
  tc.c_cflag &= ~ PARENB;           /* disable parity */
  tc.c_cflag &= ~ CSTOPB;           /* disable 2 stop bits */
  tc.c_cflag &= ~ CSIZE;            /* remove size flag... */
  tc.c_cflag |=   CS8;              /* ...enable 8 bit characters */
  tc.c_cflag |=   HUPCL;            /* enable lower control lines on close - hang up */
  /* local flags */
  tc.c_lflag &= ~ ISIG;             /* disable generating signals */
  tc.c_lflag &= ~ ICANON;           /* disable canonical mode - line by line */
  tc.c_lflag &= ~ ECHO;             /* disable echoing characters */
  tc.c_lflag &= ~ ECHONL;           /* ??? */
  tc.c_lflag &= ~ NOFLSH;           /* disable flushing on SIGINT */
  tc.c_lflag &= ~ IEXTEN;           /* disable input processing */
  /* control characters */
  memset(tc.c_cc,0,sizeof(tc.c_cc));
  /* i/o rates */
  cfsetspeed(&tc, chosenbaud);     /* set i/o baud rate */
  tcsetattr(xbee.ttyfd, TCSANOW, &tc);
  tcflow(xbee.ttyfd, TCOON|TCION); /* enable input & output transmission */
#else /* -------------- */
  /* open the serial port */
  xbee.tty = CreateFile(TEXT(path),
                       GENERIC_READ | GENERIC_WRITE,
                       0,    /* exclusive access */
                       NULL, /* default security attributes */
                       OPEN_EXISTING,
                       FILE_FLAG_OVERLAPPED,
                       NULL);
  if (xbee.tty == INVALID_HANDLE_VALUE) {
    perror("xbee_setup():CreateFile()");
    xbee_mutex_destroy(xbee.conmutex);
    xbee_mutex_destroy(xbee.pktmutex);
    xbee_mutex_destroy(xbee.sendmutex);
    Xfree(xbee.path);
    return -1;
  }

  GetCommState(xbee.tty, &tc);
  tc.BaudRate =          baudrate;
  tc.fBinary =           TRUE;
  tc.fParity =           FALSE;
  tc.fOutxCtsFlow =      FALSE;
  tc.fOutxDsrFlow =      FALSE;
  tc.fDtrControl =       DTR_CONTROL_DISABLE;
  tc.fDsrSensitivity =   FALSE;
  tc.fTXContinueOnXoff = FALSE;
  tc.fOutX =             FALSE;
  tc.fInX =              FALSE;
  tc.fErrorChar =        FALSE;
  tc.fNull =             FALSE;
  tc.fRtsControl =       RTS_CONTROL_DISABLE;
  tc.fAbortOnError =     FALSE;
  tc.ByteSize =          8;
  tc.Parity =            NOPARITY;
  tc.StopBits =          ONESTOPBIT;
  SetCommState(xbee.tty, &tc);

  timeouts.ReadIntervalTimeout = MAXDWORD;
  timeouts.ReadTotalTimeoutMultiplier = 0;
  timeouts.ReadTotalTimeoutConstant = 0;
  timeouts.WriteTotalTimeoutMultiplier = 0;
  timeouts.WriteTotalTimeoutConstant = 0;
  SetCommTimeouts(xbee.tty, &timeouts);

  GetCommMask(xbee.tty, &evtMask);
  evtMask |= EV_RXCHAR;
  SetCommMask(xbee.tty, evtMask);
#endif /* ------------- */

  /* when xbee_end() is called, if this is not 2 then ATAP will be set to this value */
  xbee.oldAPI = 2;
  xbee.cmdSeq = cmdSeq;
  xbee.cmdTime = cmdTime;
  if (xbee.cmdSeq && xbee.cmdTime) {
    if (xbee_startAPI()) {
      if (xbee.log) {
        xbee_log("Couldn't communicate with XBee...");
      }
      xbee_mutex_destroy(xbee.conmutex);
      xbee_mutex_destroy(xbee.pktmutex);
      xbee_mutex_destroy(xbee.sendmutex);
      Xfree(xbee.path);
#ifdef __GNUC__ /* ---- */
      close(xbee.ttyfd);
#endif /* ------------- */
      fclose(xbee.tty);
      return -1;
    }
  }

  /* allow the listen thread to start */
  xbee_ready = -1;

  /* can start xbee_listen thread now */
  if (xbee_thread_create(xbee.listent,xbee_listen_wrapper,info)) {
    perror("xbee_setup():xbee_thread_create()");
    xbee_mutex_destroy(xbee.conmutex);
    xbee_mutex_destroy(xbee.pktmutex);
    xbee_mutex_destroy(xbee.sendmutex);
    Xfree(xbee.path);
#ifdef __GNUC__ /* ---- */
    close(xbee.ttyfd);
#endif /* ------------- */
    fclose(xbee.tty);
    return -1;
  }

  usleep(100);
  while (xbee_ready != -2) {
    usleep(100);
    if (xbee.log) {
      xbee_log("Waiting for xbee_listen() to be ready...");
    }
  }

  /* allow other functions to be used! */
  xbee_ready = 1;

  if (xbee.log) fprintf(xbee.log,"libxbee: Started!\n");

  return 0;
}

```

```c
/* ##################################################################
   xbee_con
   produces a connection to the specified device and frameID
   if a connection had already been made, then this connection will be returned */
xbee_con *xbee_newcon(unsigned char frameID, xbee_types type, ...) {
  xbee_con *con, *ocon;
  unsigned char tAddr[8];
  va_list ap;
  int t;
  int i;

  ISREADY;

  if (!type || type == xbee_unknown) type = xbee_localAT; /* default to local AT */
  else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */

  va_start(ap,type);
  /* if: 64 bit address expected (2 ints) */
  if ((type == xbee_64bitRemoteAT) ||
      (type == xbee_64bitData) ||
      (type == xbee_64bitIO)) {
    t = va_arg(ap, int);
    tAddr[0] = (t >> 24) & 0xFF;
    tAddr[1] = (t >> 16) & 0xFF;
    tAddr[2] = (t >>  8) & 0xFF;
    tAddr[3] = (t      ) & 0xFF;
    t = va_arg(ap, int);
    tAddr[4] = (t >> 24) & 0xFF;
    tAddr[5] = (t >> 16) & 0xFF;
    tAddr[6] = (t >>  8) & 0xFF;
    tAddr[7] = (t      ) & 0xFF;

    /* if: 16 bit address expected (1 int) */
  } else if ((type == xbee_16bitRemoteAT) ||
             (type == xbee_16bitData) ||
             (type == xbee_16bitIO)) {
    t = va_arg(ap, int);
    tAddr[0] = (t >>  8) & 0xFF;
    tAddr[1] = (t      ) & 0xFF;
    tAddr[2] = 0;
    tAddr[3] = 0;
    tAddr[4] = 0;
    tAddr[5] = 0;
    tAddr[6] = 0;
    tAddr[7] = 0;

    /* otherwise clear the address */
  } else {
    memset(tAddr,0,8);
  }
  va_end(ap);

  /* lock the connection mutex */
  xbee_mutex_lock(xbee.conmutex);

  /* are there any connections? */
  if (xbee.conlist) {
    con = xbee.conlist;
    while (con) {
      /* if: after a modemStatus, and the types match! */
      if ((type == xbee_modemStatus) &&
          (con->type == type)) {
        xbee_mutex_unlock(xbee.conmutex);
        return con;

        /* if: after a txStatus and frameIDs match! */
      } else if ((type == xbee_txStatus) &&
                 (con->type == type) &&
                 (frameID == con->frameID)) {
        xbee_mutex_unlock(xbee.conmutex);
        return con;

        /* if: after a localAT, and the frameIDs match! */
      } else if ((type == xbee_localAT) &&
                 (con->type == type) &&
                 (frameID == con->frameID)) {
        xbee_mutex_unlock(xbee.conmutex);
        return con;

        /* if: connection types match, the frameIDs match, and the addresses match! */
      } else if ((type == con->type) &&
                 (frameID == con->frameID) &&
                 (!memcmp(tAddr,con->tAddr,8))) {
        xbee_mutex_unlock(xbee.conmutex);
        return con;
      }

      /* if there are more, move along, dont want to loose that last item! */
      if (con->next == NULL) break;
      con = con->next;
    }

    /* keep hold of the last connection... we will need to link it up later */
    ocon = con;
  }

  /* create a new connection and set its attributes */
  con = Xcalloc(sizeof(xbee_con));
  con->type = type;
  /* is it a 64bit connection? */
  if ((type == xbee_64bitRemoteAT) ||
      (type == xbee_64bitData) ||
      (type == xbee_64bitIO)) {
    con->tAddr64 = TRUE;
  }
  con->atQueue = 0; /* queue AT commands? */
  con->txDisableACK = 0; /* disable ACKs? */
  con->txBroadcast = 0; /* broadcast? */
  con->frameID = frameID;
  memcpy(con->tAddr,tAddr,8); /* copy in the remote address */

  if (xbee.log) {
    switch(type) {
    case xbee_localAT:
      xbee_log("New local AT connection!");
      break;
    case xbee_16bitRemoteAT:
    case xbee_64bitRemoteAT:
      xbee_logc("New %d-bit remote AT connection! (to: ",(con->tAddr64?64:16));
      for (i=0;i<(con->tAddr64?8:2);i++) {
        fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
      }
      fprintf(xbee.log,")\n");
      break;
    case xbee_16bitData:
    case xbee_64bitData:
      xbee_logc("New %d-bit data connection! (to: ",(con->tAddr64?64:16));
      for (i=0;i<(con->tAddr64?8:2);i++) {
        fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
      }
      fprintf(xbee.log,")\n");
      break;
    case xbee_16bitIO:
    case xbee_64bitIO:
      xbee_logc("New %d-bit IO connection! (to: ",(con->tAddr64?64:16));
      for (i=0;i<(con->tAddr64?8:2);i++) {
        fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
      }
      fprintf(xbee.log,")\n");
      break;
    case xbee_txStatus:
      xbee_log("New Tx status connection!");
      break;
    case xbee_modemStatus:
      xbee_log("New modem status connection!");
      break;
    case xbee_unknown:
    default:
      xbee_log("New unknown connection!");
    }
  }

  /* make it the last in the list */
  con->next = NULL;
  /* add it to the list */
  if (xbee.conlist) {
    ocon->next = con;
  } else {
    xbee.conlist = con;
  }

  /* unlock the mutex */
  xbee_mutex_unlock(xbee.conmutex);
  return con;
}

/* #####################################################################
   xbee_conflush
   removes any packets that have been collected for the specified
   connection */
```

```c
void xbee_flushcon(xbee_con *con) {
  xbee_pkt *r, *p, *n;

  /* lock the packet mutex */
  xbee_mutex_lock(xbee.pktmutex);

  /* if: there are packets */
  if ((p = xbee.pktlist) != NULL) {
    r = NULL;
    /* get all packets for this connection */
    do {
      /* does the packet match the connection? */
      if (xbee_matchpktcon(p,con)) {
        /* if it was the first packet */
        if (!r) {
          /* move the chain along */
          xbee.pktlist = p->next;
        } else {
          /* otherwise relink the list */
          r->next = p->next;
        }
        xbee.pktcount--;

        /* free this packet! */
        n = p->next;
        Xfree(p);
        /* move on */
        p = n;
      } else {
        /* move on */
        r = p;
        p = p->next;
      }
    } while (p);
    xbee.pktlast = r;
  }

  /* unlock the packet mutex */
  xbee_mutex_unlock(xbee.pktmutex);
}

/* #################################################################
   xbee_endcon
   close the unwanted connection
   free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
void xbee_endcon2(xbee_con **con) {
  xbee_con *t, *u;

  /* lock the connection mutex */
  xbee_mutex_lock(xbee.conmutex);

  u = t = xbee.conlist;
  while (t && t != *con) {
    u = t;
    t = t->next;
  }
  if (!t) {
    /* invalid connection given... */
    if (xbee.log) {
      xbee_log("Attempted to close invalid connection...");
    }
    /* unlock the connection mutex */
    xbee_mutex_unlock(xbee.conmutex);
    return;
  }
  /* extract this connection from the list */
  u->next = (*con)->next;
  if (*con == xbee.conlist) xbee.conlist = NULL;

  /* unlock the connection mutex */
  xbee_mutex_unlock(xbee.conmutex);

  /* remove all packets for this connection */
  xbee_flushcon(*con);

  /* free the connection! */
  Xfree(*con);
}

/* #################################################################
   xbee_senddata
   send the specified data to the provided connection */
int xbee_senddata(xbee_con *con, char *format, ...) {
  int ret;
  va_list ap;

  ISREADY;

  /* xbee_vsenddata() wants a va_list... */
  va_start(ap, format);
  /* hand it over :) */
  ret = xbee_vsenddata(con,format,ap);
  va_end(ap);
  return ret;
}

int xbee_vsenddata(xbee_con *con, char *format, va_list ap) {
  unsigned char data[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
  int length;

  ISREADY;

  /* make up the data and keep the length, its possible there are nulls in there */
  length = vsnprintf((char *)data,128,format,ap);

  /* hand it over :) */
  return xbee_nsenddata(con,(char *)data,length);
}

int xbee_nsenddata(xbee_con *con, char *data, int length) {
  t_data *pkt;
  int i;
  unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */

  ISREADY;

  if (!con) return -1;
  if (con->type == xbee_unknown) return -1;
  if (length > 127) return -1;


  if (xbee.log) {
    xbee_log("--== TX Packet ============--");
    xbee_logc("Connection Type: ");
    switch (con->type) {
    case xbee_unknown:       fprintf(xbee.log,"Unknown\n"); break;
    case xbee_localAT:       fprintf(xbee.log,"Local AT\n"); break;
    case xbee_remoteAT:      fprintf(xbee.log,"Remote AT\n"); break;
    case xbee_16bitRemoteAT: fprintf(xbee.log,"Remote AT (16-bit)\n"); break;
    case xbee_64bitRemoteAT: fprintf(xbee.log,"Remote AT (64-bit)\n"); break;
    case xbee_16bitData:     fprintf(xbee.log,"Data (16-bit)\n"); break;
    case xbee_64bitData:     fprintf(xbee.log,"Data (64-bit)\n"); break;
    case xbee_16bitIO:       fprintf(xbee.log,"IO (16-bit)\n"); break;
    case xbee_64bitIO:       fprintf(xbee.log,"IO (64-bit)\n"); break;
    case xbee_txStatus:      fprintf(xbee.log,"Tx Status\n"); break;
    case xbee_modemStatus:   fprintf(xbee.log,"Modem Status\n"); break;
    }
    xbee_logc("Destination: ");
    for (i=0;i<(con->tAddr64?8:2);i++) {
      fprintf(xbee.log,(i?":%02X":"%02X"),con->tAddr[i]);
    }
    fprintf(xbee.log,"\n");
    xbee_log("Length: %d",length);
    for (i=0;i<length;i++) {
      xbee_logc("%3d | 0x%02X ",i,data[i]);
      if ((data[i] > 32) && (data[i] < 127)) {
        fprintf(xbee.log,"'%c'\n",data[i]);
      } else{
        fprintf(xbee.log," _\n");
      }
    }
  }

  /* ########################################### */
  /* if: local AT */
  if (con->type == xbee_localAT) {
    /* AT commands are 2 chars long (plus optional parameter) */
    if (length < 2) return -1;

    /* use the command? */
    buf[0] = ((!con->atQueue)?0x08:0x09);
    buf[1] = con->frameID;

    /* copy in the data */
    for (i=0;i<length;i++) {
      buf[i+2] = data[i];
    }

    /* setup the packet */
    pkt = xbee_make_pkt(buf,i+2);
    /* send it on */
    xbee_send_pkt(pkt);

    return 0;

    /* ########################################## */
    /* if: remote AT */
  } else if ((con->type == xbee_16bitRemoteAT) ||
             (con->type == xbee_64bitRemoteAT)) {
    if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter) */
    buf[0] = 0x17;
    buf[1] = con->frameID;

    /* copy in the relevant address */
    if (con->tAddr64) {
      memcpy(&buf[2],con->tAddr,8);
      buf[10] = 0xFF;
      buf[11] = 0xFE;
    } else {
      memset(&buf[2],0,8);
      memcpy(&buf[10],con->tAddr,2);
    }
    /* queue the command? */
    buf[12] = ((!con->atQueue)?0x02:0x00);

    /* copy in the data */
    for (i=0;i<length;i++) {
      buf[i+13] = data[i];
    }

    /* setup the packet */
    pkt = xbee_make_pkt(buf,i+13);
    /* send it on */
    xbee_send_pkt(pkt);

    return 0;

    /* ########################################## */
    /* if: 16 or 64bit Data */
  } else if ((con->type == xbee_16bitData) ||
             (con->type == xbee_64bitData)) {
    int offset;

    /* if: 16bit Data */
    if (con->type == xbee_16bitData) {
      buf[0] = 0x01;
      offset = 5;
      /* copy in the address */
      memcpy(&buf[2],con->tAddr,2);

      /* if: 64bit Data */
    } else { /* 64bit Data */
      buf[0] = 0x00;
      offset = 11;
      /* copy in the address */
      memcpy(&buf[2],con->tAddr,8);
    }

    /* copy frameID */
    buf[1] = con->frameID;

    /* disable ack? broadcast? */
    buf[offset-1] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);

    /* copy in the data */
    for (i=0;i<length;i++) {
      buf[i+offset] = data[i];
    }

    /* setup the packet */
    pkt = xbee_make_pkt(buf,i+offset);
    /* send it on */
    xbee_send_pkt(pkt);

    return 0;

    /* ########################################## */
    /* if: I/O */
  } else if ((con->type == xbee_64bitIO) ||
             (con->type == xbee_16bitIO)) {
    /* not currently implemented... is it even allowed? */
    if (xbee.log) {
      fprintf(xbee.log,"******* TODO ********\n");
    }
  }

  return -2;
}

/* ##################################################################
   xbee_getpacket
   retrieves the next packet destined for the given connection
   once the packet has been retrieved, it is removed for the list! */
xbee_pkt *xbee_getpacketwait(xbee_con *con) {
  xbee_pkt *p;
  int i;

  /* 50ms * 20 = 1 second */
  for (i = 0; i < 20; i++) {
    p = xbee_getpacket(con);
    if (p) break;
    usleep(50000); /* 50ms */
  }

  return p;
}
xbee_pkt *xbee_getpacket(xbee_con *con) {
  xbee_pkt *l, *p, *q;
  /*if (xbee.log) {
    xbee_log("--== Get Packet ==========--");
    }*/

  /* lock the packet mutex */
  xbee_mutex_lock(xbee.pktmutex);

  /* if: there are no packets */
  if ((p = xbee.pktlist) == NULL) {
    xbee_mutex_unlock(xbee.pktmutex);
    /*if (xbee.log) {
      xbee_log("No packets avaliable...");
      }*/
    return NULL;
  }

  l = NULL;
  q = NULL;
  /* get the first avaliable packet for this connection */
  do {
    /* does the packet match the connection? */
    if (xbee_matchpktcon(p,con)) {
      q = p;
      break;
    }
    /* move on */
    l = p;
    p = p->next;
  } while (p);

  /* if: no packet was found */
  if (!q) {
    xbee_mutex_unlock(xbee.pktmutex);
    /*if (xbee.log) {
      xbee_log("No packets avaliable (for connection)...");
      }*/
    return NULL;
  }

  /* if it was the first packet */
  if (l) {
    /* relink the list */
    l->next = p->next;
    if (!l->next) xbee.pktlast = l;
  } else {
    /* move the chain along */
    xbee.pktlist = p->next;
    if (!xbee.pktlist) {
      xbee.pktlast = NULL;
    } else if (!xbee.pktlist->next) {
      xbee.pktlast = xbee.pktlist;
    }
  }
  xbee.pktcount--;

  /* unlink this packet from the chain! */
  q->next = NULL;

  if (xbee.log) {
    xbee_log("--== Get Packet ==========--");
    xbee_log("Got a packet");
    xbee_log("Packets left: %d",xbee.pktcount);
```

```c
  }

  /* unlock the packet mutex */
  xbee_mutex_unlock(xbee.pktmutex);

  /* and return the packet (must be free’d by caller!) */
  return q;
}

/* #################################################################
   xbee_matchpktcon - INTERNAL
   checks if the packet matches the connection */
static int xbee_matchpktcon(xbee_pkt *pkt, xbee_con *con) {
  /* if: the connection type matches the packet type OR
     the connection is 16/64bit remote AT, and the packet is a remote AT response */
  if ((pkt->type == con->type) || /* -- */
      ((pkt->type == xbee_remoteAT) && /* -- */
       ((con->type == xbee_16bitRemoteAT) ||
        (con->type == xbee_64bitRemoteAT)))) {

    /* if: the packet is modem status OR
       the packet is tx status or AT data and the frame IDs match OR
       the addresses match */
    if (pkt->type == xbee_modemStatus) return 1;

    if ((pkt->type == xbee_txStatus) ||
        (pkt->type == xbee_localAT) ||
        (pkt->type == xbee_remoteAT)) {
      if (pkt->frameID == con->frameID) {
        return 1;
      }
    } else if (pkt->sAddr64 && !memcmp(pkt->Addr64,con->tAddr,8)) {
      return 1;
    } else if (!pkt->sAddr64 && !memcmp(pkt->Addr16,con->tAddr,2)) {
      return 1;
    }
  }
  return 0;
}

/* #################################################################
   xbee_parse_io - INTERNAL
   parses the data given into the packet io information */
static int xbee_parse_io(xbee_pkt *p, unsigned char *d, int maskOffset, int sampleOffset, int sample) {
  xbee_sample *s = &(p->IOdata[sample]);

  /* copy in the I/O data mask */
  s->IOmask = (((d[maskOffset]<<8) | d[maskOffset + 1]) & 0x7FFF);

  /* copy in the digital I/O data */
  s->IOdigital = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x01FF);

  /* advance over the digital data, if its there */
  sampleOffset += ((s->IOmask & 0x01FF)?2:0);

  /* copy in the analog I/O data */
  if (s->IOmask & 0x0200) {
    s->IOanalog[0] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
    sampleOffset+=2;
  }
  if (s->IOmask & 0x0400) {
    s->IOanalog[1] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
    sampleOffset+=2;
  }
  if (s->IOmask & 0x0800) {
    s->IOanalog[2] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
    sampleOffset+=2;
  }
  if (s->IOmask & 0x1000) {
    s->IOanalog[3] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
    sampleOffset+=2;
  }
  if (s->IOmask & 0x2000) {
    s->IOanalog[4] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
    sampleOffset+=2;
  }
  if (s->IOmask & 0x4000) {
    s->IOanalog[5] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
    sampleOffset+=2;
  }

  if (xbee.log) {
    if (s->IOmask & 0x0001)
      xbee_log("Digital 0: %c",((s->IOdigital & 0x0001)?’1’:’0’));
    if (s->IOmask & 0x0002)
```

```c
      xbee_log("Digital 1: %c",((s->IOdigital & 0x0002)?'1':'0'));
    if (s->IOmask & 0x0004)
      xbee_log("Digital 2: %c",((s->IOdigital & 0x0004)?'1':'0'));
    if (s->IOmask & 0x0008)
      xbee_log("Digital 3: %c",((s->IOdigital & 0x0008)?'1':'0'));
    if (s->IOmask & 0x0010)
      xbee_log("Digital 4: %c",((s->IOdigital & 0x0010)?'1':'0'));
    if (s->IOmask & 0x0020)
      xbee_log("Digital 5: %c",((s->IOdigital & 0x0020)?'1':'0'));
    if (s->IOmask & 0x0040)
      xbee_log("Digital 6: %c",((s->IOdigital & 0x0040)?'1':'0'));
    if (s->IOmask & 0x0080)
      xbee_log("Digital 7: %c",((s->IOdigital & 0x0080)?'1':'0'));
    if (s->IOmask & 0x0100)
      xbee_log("Digital 8: %c",((s->IOdigital & 0x0100)?'1':'0'));
    if (s->IOmask & 0x0200)
      xbee_log("Analog  0: %d (~%.2fv)\n",s->IOanalog[0],(3.3/1023)*s->IOanalog[0]);
    if (s->IOmask & 0x0400)
      xbee_log("Analog  1: %d (~%.2fv)\n",s->IOanalog[1],(3.3/1023)*s->IOanalog[1]);
    if (s->IOmask & 0x0800)
      xbee_log("Analog  2: %d (~%.2fv)\n",s->IOanalog[2],(3.3/1023)*s->IOanalog[2]);
    if (s->IOmask & 0x1000)
      xbee_log("Analog  3: %d (~%.2fv)\n",s->IOanalog[3],(3.3/1023)*s->IOanalog[3]);
    if (s->IOmask & 0x2000)
      xbee_log("Analog  4: %d (~%.2fv)\n",s->IOanalog[4],(3.3/1023)*s->IOanalog[4]);
    if (s->IOmask & 0x4000)
      xbee_log("Analog  5: %d (~%.2fv)\n",s->IOanalog[5],(3.3/1023)*s->IOanalog[5]);
  }

  return sampleOffset;
}

/* ################################################################
   xbee_listen_stop
   stops the listen thread after the current packet has been processed */
void xbee_listen_stop(void) {
  xbee.listenrun = 0;
}

/* ################################################################
   xbee_listen_wrapper - INTERNAL
   the xbee_listen wrapper. Prints an error when xbee_listen ends */
static void xbee_listen_wrapper(t_info *info) {
  int ret;
  /* just falls out if the proper 'go-ahead' isn't given */
  if (xbee_ready != -1) return;
  /* now allow the parent to continue */
  xbee_ready = -2;

#ifdef _WIN32 /* ---- */
  /* win32 requires this delay... no idea why */
  usleep(1000000);
#endif /* ----------- */

  while (xbee.listenrun) {
    info->i = -1;
    ret = xbee_listen(info);
    if (!xbee.listenrun) break;
    if (xbee.log) {
      xbee_log("xbee_listen() returned [%d]... Restarting in 250ms!",ret);
    }
    usleep(25000);
  }
}

/* xbee_listen - INTERNAL
   the xbee xbee_listen thread
   reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
static int xbee_listen(t_info *info) {
  unsigned char c, t, d[1024];
  unsigned int l, i, chksum, o;
  int j;
  xbee_pkt *p, *q;
  xbee_con *con;
  int hasCon;

  /* just falls out if the proper 'go-ahead' isn't given */
  if (info->i != -1) return -1;
  /* do this forever :) */
  while (xbee.listenrun) {
    /* wait for a valid start byte */
    if (xbee_getrawbyte() != 0x7E) continue;
    if (!xbee.listenrun) return 0;

    if (xbee.log) {
      xbee_log("--== RX Packet ===========--");
      xbee_log("Got a packet!...");
    }

    /* get the length */
    l = xbee_getbyte() << 8;
    l += xbee_getbyte();

    /* check it is a valid length... */
    if (!l) {
      if (xbee.log) {
        xbee_log("Recived zero length packet!");
      }
      continue;
    }
    if (l > 100) {
      if (xbee.log) {
        xbee_log("Recived oversized packet! Length: %d",l - 1);
      }
    }
    if (l > sizeof(d) - 1) {
      if (xbee.log) {
        xbee_log("Recived packet larger than buffer! Discarding...");
      }
      continue;
    }

    if (xbee.log) {
      xbee_log("Length: %d",l - 1);
    }

    /* get the packet type */
    t = xbee_getbyte();

    /* start the checksum */
    chksum = t;

    /* suck in all the data */
    for (i = 0; l > 1 && i < 128; l--, i++) {
      /* get an unescaped byte */
      c = xbee_getbyte();
      d[i] = c;
      chksum += c;
      if (xbee.log) {
        xbee_logc("%3d | 0x%02X | ",i,c);
        if ((c > 32) && (c < 127)) fprintf(xbee.log,"'%c'",c); else fprintf(xbee.log," _ ");

        if ((t == 0x80 && i == (8 + 2)) || /* 64-bit Data packet */
            (t == 0x81 && i == (2 + 2))) { /* 16-bit Data packet */
          /* mark the beginning of the 'data' bytes */
          fprintf(xbee.log,"   <-- data starts");
        }

        fprintf(xbee.log,"\n");
      }
    }
    i--; /* it went up too many times!... */

    /* add the checksum */
    chksum += xbee_getbyte();

    /* check if the whole packet was recieved, or something else occured... unlikely... */
    if (l>1) {
      if (xbee.log) {
        xbee_log("Didn't get whole packet... :(");
      }
      continue;
    }

    /* check the checksum */
    if ((chksum & 0xFF) != 0xFF) {
      if (xbee.log) {
        xbee_log("Invalid Checksum: 0x%02X",chksum);
      }
      continue;
    }

    /* make a new packet */
    p = Xcalloc(sizeof(xbee_pkt));
    q = NULL;
    p->datalen = 0;

    /* ########################################## */
    /* if: modem status */
    if (t == 0x8A) {
      if (xbee.log) {
        xbee_log("Packet type: Modem Status (0x8A)");
        xbee_logc("Event: ");
        switch (d[0]) {
        case 0x00: fprintf(xbee.log,"Hardware reset"); break;
        case 0x01: fprintf(xbee.log,"Watchdog timer reset"); break;
        case 0x02: fprintf(xbee.log,"Associated"); break;
        case 0x03: fprintf(xbee.log,"Disassociated"); break;
        case 0x04: fprintf(xbee.log,"Synchronization lost"); break;
        case 0x05: fprintf(xbee.log,"Coordinator realignment"); break;
        case 0x06: fprintf(xbee.log,"Coordinator started"); break;
        }
        fprintf(xbee.log,"... (0x%02X)\n",d[0]);
      }
      p->type = xbee_modemStatus;

      p->sAddr64 = FALSE;
      p->dataPkt = FALSE;
      p->txStatusPkt = FALSE;
      p->modemStatusPkt = TRUE;
      p->remoteATPkt = FALSE;
      p->IOPkt = FALSE;

      /* modem status can only ever give 1 'data' byte */
      p->datalen = 1;
      p->data[0] = d[0];

      /* ########################################### */
      /* if: local AT response */
    } else if (t == 0x88) {
      if (xbee.log) {
        xbee_log("Packet type: Local AT Response (0x88)");
        xbee_log("FrameID: 0x%02X",d[0]);
        xbee_log("AT Command: %c%c",d[1],d[2]);
        xbee_logc("Status: ");
        if (d[3] == 0) fprintf(xbee.log,"OK");
        else if (d[3] == 1) fprintf(xbee.log,"Error");
        else if (d[3] == 2) fprintf(xbee.log,"Invalid Command");
        else if (d[3] == 3) fprintf(xbee.log,"Invalid Parameter");
        fprintf(xbee.log," (0x%02X)\n",d[3]);
      }
      p->type = xbee_localAT;

      p->sAddr64 = FALSE;
      p->dataPkt = FALSE;
      p->txStatusPkt = FALSE;
      p->modemStatusPkt = FALSE;
      p->remoteATPkt = FALSE;
      p->IOPkt = FALSE;

      p->frameID = d[0];
      p->atCmd[0] = d[1];
      p->atCmd[1] = d[2];

      p->status = d[3];

      /* copy in the data */
      p->datalen = i-3;
      for (;i>3;i--) p->data[i-4] = d[i];

      /* ########################################### */
      /* if: remote AT response */
    } else if (t == 0x97) {
      if (xbee.log) {
        xbee_log("Packet type: Remote AT Response (0x97)");
        xbee_log("FrameID: 0x%02X",d[0]);
        xbee_logc("64-bit Address: ");
        for (j=0;j<8;j++) {
          fprintf(xbee.log,(j?":%02X":"%02X"),d[1+j]);
        }
        fprintf(xbee.log,"\n");
        xbee_logc("16-bit Address: ");
        for (j=0;j<2;j++) {
          fprintf(xbee.log,(j?":%02X":"%02X"),d[9+j]);
        }
        fprintf(xbee.log,"\n");
        xbee_log("AT Command: %c%c",d[11],d[12]);
        xbee_logc("Status: ");
        if (d[13] == 0) fprintf(xbee.log,"OK");
        else if (d[13] == 1) fprintf(xbee.log,"Error");
        else if (d[13] == 2) fprintf(xbee.log,"Invalid Command");
        else if (d[13] == 3) fprintf(xbee.log,"Invalid Parameter");
        else if (d[13] == 4) fprintf(xbee.log,"No Response");
        fprintf(xbee.log," (0x%02X)\n",d[13]);
      }
      p->type = xbee_remoteAT;

      p->sAddr64 = FALSE;
      p->dataPkt = FALSE;
      p->txStatusPkt = FALSE;
      p->modemStatusPkt = FALSE;
      p->remoteATPkt = TRUE;
      p->IOPkt = FALSE;

      p->frameID = d[0];

      p->Addr64[0] = d[1];
      p->Addr64[1] = d[2];
      p->Addr64[2] = d[3];
      p->Addr64[3] = d[4];
      p->Addr64[4] = d[5];
      p->Addr64[5] = d[6];
      p->Addr64[6] = d[7];
      p->Addr64[7] = d[8];

      p->Addr16[0] = d[9];
      p->Addr16[1] = d[10];

      p->atCmd[0] = d[11];
      p->atCmd[1] = d[12];

      p->status = d[13];

      p->samples = 1;

      if (p->status == 0x00 && p->atCmd[0] == 'I' && p->atCmd[1] == 'S') {
        /* parse the io data */
        if (xbee.log) xbee_log("--- Sample -----------------");
        xbee_parse_io(p, d, 15, 17, 0);
        if (xbee.log) xbee_log("----------------------------");
      } else {
        /* copy in the data */
        p->datalen = i-13;
        for (;i>13;i--) p->data[i-14] = d[i];
      }

      /* ########################################### */
      /* if: TX status */
    } else if (t == 0x89) {
      if (xbee.log) {
        xbee_log("Packet type: TX Status Report (0x89)");
        xbee_log("FrameID: 0x%02X",d[0]);
        xbee_logc("Status: ");
        if (d[1] == 0) fprintf(xbee.log,"Success");
        else if (d[1] == 1) fprintf(xbee.log,"No ACK");
        else if (d[1] == 2) fprintf(xbee.log,"CCA Failure");
        else if (d[1] == 3) fprintf(xbee.log,"Purged");
        fprintf(xbee.log," (0x%02X)\n",d[1]);
      }
      p->type = xbee_txStatus;

      p->sAddr64 = FALSE;
      p->dataPkt = FALSE;
      p->txStatusPkt = TRUE;
      p->modemStatusPkt = FALSE;
      p->remoteATPkt = FALSE;
      p->IOPkt = FALSE;

      p->frameID = d[0];

      p->status = d[1];

      /* never returns data */
      p->datalen = 0;

      /* ########################################### */
      /* if: 16 / 64bit data recieve */
    } else if ((t == 0x80) ||
               (t == 0x81)) {
      int offset;
      if (t == 0x80) { /* 64bit */
        offset = 8;
      } else { /* 16bit */
        offset = 2;
      }
      if (xbee.log) {
        xbee_log("Packet type: %d-bit RX Data (0x%02X)",((t == 0x80)?64:16),t);
        xbee_logc("%d-bit Address: ",((t == 0x80)?64:16));
        for (j=0;j<offset;j++) {
          fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
        }
        fprintf(xbee.log,"\n");
        xbee_log("RSSI: -%ddB",d[offset]);
        if (d[offset + 1] & 0x02) xbee_log("Options: Address Broadcast");
        if (d[offset + 1] & 0x03) xbee_log("Options: PAN Broadcast");
      }
      p->dataPkt = TRUE;
      p->txStatusPkt = FALSE;
      p->modemStatusPkt = FALSE;
      p->remoteATPkt = FALSE;
      p->IOPkt = FALSE;

      if (t == 0x80) { /* 64bit */
        p->type = xbee_64bitData;

        p->sAddr64 = TRUE;

        p->Addr64[0] = d[0];
        p->Addr64[1] = d[1];
        p->Addr64[2] = d[2];
        p->Addr64[3] = d[3];
        p->Addr64[4] = d[4];
        p->Addr64[5] = d[5];
        p->Addr64[6] = d[6];
        p->Addr64[7] = d[7];
      } else { /* 16bit */
        p->type = xbee_16bitData;

        p->sAddr64 = FALSE;

        p->Addr16[0] = d[0];
        p->Addr16[1] = d[1];
      }

      /* save the RSSI / signal strength
         this can be used with printf as:
         printf("-%ddB\n",p->RSSI); */
      p->RSSI = d[offset];

      p->status = d[offset + 1];

      /* copy in the data */
      p->datalen = i-(offset + 1);
      for (;i>offset + 1;i--) p->data[i-(offset + 2)] = d[i];

      /* ########################################### */
      /* if: 16 / 64bit I/O recieve */
    } else if ((t == 0x82) ||
               (t == 0x83)) {
      int offset;
      if (t == 0x82) { /* 64bit */
        p->type = xbee_64bitIO;

        p->sAddr64 = TRUE;

        p->Addr64[0] = d[0];
        p->Addr64[1] = d[1];
        p->Addr64[2] = d[2];
        p->Addr64[3] = d[3];
        p->Addr64[4] = d[4];
        p->Addr64[5] = d[5];
        p->Addr64[6] = d[6];
        p->Addr64[7] = d[7];

        offset = 8;
        p->samples = d[10];
      } else { /* 16bit */
        p->type = xbee_16bitIO;

        p->sAddr64 = FALSE;

        p->Addr16[0] = d[0];
        p->Addr16[1] = d[1];

        offset = 2;
        p->samples = d[4];
      }
      if (p->samples > 1) {
        p = Xrealloc(p, sizeof(xbee_pkt) + (sizeof(xbee_sample) * (p->samples - 1)));
      }
      if (xbee.log) {
        xbee_logc("Packet type: %d-bit RX I/O Data (0x%02X)\n",((t == 0x82)?64:16),t);
        xbee_logc("%d-bit Address: ",((t == 0x82)?64:16));
        for (j = 0; j < offset; j++) {
          fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
        }
        fprintf(xbee.log,"\n");
        xbee_log("RSSI: -%ddB",d[offset]);
        if (d[9] & 0x02) xbee_log("Options: Address Broadcast");
        if (d[9] & 0x02) xbee_log("Options: PAN Broadcast");
        xbee_log("Samples: %d",d[offset + 2]);
      }
      i = offset + 5;

      /* never returns data */
      p->datalen = 0;

      p->dataPkt = FALSE;
      p->txStatusPkt = FALSE;
      p->modemStatusPkt = FALSE;
      p->remoteATPkt = FALSE;
      p->IOPkt = TRUE;

      /* save the RSSI / signal strength
         this can be used with printf as:
         printf("-%ddB\n",p->RSSI); */
      p->RSSI = d[offset];

      p->status = d[offset + 1];

      /* each sample is split into its own packet here, for simplicity */
      for (o = 0; o < p->samples; o++) {
        if (xbee.log) {
          xbee_log("--- Sample %3d -------------", o);
        }

        /* parse the io data */
        i = xbee_parse_io(p, d, offset + 3, i, o);
      }
      if (xbee.log) {
        xbee_log("----------------------------");
      }

      /* ########################################### */
      /* if: Unknown */
    } else {
      if (xbee.log) {
        xbee_log("Packet type: Unknown (0x%02X)",t);
      }
      p->type = xbee_unknown;
    }
    p->next = NULL;

    /* lock the connection mutex */
    xbee_mutex_lock(xbee.conmutex);

    con = xbee.conlist;
    hasCon = 0;
    while (con) {
      if (xbee_matchpktcon(p,con)) {
        hasCon = 1;
        break;
      }
      con = con->next;
    }

    /* unlock the connection mutex */
    xbee_mutex_unlock(xbee.conmutex);

    /* if the packet doesn’t have a connection, don’t add it! */
    if (!hasCon) {
      Xfree(p);
      if (xbee.log) {
        xbee_log("Connectionless packet... discarding!");
      }
      continue;
    }

    /* lock the packet mutex, so we can safely add the packet to the list */
    xbee_mutex_lock(xbee.pktmutex);

    /* if: the list is empty */
    if (!xbee.pktlist) {
      /* start the list! */
      xbee.pktlist = p;
    } else if (xbee.pktlast) {
      /* add the packet to the end */
      xbee.pktlast->next = p;
    } else {
      /* pktlast wasnt set... look for the end and then set it */
```

```c
      i = 0;
      q = xbee.pktlist;
      while (q->next) {
        q = q->next;
        i++;
      }
      q->next = p;
      xbee.pktcount = i;
    }
    xbee.pktlast = p;
    xbee.pktcount++;

    /* unlock the packet mutex */
    xbee_mutex_unlock(xbee.pktmutex);

    if (xbee.log) {
      xbee_log("--========================--");
      xbee_log("Packets: %d",xbee.pktcount);
    }

    p = q = NULL;
  }
  return 0;
}

/* #################################################################
   xbee_getbyte - INTERNAL
   waits for an escaped byte of data */
static unsigned char xbee_getbyte(void) {
  unsigned char c;

  ISREADY;

  /* take a byte */
  c = xbee_getrawbyte();
  /* if its escaped, take another and un-escape */
  if (c == 0x7D) c = xbee_getrawbyte() ^ 0x20;

  return (c & 0xFF);
}

/* #################################################################
   xbee_getrawbyte - INTERNAL
   waits for a raw byte of data */
static unsigned char xbee_getrawbyte(void) {
  struct timeval to;
  int ret;
  unsigned char c = 0x00;

  ISREADY;

  /* the loop is just incase there actually isnt a byte there to be read... */
  do {
    /* wait for a read to be possible */
    /* timeout every 1 second to keep alive */
    memset(&to, 0, sizeof(to));
    to.tv_usec = 1000 * 1000;
    if ((ret = xbee_select(&to)) == -1) {
      perror("libxbee:xbee_getrawbyte()");
      exit(1);
    }
    if (!xbee.listenrun) break;
    if (ret == 0) continue;

    /* read 1 character */
    xbee_read(&c,1);
#ifdef _WIN32 /* ---- */
    ret = xbee.ttyr;
    if (ret == 0) {
      usleep(10);
      continue;
    }
#endif /* ----------- */
  } while (0);

  return (c & 0xFF);
}

/* #################################################################
   xbee_send_pkt - INTERNAL
   sends a complete packet of data */
static void xbee_send_pkt(t_data *pkt) {
  ISREADY;

  /* lock the send mutex */
  xbee_mutex_lock(xbee.sendmutex);

  /* write and flush the data */
  xbee_write(pkt->data,pkt->length);

  /* unlock the mutex */
  xbee_mutex_unlock(xbee.sendmutex);

  if (xbee.log) {
    int i,x,y;
    /* prints packet in hex byte-by-byte */
    xbee_logc("TX Packet:");
    for (i=0,x=0,y=0;i<pkt->length;i++,x--) {
      if (x == 0) {
        fprintf(xbee.log,"\n  0x%04X | ",y);
        x = 0x8;
        y += x;
      }
      if (x == 4) {
        fprintf(xbee.log,"  ");
      }
      fprintf(xbee.log,"0x%02X ",pkt->data[i]);
    }
    fprintf(xbee.log,"\n");
  }

  /* free the packet */
  Xfree(pkt);
}

/* #################################################################
   xbee_make_pkt - INTERNAL
   adds delimiter field
   calculates length and checksum
   escapes bytes */
static t_data *xbee_make_pkt(unsigned char *data, int length) {
  t_data *pkt;
  unsigned int l, i, o, t, x, m;
  char d = 0;

  ISREADY;

  /* check the data given isnt too long
     100 bytes maximum payload + 12 bytes header information */
  if (length > 100 + 12) return NULL;

  /* calculate the length of the whole packet
     start, length (MSB), length (LSB), DATA, checksum */
  l = 3 + length + 1;

  /* prepare memory */
  pkt = Xcalloc(sizeof(t_data));

  /* put start byte on */
  pkt->data[0] = 0x7E;

  /* copy data into packet */
  for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
    /* if: its time for the checksum */
    if (i == length) d = M8((0xFF - M8(t)));
    /* if: its time for the high length byte */
    else if (m == 1) d = M8(length >> 8);
    /* if: its time for the low length byte */
    else if (m == 2) d = M8(length);
    /* if: its time for the normal data */
    else if (m > 2) d = data[i];

    x = 0;
    /* check for any escapes needed */
    if ((d == 0x11) || /* XON */
        (d == 0x13) || /* XOFF */
        (d == 0x7D) || /* Escape */
        (d == 0x7E)) { /* Frame Delimiter */
      l++;
      pkt->data[o++] = 0x7D;
      x = 1;
    }

    /* move data in */
    pkt->data[o] = ((!x)?d:d^0x20);
    if (m > 2) {
      i++;
      t += d;
    }
  }
```

```c

  /* remember the length */
  pkt->length = l;

  return pkt;
}
```