```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;
}
```

```
  86:
  87: /* realloc wrapper function */
  88: static void *Xrealloc(void *ptr, size_t size) {
  89:   void *t;
  90:   t = realloc(ptr,size);
  91:   if (!t) {
  92:     /* uhoh... thats pretty bad... */
  93:     perror("libxbee:realloc()");
  94:     exit(1);
  95:   }
  96:   return t;
  97: }
  98:
  99: /* free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
 100: static void Xfree2(void **ptr) {
 101:   if (!*ptr) return;
 102:   free(*ptr);
 103:   *ptr = NULL;
 104: }
 105:
 106: /* ######################################################################## */
 107: /* ### Helper Functions ################################################### */
 108: /* ######################################################################## */
 109:
 110: /* ########################################################################
 111:    returns 1 if the packet has data for the digital input else 0 */
 112: int xbee_hasdigital(xbee_pkt *pkt, int sample, int input) {
 113:   int mask = 0x0001;
 114:   if (input < 0 || input > 7) return 0;
 115:   if (sample >= pkt->samples) return 0;
 116:
 117:   mask <<= input;
 118:   return !!(pkt->IOdata[sample].IOmask & mask);
 119: }
 120:
 121: /* ########################################################################
 122:    returns 1 if the digital input is high else 0 (or 0 if no digital data present) */
 123: int xbee_getdigital(xbee_pkt *pkt, int sample, int input) {
 124:   int mask = 0x0001;
 125:   if (!xbee_hasdigital(pkt,sample,input)) return 0;
 126:
 127:   mask <<= input;
 128:   return !!(pkt->IOdata[sample].IOdigital & mask);
 129: }
 130:
 131: /* ########################################################################
 132:    returns 1 if the packet has data for the analog input else 0 */
 133: int xbee_hasanalog(xbee_pkt *pkt, int sample, int input) {
 134:   int mask = 0x0200;
 135:   if (input < 0 || input > 5) return 0;
 136:   if (sample >= pkt->samples) return 0;
 137:
 138:   mask <<= input;
 139:   return !!(pkt->IOdata[sample].IOmask & mask);
 140: }
 141:
 142: /* ########################################################################
 143:    returns analog input as a voltage if vRef is non-zero, else raw value (or 0 if no analog data present) */
 144: double xbee_getanalog(xbee_pkt *pkt, int sample, int input, double Vref) {
 145:   if (!xbee_hasanalog(pkt,sample,input)) return 0;
 146:
 147:   if (Vref) return (Vref / 1023) * pkt->IOdata[sample].IOanalog[input];
 148:   return pkt->IOdata[sample].IOanalog[input];
 149: }
 150:
 151: /* ######################################################################## */
 152: /* ### XBee Functions ##################################################### */
 153: /* ######################################################################## */
 154:
 155: static void xbee_logf(const char *logformat, const char *function, char *format, ...) {
 156:   char buf[128];
 157:   va_list ap;
 158:   FILE *log;
 159:   va_start(ap,format);
 160:   vsnprintf(buf,127,format,ap);
 161:   va_end(ap);
 162:   if (xbee.log) {
 163:     log = xbee.log;
 164:   } else {
 165:     log = stderr;
 166:   }
 167:   fprintf(log,logformat,function,buf);
 168: }
 169:
 170: /* ########################################################################
```

```
171:    xbee_sendAT - INTERNAL
172:    allows for an at command to be send, and the reply to be captured */
173: static int xbee_sendAT(char *command, char *retBuf, int retBuflen) {
174:   return xbee_sendATdelay(0,command,retBuf, retBuflen);
175: }
176: static int xbee_sendATdelay(int guartTime, char *command, char *retBuf, int retBuflen) {
177:   struct timeval to;
178:
179:   int ret;
180:   int bufi = 0;
181:
182:   /* if there is a guartTime given, then use it and a bit more */
183:   if (guartTime) usleep(guartTime * 1200);
184:
185:   /* get rid of any pre-command sludge... */
186:   memset(&to, 0, sizeof(to));
187:   ret = xbee_select(&to);
188:   if (ret > 0) {
189:     char t[128];
190:     while (xbee_read(t,127));
191:   }
192:
193:   /* send the requested command */
194:   if (xbee.log) xbee_log("sendATdelay: Sending '%s'", command);
195:   xbee_write(command, strlen(command));
196:
197:   /* if there is a guartTime, then use it */
198:   if (guartTime) {
199:     usleep(guartTime * 900);
200:
201:     /* get rid of any post-command sludge... */
202:     memset(&to, 0, sizeof(to));
203:     ret = xbee_select(&to);
204:     if (ret > 0) {
205:       char t[128];
206:       while (xbee_read(t,127));
207:     }
208:   }
209:
210:   /* retrieve the data */
211:   memset(retBuf, 0, retBuflen);
212:   memset(&to, 0, sizeof(to));
213:   if (guartTime) {
214:     /* select on the xbee fd... wait at most 0.2 the guartTime for the response */
215:     to.tv_usec = guartTime * 200;
216:   } else {
217:     /* or 250ms */
218:     to.tv_usec = 250000;
219:   }
220:   if ((ret = xbee_select(&to)) == -1) {
221:     perror("libxbee:xbee_sendATdelay()");
222:     exit(1);
223:   }
224:
225:   if (!ret) {
226:     /* timed out, and there is nothing to be read */
227:     if (xbee.log) xbee_log("sendATdelay: No Data to read - Timeout...");
228:     return 1;
229:   }
230:
231:   /* check for any dribble... */
232:   do {
233:     /* if there is actually no space in the retBuf then break out */
234:     if (bufi >= retBuflen - 1) {
235:       break;
236:     }
237:
238:     /* read as much data as is possible into retBuf */
239:     if ((ret = xbee_read(&retBuf[bufi], retBuflen - bufi - 1)) == 0) {
240:       break;
241:     }
242:
243:     /* advance the 'end of string' pointer */
244:     bufi += ret;
245:
246:     /* wait at most 150ms for any more data */
247:     memset(&to, 0, sizeof(to));
248:     to.tv_usec = 150000;
249:     if ((ret = xbee_select(&to)) == -1) {
250:       perror("libxbee:xbee_sendATdelay()");
251:       exit(1);
252:     }
253:
254:     /* loop while data was read */
255:   } while (ret);
```

```
256:
257:   if (!bufi) {
258:     if (xbee.log) xbee_log("sendATdelay: No response...");
259:     return 1;
260:   }
261:
262:   /* terminate the string */
263:   retBuf[bufi] = '\0';
264:
265:   if (xbee.log) xbee_log("sendATdelay: Recieved '%s'",retBuf);
266:   return 0;
267: }
268:
269:
270: /* ####################################################################
271:    xbee_start
272:    sets up the correct API mode for the xbee
273:    cmdSeq  = CC
274:    cmdTime = GT */
275: static int xbee_startAPI(void) {
276:   char buf[256];
277:
278:   if (xbee.cmdSeq == 0 || xbee.cmdTime == 0) return 1;
279:
280:   /* setup the command sequence string */
281:   memset(buf,xbee.cmdSeq,3);
282:   buf[3] = '\0';
283:
284:   /* try the command sequence */
285:   if (xbee_sendATdelay(xbee.cmdTime, buf, buf, sizeof(buf))) {
286:     /* if it failed... try just entering 'AT' which should return OK */
287:     if (xbee_sendAT("AT\r\n", buf, 4) || strncmp(buf,"OK\r",3)) return 1;
288:   } else if (strncmp(&buf[strlen(buf)-3],"OK\r",3)) {
289:     /* if data was returned, but it wasn't OK... then something went wrong! */
290:     return 1;
291:   }
292:
293:   /* get the current API mode */
294:   if (xbee_sendAT("ATAP\r\n", buf, 3)) return 1;
295:   buf[1] = '\0';
296:   xbee.oldAPI = atoi(buf);
297:
298:   if (xbee.oldAPI != 2) {
299:     /* if it wasnt set to mode 2 already, then set it to mode 2 */
300:     if (xbee_sendAT("ATAP2\r\n", buf, 4) || strncmp(buf,"OK\r",3)) return 1;
301:   }
302:
303:   /* quit from command mode, ready for some packets! :) */
304:   if (xbee_sendAT("ATCN\r\n", buf, 4) || strncmp(buf,"OK\r",3)) return 1;
305:
306:   return 0;
307: }
308:
309: /* ####################################################################
310:    xbee_end
311:    resets the API mode to the saved value - you must have called xbee_setup[log]API */
312: int xbee_end(void) {
313:   int ret = 1;
314:   xbee_con *con, *ncon;
315:   xbee_pkt *pkt, *npkt;
316:
317:   ISREADY;
318:   if (xbee.log) fprintf(xbee.log,"libxbee: Stopping...\n");
319:
320:   /* if the api mode was not 2 to begin with then put it back */
321:   if (xbee.oldAPI == 2) {
322:     ret = 0;
323:   } else {
324:     int to = 5;
325:
326:     con = xbee_newcon('I',xbee_localAT);
327:     xbee_senddata(con,"AP%c",xbee.oldAPI);
328:
329:     pkt = NULL;
330:
331:     while (!pkt && to--) {
332:       pkt = xbee_getpacketwait(con);
333:     }
334:     if (pkt) {
335:       ret = pkt->status;
336:       Xfree(pkt);
337:     }
338:     xbee_endcon(con);
339:   }
340:
```

```
341:   /* stop listening for data... either after timeout or next char read which ever is first */
342:   xbee.listenrun = 0;
343:   xbee_thread_kill(xbee.listent,0);
344:   /* xbee_* functions may no longer run... */
345:   xbee_ready = 0;
346:
347:   if (xbee.log) fflush(xbee.log);
348:
349:   /* nullify everything */
350:
351:   /* free all connections */
352:   con = xbee.conlist;
353:   xbee.conlist = NULL;
354:   while (con) {
355:     ncon = con->next;
356:     Xfree(con);
357:     con = ncon;
358:   }
359:
360:   /* free all packets */
361:   xbee.pktlast = NULL;
362:   pkt = xbee.pktlist;
363:   xbee.pktlist = NULL;
364:   while (pkt) {
365:     npkt = pkt->next;
366:     Xfree(pkt);
367:     pkt = npkt;
368:   }
369:
370:   /* destroy mutexes */
371:   xbee_mutex_destroy(xbee.conmutex);
372:   xbee_mutex_destroy(xbee.pktmutex);
373:   xbee_mutex_destroy(xbee.sendmutex);
374:
375:   /* close the serial port */
376:   Xfree(xbee.path);
377: #ifdef __GNUC__ /* ---- */
378:   if (xbee.tty) xbee_close(xbee.tty);
379:   if (xbee.ttyfd) close(xbee.ttyfd);
380: #else /* -------------- */
381:   if (xbee.tty) CloseHandle(xbee.tty);
382: #endif /* ------------- */
383:
384:   /* close log and tty */
385:   if (xbee.log) {
386:     fprintf(xbee.log,"libxbee: Stopped! (%s)\n",xbee_svn_version());
387:     fflush(xbee.log);
388:     xbee_close(xbee.log);
389:   }
390:
391:   /* wipe everything else... */
392:   memset(&xbee,0,sizeof(xbee));
393:
394:   return ret;
395: }
396:
397: /* ########################################################################
398:    xbee_setup
399:    opens xbee serial port & creates xbee listen thread
400:    the xbee must be configured for API mode 2
401:    THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
402: int xbee_setup(char *path, int baudrate) {
403:   return xbee_setuplogAPI(path,baudrate,0,0,0);
404: }
405: int xbee_setuplog(char *path, int baudrate, int logfd) {
406:   return xbee_setuplogAPI(path,baudrate,logfd,0,0);
407: }
408: int xbee_setupAPI(char *path, int baudrate, char cmdSeq, int cmdTime) {
409:   return xbee_setuplogAPI(path,baudrate,0,cmdSeq,cmdTime);
410: }
411: int xbee_setuplogAPI(char *path, int baudrate, int logfd, char cmdSeq, int cmdTime) {
412:   t_info info;
413:   int ret;
414:
415:   memset(&xbee,0,sizeof(xbee));
416:
417: #ifdef DEBUG
418:   /* logfd or stderr */
419:   xbee.logfd = ((logfd)?logfd:2);
420: #else
421:   xbee.logfd = logfd;
422: #endif
423:   if (xbee.logfd) {
424:     xbee.log = fdopen(xbee.logfd,"w");
425:     if (!xbee.log) {
```

```
426:       /* errno == 9 is bad file descriptor (probrably not provided) */
427:       if (errno != 9) perror("xbee_setup(): Failed opening logfile");
428:       xbee.logfd = 0;
429:     } else {
430: #ifdef __GNUC__ /* ---- */
431:       /* set to line buffer - ensure lines are written to file when complete */
432:       setvbuf(xbee.log,NULL,_IOLBF,BUFSIZ);
433: #else /* -------------- */
434:       /* Win32 is rubbish... so we have to completely disable buffering... */
435:       setvbuf(xbee.log,NULL,_IONBF,BUFSIZ);
436: #endif /* ------------- */
437:     }
438:   }
439:
440:   if (xbee.log) fprintf(xbee.log,"libxbee: Starting (%s)...\n",xbee_svn_version());
441:
442:   /* setup the connection stuff */
443:   xbee.conlist = NULL;
444:
445:   /* setup the packet stuff */
446:   xbee.pktlist = NULL;
447:   xbee.pktlast = NULL;
448:   xbee.pktcount = 0;
449:   xbee.listenrun = 1;
450:
451:   /* setup the mutexes */
452:   if (xbee_mutex_init(xbee.conmutex)) {
453:     perror("xbee_setup():xbee_mutex_init(conmutex)");
454:     return -1;
455:   }
456:   if (xbee_mutex_init(xbee.pktmutex)) {
457:     perror("xbee_setup():xbee_mutex_init(pktmutex)");
458:     xbee_mutex_destroy(xbee.conmutex);
459:     return -1;
460:   }
461:   if (xbee_mutex_init(xbee.sendmutex)) {
462:     perror("xbee_setup():xbee_mutex_init(sendmutex)");
463:     xbee_mutex_destroy(xbee.conmutex);
464:     xbee_mutex_destroy(xbee.pktmutex);
465:     return -1;
466:   }
467:
468:   /* take a copy of the XBee device path */
469:   if ((xbee.path = Xmalloc(sizeof(char) * (strlen(path) + 1))) == NULL) {
470:     perror("xbee_setup():Xmalloc(path)");
471:     xbee_mutex_destroy(xbee.conmutex);
472:     xbee_mutex_destroy(xbee.pktmutex);
473:     xbee_mutex_destroy(xbee.sendmutex);
474:     return -1;
475:   }
476:   strcpy(xbee.path,path);
477:
478:   /* call the relevant init function */
479:   if ((ret = init_serial(baudrate)) != 0) {
480:     return ret;
481:   }
482:
483:   /* when xbee_end() is called, if this is not 2 then ATAP will be set to this value */
484:   xbee.oldAPI = 2;
485:   xbee.cmdSeq = cmdSeq;
486:   xbee.cmdTime = cmdTime;
487:   if (xbee.cmdSeq && xbee.cmdTime) {
488:     if (xbee_startAPI()) {
489:       if (xbee.log) {
490:         xbee_log("Couldn't communicate with XBee...");
491:       }
492:       xbee_mutex_destroy(xbee.conmutex);
493:       xbee_mutex_destroy(xbee.pktmutex);
494:       xbee_mutex_destroy(xbee.sendmutex);
495:       Xfree(xbee.path);
496: #ifdef __GNUC__ /* ---- */
497:       close(xbee.ttyfd);
498: #endif /* ------------- */
499:       xbee_close(xbee.tty);
500:       return -1;
501:     }
502:   }
503:
504:   /* allow the listen thread to start */
505:   xbee_ready = -1;
506:
507:   /* can start xbee_listen thread now */
508:   if (xbee_thread_create(xbee.listent,xbee_listen_wrapper,info)) {
509:     perror("xbee_setup():xbee_thread_create()");
510:     xbee_mutex_destroy(xbee.conmutex);
```

```
511:     xbee_mutex_destroy(xbee.pktmutex);
512:     xbee_mutex_destroy(xbee.sendmutex);
513:     Xfree(xbee.path);
514: #ifdef __GNUC__ /* ---- */
515:     close(xbee.ttyfd);
516: #endif /* ------------- */
517:     xbee_close(xbee.tty);
518:     return -1;
519:   }
520:
521:   usleep(100);
522:   while (xbee_ready != -2) {
523:     usleep(100);
524:     if (xbee.log) {
525:       xbee_log("Waiting for xbee_listen() to be ready...");
526:     }
527:   }
528:
529:   /* allow other functions to be used! */
530:   xbee_ready = 1;
531:
532:   if (xbee.log) fprintf(xbee.log,"libxbee: Started!\n");
533:
534:   return 0;
535: }
536:
537: /* ################################################################
538:    xbee_con
539:    produces a connection to the specified device and frameID
540:    if a connection had already been made, then this connection will be returned */
541: xbee_con *xbee_newcon(unsigned char frameID, xbee_types type, ...) {
542:   xbee_con *con, *ocon;
543:   unsigned char tAddr[8];
544:   va_list ap;
545:   int t;
546:   int i;
547:
548:   ISREADY;
549:
550:   if (!type || type == xbee_unknown) type = xbee_localAT; /* default to local AT */
551:   else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */
552:
553:   va_start(ap,type);
554:   /* if: 64 bit address expected (2 ints) */
555:   if ((type == xbee_64bitRemoteAT) ||
556:       (type == xbee_64bitData) ||
557:       (type == xbee_64bitIO)) {
558:     t = va_arg(ap, int);
559:     tAddr[0] = (t >> 24) & 0xFF;
560:     tAddr[1] = (t >> 16) & 0xFF;
561:     tAddr[2] = (t >>  8) & 0xFF;
562:     tAddr[3] = (t      ) & 0xFF;
563:     t = va_arg(ap, int);
564:     tAddr[4] = (t >> 24) & 0xFF;
565:     tAddr[5] = (t >> 16) & 0xFF;
566:     tAddr[6] = (t >>  8) & 0xFF;
567:     tAddr[7] = (t      ) & 0xFF;
568:
569:     /* if: 16 bit address expected (1 int) */
570:   } else if ((type == xbee_16bitRemoteAT) ||
571:              (type == xbee_16bitData) ||
572:              (type == xbee_16bitIO)) {
573:     t = va_arg(ap, int);
574:     tAddr[0] = (t >>  8) & 0xFF;
575:     tAddr[1] = (t      ) & 0xFF;
576:     tAddr[2] = 0;
577:     tAddr[3] = 0;
578:     tAddr[4] = 0;
579:     tAddr[5] = 0;
580:     tAddr[6] = 0;
581:     tAddr[7] = 0;
582:
583:     /* otherwise clear the address */
584:   } else {
585:     memset(tAddr,0,8);
586:   }
587:   va_end(ap);
588:
589:   /* lock the connection mutex */
590:   xbee_mutex_lock(xbee.conmutex);
591:
592:   /* are there any connections? */
593:   if (xbee.conlist) {
594:     con = xbee.conlist;
595:     while (con) {
```

```
596:       /* if: after a modemStatus, and the types match! */
597:       if ((type == xbee_modemStatus) &&
598:           (con->type == type)) {
599:         xbee_mutex_unlock(xbee.conmutex);
600:         return con;
601:
602:         /* if: after a txStatus and frameIDs match! */
603:       } else if ((type == xbee_txStatus) &&
604:                  (con->type == type) &&
605:                  (frameID == con->frameID)) {
606:         xbee_mutex_unlock(xbee.conmutex);
607:         return con;
608:
609:         /* if: after a localAT, and the frameIDs match! */
610:       } else if ((type == xbee_localAT) &&
611:                  (con->type == type) &&
612:                  (frameID == con->frameID)) {
613:         xbee_mutex_unlock(xbee.conmutex);
614:         return con;
615:
616:         /* if: connection types match, the frameIDs match, and the addresses match! */
617:       } else if ((type == con->type) &&
618:                  (frameID == con->frameID) &&
619:                  (!memcmp(tAddr,con->tAddr,8))) {
620:         xbee_mutex_unlock(xbee.conmutex);
621:         return con;
622:       }
623:
624:       /* if there are more, move along, dont want to loose that last item! */
625:       if (con->next == NULL) break;
626:       con = con->next;
627:     }
628:
629:     /* keep hold of the last connection... we will need to link it up later */
630:     ocon = con;
631:   }
632:
633:   /* create a new connection and set its attributes */
634:   con = Xcalloc(sizeof(xbee_con));
635:   con->type = type;
636:   /* is it a 64bit connection? */
637:   if ((type == xbee_64bitRemoteAT) ||
638:       (type == xbee_64bitData) ||
639:       (type == xbee_64bitIO)) {
640:     con->tAddr64 = TRUE;
641:   }
642:   con->atQueue = 0; /* queue AT commands? */
643:   con->txDisableACK = 0; /* disable ACKs? */
644:   con->txBroadcast = 0; /* broadcast? */
645:   con->frameID = frameID;
646:   memcpy(con->tAddr,tAddr,8); /* copy in the remote address */
647:
648:   if (xbee.log) {
649:     switch(type) {
650:     case xbee_localAT:
651:       xbee_log("New local AT connection!");
652:       break;
653:     case xbee_16bitRemoteAT:
654:     case xbee_64bitRemoteAT:
655:       xbee_logc("New %d-bit remote AT connection! (to: ",(con->tAddr64?64:16));
656:       for (i=0;i<(con->tAddr64?8:2);i++) {
657:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
658:       }
659:       fprintf(xbee.log,")\n");
660:       break;
661:     case xbee_16bitData:
662:     case xbee_64bitData:
663:       xbee_logc("New %d-bit data connection! (to: ",(con->tAddr64?64:16));
664:       for (i=0;i<(con->tAddr64?8:2);i++) {
665:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
666:       }
667:       fprintf(xbee.log,")\n");
668:       break;
669:     case xbee_16bitIO:
670:     case xbee_64bitIO:
671:       xbee_logc("New %d-bit IO connection! (to: ",(con->tAddr64?64:16));
672:       for (i=0;i<(con->tAddr64?8:2);i++) {
673:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
674:       }
675:       fprintf(xbee.log,")\n");
676:       break;
677:     case xbee_txStatus:
678:       xbee_log("New Tx status connection!");
679:       break;
680:     case xbee_modemStatus:
```

```
681:       xbee_log("New modem status connection!");
682:       break;
683:     case xbee_unknown:
684:     default:
685:       xbee_log("New unknown connection!");
686:     }
687:   }
688:
689:   /* make it the last in the list */
690:   con->next = NULL;
691:   /* add it to the list */
692:   if (xbee.conlist) {
693:     ocon->next = con;
694:   } else {
695:     xbee.conlist = con;
696:   }
697:
698:   /* unlock the mutex */
699:   xbee_mutex_unlock(xbee.conmutex);
700:   return con;
701: }
702:
703: /* ###################################################################
704:    xbee_conflush
705:    removes any packets that have been collected for the specified
706:    connection */
707: void xbee_flushcon(xbee_con *con) {
708:   xbee_pkt *r, *p, *n;
709:
710:   /* lock the packet mutex */
711:   xbee_mutex_lock(xbee.pktmutex);
712:
713:   /* if: there are packets */
714:   if ((p = xbee.pktlist) != NULL) {
715:     r = NULL;
716:     /* get all packets for this connection */
717:     do {
718:       /* does the packet match the connection? */
719:       if (xbee_matchpktcon(p,con)) {
720:         /* if it was the first packet */
721:         if (!r) {
722:           /* move the chain along */
723:           xbee.pktlist = p->next;
724:         } else {
725:           /* otherwise relink the list */
726:           r->next = p->next;
727:         }
728:         xbee.pktcount--;
729:
730:         /* free this packet! */
731:         n = p->next;
732:         Xfree(p);
733:         /* move on */
734:         p = n;
735:       } else {
736:         /* move on */
737:         r = p;
738:         p = p->next;
739:       }
740:     } while (p);
741:     xbee.pktlast = r;
742:   }
743:
744:   /* unlock the packet mutex */
745:   xbee_mutex_unlock(xbee.pktmutex);
746: }
747:
748: /* ###################################################################
749:    xbee_endcon
750:    close the unwanted connection
751:    free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
752: void xbee_endcon2(xbee_con **con) {
753:   xbee_con *t, *u;
754:
755:   /* lock the connection mutex */
756:   xbee_mutex_lock(xbee.conmutex);
757:
758:   u = t = xbee.conlist;
759:   while (t && t != *con) {
760:     u = t;
761:     t = t->next;
762:   }
763:   if (!t) {
764:     /* invalid connection given... */
765:     if (xbee.log) {
```

```
766:       xbee_log("Attempted to close invalid connection...");
767:     }
768:     /* unlock the connection mutex */
769:     xbee_mutex_unlock(xbee.conmutex);
770:     return;
771:   }
772:   /* extract this connection from the list */
773:   u->next = (*con)->next;
774:   if (*con == xbee.conlist) xbee.conlist = NULL;
775:
776:   /* unlock the connection mutex */
777:   xbee_mutex_unlock(xbee.conmutex);
778:
779:   /* remove all packets for this connection */
780:   xbee_flushcon(*con);
781:
782:   /* free the connection! */
783:   Xfree(*con);
784: }
785:
786: /* ####################################################################
787:    xbee_senddata
788:    send the specified data to the provided connection */
789: int xbee_senddata(xbee_con *con, char *format, ...) {
790:   int ret;
791:   va_list ap;
792:
793:   ISREADY;
794:
795:   /* xbee_vsenddata() wants a va_list... */
796:   va_start(ap, format);
797:   /* hand it over :) */
798:   ret = xbee_vsenddata(con,format,ap);
799:   va_end(ap);
800:   return ret;
801: }
802:
803: int xbee_vsenddata(xbee_con *con, char *format, va_list ap) {
804:   unsigned char data[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
805:   int length;
806:
807:   ISREADY;
808:
809:   /* make up the data and keep the length, its possible there are nulls in there */
810:   length = vsnprintf((char *)data,128,format,ap);
811:
812:   /* hand it over :) */
813:   return xbee_nsenddata(con,(char *)data,length);
814: }
815:
816: int xbee_nsenddata(xbee_con *con, char *data, int length) {
817:   t_data *pkt;
818:   int i;
819:   unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
820:
821:   ISREADY;
822:
823:   if (!con) return -1;
824:   if (con->type == xbee_unknown) return -1;
825:   if (length > 127) return -1;
826:
827:
828:   if (xbee.log) {
829:     xbee_log("--== TX Packet ============--");
830:     xbee_logc("Connection Type: ");
831:     switch (con->type) {
832:     case xbee_unknown:       fprintf(xbee.log,"Unknown\n"); break;
833:     case xbee_localAT:       fprintf(xbee.log,"Local AT\n"); break;
834:     case xbee_remoteAT:      fprintf(xbee.log,"Remote AT\n"); break;
835:     case xbee_16bitRemoteAT: fprintf(xbee.log,"Remote AT (16-bit)\n"); break;
836:     case xbee_64bitRemoteAT: fprintf(xbee.log,"Remote AT (64-bit)\n"); break;
837:     case xbee_16bitData:     fprintf(xbee.log,"Data (16-bit)\n"); break;
838:     case xbee_64bitData:     fprintf(xbee.log,"Data (64-bit)\n"); break;
839:     case xbee_16bitIO:       fprintf(xbee.log,"IO (16-bit)\n"); break;
840:     case xbee_64bitIO:       fprintf(xbee.log,"IO (64-bit)\n"); break;
841:     case xbee_txStatus:      fprintf(xbee.log,"Tx Status\n"); break;
842:     case xbee_modemStatus:   fprintf(xbee.log,"Modem Status\n"); break;
843:     }
844:     xbee_logc("Destination: ");
845:     for (i=0;i<(con->tAddr64?8:2);i++) {
846:       fprintf(xbee.log,(i?":%02X":"%02X"),con->tAddr[i]);
847:     }
848:     fprintf(xbee.log,"\n");
849:     xbee_log("Length: %d",length);
850:     for (i=0;i<length;i++) {
```

```
851:       xbee_logc("%3d | 0x%02X ",i,(unsigned char)data[i]);
852:       if ((data[i] > 32) && (data[i] < 127)) {
853:         fprintf(xbee.log,"'%c'\n",data[i]);
854:       } else{
855:         fprintf(xbee.log," _\n");
856:       }
857:     }
858:   }
859:
860:   /* ######################################### */
861:   /* if: local AT */
862:   if (con->type == xbee_localAT) {
863:     /* AT commands are 2 chars long (plus optional parameter) */
864:     if (length < 2) return -1;
865:
866:     /* use the command? */
867:     buf[0] = ((!con->atQueue)?0x08:0x09);
868:     buf[1] = con->frameID;
869:
870:     /* copy in the data */
871:     for (i=0;i<length;i++) {
872:       buf[i+2] = data[i];
873:     }
874:
875:     /* setup the packet */
876:     pkt = xbee_make_pkt(buf,i+2);
877:     /* send it on */
878:     xbee_send_pkt(pkt);
879:
880:     return 0;
881:
882:     /* ######################################### */
883:     /* if: remote AT */
884:   } else if ((con->type == xbee_16bitRemoteAT) ||
885:              (con->type == xbee_64bitRemoteAT)) {
886:     if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter) */
887:     buf[0] = 0x17;
888:     buf[1] = con->frameID;
889:
890:     /* copy in the relevant address */
891:     if (con->tAddr64) {
892:       memcpy(&buf[2],con->tAddr,8);
893:       buf[10] = 0xFF;
894:       buf[11] = 0xFE;
895:     } else {
896:       memset(&buf[2],0,8);
897:       memcpy(&buf[10],con->tAddr,2);
898:     }
899:     /* queue the command? */
900:     buf[12] = ((!con->atQueue)?0x02:0x00);
901:
902:     /* copy in the data */
903:     for (i=0;i<length;i++) {
904:       buf[i+13] = data[i];
905:     }
906:
907:     /* setup the packet */
908:     pkt = xbee_make_pkt(buf,i+13);
909:     /* send it on */
910:     xbee_send_pkt(pkt);
911:
912:     return 0;
913:
914:     /* ######################################### */
915:     /* if: 16 or 64bit Data */
916:   } else if ((con->type == xbee_16bitData) ||
917:              (con->type == xbee_64bitData)) {
918:     int offset;
919:
920:     /* if: 16bit Data */
921:     if (con->type == xbee_16bitData) {
922:       buf[0] = 0x01;
923:       offset = 5;
924:       /* copy in the address */
925:       memcpy(&buf[2],con->tAddr,2);
926:
927:       /* if: 64bit Data */
928:     } else { /* 64bit Data */
929:       buf[0] = 0x00;
930:       offset = 11;
931:       /* copy in the address */
932:       memcpy(&buf[2],con->tAddr,8);
933:     }
934:
935:     /* copy frameID */
```

```
936:     buf[1] = con->frameID;
937:
938:     /* disable ack? broadcast? */
939:     buf[offset-1] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
940:
941:     /* copy in the data */
942:     for (i=0;i<length;i++) {
943:       buf[i+offset] = data[i];
944:     }
945:
946:     /* setup the packet */
947:     pkt = xbee_make_pkt(buf,i+offset);
948:     /* send it on */
949:     xbee_send_pkt(pkt);
950:
951:     return 0;
952:
953:     /* ######################################### */
954:     /* if: I/O */
955:   } else if ((con->type == xbee_64bitIO) ||
956:              (con->type == xbee_16bitIO)) {
957:     /* not currently implemented... is it even allowed? */
958:     if (xbee.log) {
959:       fprintf(xbee.log,"******* TODO ********\n");
960:     }
961:   }
962:
963:   return -2;
964: }
965:
966: /* ###################################################################
967:    xbee_getpacket
968:    retrieves the next packet destined for the given connection
969:    once the packet has been retrieved, it is removed for the list! */
970: xbee_pkt *xbee_getpacketwait(xbee_con *con) {
971:   xbee_pkt *p;
972:   int i;
973:
974:   /* 50ms * 20 = 1 second */
975:   for (i = 0; i < 20; i++) {
976:     p = xbee_getpacket(con);
977:     if (p) break;
978:     usleep(50000); /* 50ms */
979:   }
980:
981:   return p;
982: }
983: xbee_pkt *xbee_getpacket(xbee_con *con) {
984:   xbee_pkt *l, *p, *q;
985:   /*if (xbee.log) {
986:     xbee_log("-:== Get Packet ==========--");
987:     }*/
988:
989:   /* lock the packet mutex */
990:   xbee_mutex_lock(xbee.pktmutex);
991:
992:   /* if: there are no packets */
993:   if ((p = xbee.pktlist) == NULL) {
994:     xbee_mutex_unlock(xbee.pktmutex);
995:     /*if (xbee.log) {
996:       xbee_log("No packets avaliable...");
997:       }*/
998:     return NULL;
999:   }
1000:
1001:   l = NULL;
1002:   q = NULL;
1003:   /* get the first avaliable packet for this connection */
1004:   do {
1005:     /* does the packet match the connection? */
1006:     if (xbee_matchpktcon(p,con)) {
1007:       q = p;
1008:       break;
1009:     }
1010:     /* move on */
1011:     l = p;
1012:     p = p->next;
1013:   } while (p);
1014:
1015:   /* if: no packet was found */
1016:   if (!q) {
1017:     xbee_mutex_unlock(xbee.pktmutex);
1018:     /*if (xbee.log) {
1019:       xbee_log("No packets avaliable (for connection)...");
1020:       }*/
```

```
1021:     return NULL;
1022:   }
1023:
1024:   /* if it was the first packet */
1025:   if (l) {
1026:     /* relink the list */
1027:     l->next = p->next;
1028:     if (!l->next) xbee.pktlast = l;
1029:   } else {
1030:     /* move the chain along */
1031:     xbee.pktlist = p->next;
1032:     if (!xbee.pktlist) {
1033:       xbee.pktlast = NULL;
1034:     } else if (!xbee.pktlist->next) {
1035:       xbee.pktlast = xbee.pktlist;
1036:     }
1037:   }
1038:   xbee.pktcount--;
1039:
1040:   /* unlink this packet from the chain! */
1041:   q->next = NULL;
1042:
1043:   if (xbee.log) {
1044:     xbee_log("--== Get Packet ==========--");
1045:     xbee_log("Got a packet");
1046:     xbee_log("Packets left: %d",xbee.pktcount);
1047:   }
1048:
1049:   /* unlock the packet mutex */
1050:   xbee_mutex_unlock(xbee.pktmutex);
1051:
1052:   /* and return the packet (must be free'd by caller!) */
1053:   return q;
1054: }
1055:
1056: /* ########################################################################
1057:    xbee_matchpktcon - INTERNAL
1058:    checks if the packet matches the connection */
1059: static int xbee_matchpktcon(xbee_pkt *pkt, xbee_con *con) {
1060:   /* if: the connection type matches the packet type OR
1061:      the connection is 16/64bit remote AT, and the packet is a remote AT response */
1062:   if ((pkt->type == con->type) || /* -- */
1063:       ((pkt->type == xbee_remoteAT) && /* -- */
1064:        ((con->type == xbee_16bitRemoteAT) ||
1065:         (con->type == xbee_64bitRemoteAT)))) {
1066:
1067:     /* if: the packet is modem status OR
1068:        the packet is tx status or AT data and the frame IDs match OR
1069:        the addresses match */
1070:     if (pkt->type == xbee_modemStatus) return 1;
1071:
1072:     if ((pkt->type == xbee_txStatus) ||
1073:         (pkt->type == xbee_localAT) ||
1074:         (pkt->type == xbee_remoteAT)) {
1075:       if (pkt->frameID == con->frameID) {
1076:         return 1;
1077:       }
1078:     } else if (pkt->sAddr64 && !memcmp(pkt->Addr64,con->tAddr,8)) {
1079:       return 1;
1080:     } else if (!pkt->sAddr64 && !memcmp(pkt->Addr16,con->tAddr,2)) {
1081:       return 1;
1082:     }
1083:   }
1084:   return 0;
1085: }
1086:
1087: /* ########################################################################
1088:    xbee_parse_io - INTERNAL
1089:    parses the data given into the packet io information */
1090: static int xbee_parse_io(xbee_pkt *p, unsigned char *d, int maskOffset, int sampleOffset, int sample) {
1091:   xbee_sample *s = &(p->IOdata[sample]);
1092:
1093:   /* copy in the I/O data mask */
1094:   s->IOmask = (((d[maskOffset]<<8) | d[maskOffset + 1]) & 0x7FFF);
1095:
1096:   /* copy in the digital I/O data */
1097:   s->IOdigital = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x01FF);
1098:
1099:   /* advance over the digital data, if its there */
1100:   sampleOffset += ((s->IOmask & 0x01FF)?2:0);
1101:
1102:   /* copy in the analog I/O data */
1103:   if (s->IOmask & 0x0200) {
1104:     s->IOanalog[0] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1105:     sampleOffset+=2;
```

```
1106:   }
1107:   if (s->IOmask & 0x0400) {
1108:     s->IOanalog[1] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1109:     sampleOffset+=2;
1110:   }
1111:   if (s->IOmask & 0x0800) {
1112:     s->IOanalog[2] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1113:     sampleOffset+=2;
1114:   }
1115:   if (s->IOmask & 0x1000) {
1116:     s->IOanalog[3] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1117:     sampleOffset+=2;
1118:   }
1119:   if (s->IOmask & 0x2000) {
1120:     s->IOanalog[4] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1121:     sampleOffset+=2;
1122:   }
1123:   if (s->IOmask & 0x4000) {
1124:     s->IOanalog[5] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1125:     sampleOffset+=2;
1126:   }
1127:
1128:   if (xbee.log) {
1129:     if (s->IOmask & 0x0001)
1130:       xbee_log("Digital 0: %c",((s->IOdigital & 0x0001)?'1':'0'));
1131:     if (s->IOmask & 0x0002)
1132:       xbee_log("Digital 1: %c",((s->IOdigital & 0x0002)?'1':'0'));
1133:     if (s->IOmask & 0x0004)
1134:       xbee_log("Digital 2: %c",((s->IOdigital & 0x0004)?'1':'0'));
1135:     if (s->IOmask & 0x0008)
1136:       xbee_log("Digital 3: %c",((s->IOdigital & 0x0008)?'1':'0'));
1137:     if (s->IOmask & 0x0010)
1138:       xbee_log("Digital 4: %c",((s->IOdigital & 0x0010)?'1':'0'));
1139:     if (s->IOmask & 0x0020)
1140:       xbee_log("Digital 5: %c",((s->IOdigital & 0x0020)?'1':'0'));
1141:     if (s->IOmask & 0x0040)
1142:       xbee_log("Digital 6: %c",((s->IOdigital & 0x0040)?'1':'0'));
1143:     if (s->IOmask & 0x0080)
1144:       xbee_log("Digital 7: %c",((s->IOdigital & 0x0080)?'1':'0'));
1145:     if (s->IOmask & 0x0100)
1146:       xbee_log("Digital 8: %c",((s->IOdigital & 0x0100)?'1':'0'));
1147:     if (s->IOmask & 0x0200)
1148:       xbee_log("Analog  0: %d (~%.2fv)\n",s->IOanalog[0],(3.3/1023)*s->IOanalog[0]);
1149:     if (s->IOmask & 0x0400)
1150:       xbee_log("Analog  1: %d (~%.2fv)\n",s->IOanalog[1],(3.3/1023)*s->IOanalog[1]);
1151:     if (s->IOmask & 0x0800)
1152:       xbee_log("Analog  2: %d (~%.2fv)\n",s->IOanalog[2],(3.3/1023)*s->IOanalog[2]);
1153:     if (s->IOmask & 0x1000)
1154:       xbee_log("Analog  3: %d (~%.2fv)\n",s->IOanalog[3],(3.3/1023)*s->IOanalog[3]);
1155:     if (s->IOmask & 0x2000)
1156:       xbee_log("Analog  4: %d (~%.2fv)\n",s->IOanalog[4],(3.3/1023)*s->IOanalog[4]);
1157:     if (s->IOmask & 0x4000)
1158:       xbee_log("Analog  5: %d (~%.2fv)\n",s->IOanalog[5],(3.3/1023)*s->IOanalog[5]);
1159:   }
1160:
1161:   return sampleOffset;
1162: }
1163:
1164: /* ########################################################################
1165:    xbee_listen_stop
1166:    stops the listen thread after the current packet has been processed */
1167: void xbee_listen_stop(void) {
1168:   xbee.listenrun = 0;
1169: }
1170:
1171: /* ########################################################################
1172:    xbee_listen_wrapper - INTERNAL
1173:    the xbee_listen wrapper. Prints an error when xbee_listen ends */
1174: static void xbee_listen_wrapper(t_info *info) {
1175:   int ret;
1176:   /* just falls out if the proper 'go-ahead' isn't given */
1177:   if (xbee_ready != -1) return;
1178:   /* now allow the parent to continue */
1179:   xbee_ready = -2;
1180:
1181: #ifdef _WIN32 /* ---- */
1182:   /* win32 requires this delay... no idea why */
1183:   usleep(1000000);
1184: #endif /* ----------- */
1185:
1186:   while (xbee.listenrun) {
1187:     info->i = -1;
1188:     ret = xbee_listen(info);
1189:     if (!xbee.listenrun) break;
1190:     if (xbee.log) {
```

```
1191:       xbee_log("xbee_listen() returned [%d]... Restarting in 250ms!",ret);
1192:     }
1193:     usleep(25000);
1194:   }
1195: }
1196:
1197: /* xbee_listen - INTERNAL
1198:    the xbee xbee_listen thread
1199:    reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
1200: static int xbee_listen(t_info *info) {
1201:   unsigned char c, t, d[1024];
1202:   unsigned int l, i, chksum, o;
1203:   int j;
1204:   xbee_pkt *p, *q;
1205:   xbee_con *con;
1206:   int hasCon;
1207:
1208:   /* just falls out if the proper 'go-ahead' isn't given */
1209:   if (info->i != -1) return -1;
1210:   /* do this forever :) */
1211:   while (xbee.listenrun) {
1212:     /* wait for a valid start byte */
1213:     if (xbee_getrawbyte() != 0x7E) continue;
1214:     if (!xbee.listenrun) return 0;
1215:
1216:     if (xbee.log) {
1217:       xbee_log("--== RX Packet ==========--");
1218:       xbee_log("Got a packet!...");
1219:     }
1220:
1221:     /* get the length */
1222:     l = xbee_getbyte() << 8;
1223:     l += xbee_getbyte();
1224:
1225:     /* check it is a valid length... */
1226:     if (!l) {
1227:       if (xbee.log) {
1228:         xbee_log("Recived zero length packet!");
1229:       }
1230:       continue;
1231:     }
1232:     if (l > 100) {
1233:       if (xbee.log) {
1234:         xbee_log("Recived oversized packet! Length: %d",l - 1);
1235:       }
1236:     }
1237:     if (l > sizeof(d) - 1) {
1238:       if (xbee.log) {
1239:         xbee_log("Recived packet larger than buffer! Discarding...");
1240:       }
1241:       continue;
1242:     }
1243:
1244:     if (xbee.log) {
1245:       xbee_log("Length: %d",l - 1);
1246:     }
1247:
1248:     /* get the packet type */
1249:     t = xbee_getbyte();
1250:
1251:     /* start the checksum */
1252:     chksum = t;
1253:
1254:     /* suck in all the data */
1255:     for (i = 0; l > 1 && i < 128; l--, i++) {
1256:       /* get an unescaped byte */
1257:       c = xbee_getbyte();
1258:       d[i] = c;
1259:       chksum += c;
1260:       if (xbee.log) {
1261:         xbee_logc("%3d | 0x%02X | ",i,c);
1262:         if ((c > 32) && (c < 127)) fprintf(xbee.log,"'%c'",c); else fprintf(xbee.log," _ ");
1263:
1264:         if ((t == 0x80 && i == (8 + 2)) || /* 64-bit Data packet */
1265:             (t == 0x81 && i == (2 + 2))) { /* 16-bit Data packet */
1266:           /* mark the beginning of the 'data' bytes */
1267:           fprintf(xbee.log,"   <-- data starts");
1268:         }
1269:
1270:         fprintf(xbee.log,"\n");
1271:       }
1272:     }
1273:     i--; /* it went up too many times!... */
1274:
1275:     /* add the checksum */
```

```
1276:     chksum += xbee_getbyte();
1277:
1278:     /* check if the whole packet was recieved, or something else occured... unlikely... */
1279:     if (l>1) {
1280:       if (xbee.log) {
1281:         xbee_log("Didn't get whole packet... :(");
1282:       }
1283:       continue;
1284:     }
1285:
1286:     /* check the checksum */
1287:     if ((chksum & 0xFF) != 0xFF) {
1288:       if (xbee.log) {
1289:         xbee_log("Invalid Checksum: 0x%02X",chksum);
1290:       }
1291:       continue;
1292:     }
1293:
1294:     /* make a new packet */
1295:     p = Xcalloc(sizeof(xbee_pkt));
1296:     q = NULL;
1297:     p->datalen = 0;
1298:
1299:     /* ######################################### */
1300:     /* if: modem status */
1301:     if (t == 0x8A) {
1302:       if (xbee.log) {
1303:         xbee_log("Packet type: Modem Status (0x8A)");
1304:         xbee_logc("Event: ");
1305:         switch (d[0]) {
1306:         case 0x00: fprintf(xbee.log,"Hardware reset"); break;
1307:         case 0x01: fprintf(xbee.log,"Watchdog timer reset"); break;
1308:         case 0x02: fprintf(xbee.log,"Associated"); break;
1309:         case 0x03: fprintf(xbee.log,"Disassociated"); break;
1310:         case 0x04: fprintf(xbee.log,"Synchronization lost"); break;
1311:         case 0x05: fprintf(xbee.log,"Coordinator realignment"); break;
1312:         case 0x06: fprintf(xbee.log,"Coordinator started"); break;
1313:         }
1314:         fprintf(xbee.log,"... (0x%02X)\n",d[0]);
1315:       }
1316:       p->type = xbee_modemStatus;
1317:
1318:       p->sAddr64 = FALSE;
1319:       p->dataPkt = FALSE;
1320:       p->txStatusPkt = FALSE;
1321:       p->modemStatusPkt = TRUE;
1322:       p->remoteATPkt = FALSE;
1323:       p->IOPkt = FALSE;
1324:
1325:       /* modem status can only ever give 1 'data' byte */
1326:       p->datalen = 1;
1327:       p->data[0] = d[0];
1328:
1329:       /* ######################################### */
1330:       /* if: local AT response */
1331:     } else if (t == 0x88) {
1332:       if (xbee.log) {
1333:         xbee_log("Packet type: Local AT Response (0x88)");
1334:         xbee_log("FrameID: 0x%02X",d[0]);
1335:         xbee_log("AT Command: %c%c",d[1],d[2]);
1336:         xbee_logc("Status: ");
1337:         if (d[3] == 0) fprintf(xbee.log,"OK");
1338:         else if (d[3] == 1) fprintf(xbee.log,"Error");
1339:         else if (d[3] == 2) fprintf(xbee.log,"Invalid Command");
1340:         else if (d[3] == 3) fprintf(xbee.log,"Invalid Parameter");
1341:         fprintf(xbee.log," (0x%02X)\n",d[3]);
1342:       }
1343:       p->type = xbee_localAT;
1344:
1345:       p->sAddr64 = FALSE;
1346:       p->dataPkt = FALSE;
1347:       p->txStatusPkt = FALSE;
1348:       p->modemStatusPkt = FALSE;
1349:       p->remoteATPkt = FALSE;
1350:       p->IOPkt = FALSE;
1351:
1352:       p->frameID = d[0];
1353:       p->atCmd[0] = d[1];
1354:       p->atCmd[1] = d[2];
1355:
1356:       p->status = d[3];
1357:
1358:       /* copy in the data */
1359:       p->datalen = i-3;
1360:       for (;i>3;i--) p->data[i-4] = d[i];
```

```
1361:
1362:       /* ######################################### */
1363:       /* if: remote AT response */
1364:     } else if (t == 0x97) {
1365:       if (xbee.log) {
1366:         xbee_log("Packet type: Remote AT Response (0x97)");
1367:         xbee_log("FrameID: 0x%02X",d[0]);
1368:         xbee_logc("64-bit Address: ");
1369:         for (j=0;j<8;j++) {
1370:           fprintf(xbee.log,(j?":%02X":"%02X"),d[1+j]);
1371:         }
1372:         fprintf(xbee.log,"\n");
1373:         xbee_logc("16-bit Address: ");
1374:         for (j=0;j<2;j++) {
1375:           fprintf(xbee.log,(j?":%02X":"%02X"),d[9+j]);
1376:         }
1377:         fprintf(xbee.log,"\n");
1378:         xbee_log("AT Command: %c%c",d[11],d[12]);
1379:         xbee_logc("Status: ");
1380:         if (d[13] == 0) fprintf(xbee.log,"OK");
1381:         else if (d[13] == 1) fprintf(xbee.log,"Error");
1382:         else if (d[13] == 2) fprintf(xbee.log,"Invalid Command");
1383:         else if (d[13] == 3) fprintf(xbee.log,"Invalid Parameter");
1384:         else if (d[13] == 4) fprintf(xbee.log,"No Response");
1385:         fprintf(xbee.log," (0x%02X)\n",d[13]);
1386:       }
1387:       p->type = xbee_remoteAT;
1388:
1389:       p->sAddr64 = FALSE;
1390:       p->dataPkt = FALSE;
1391:       p->txStatusPkt = FALSE;
1392:       p->modemStatusPkt = FALSE;
1393:       p->remoteATPkt = TRUE;
1394:       p->IOPkt = FALSE;
1395:
1396:       p->frameID = d[0];
1397:
1398:       p->Addr64[0] = d[1];
1399:       p->Addr64[1] = d[2];
1400:       p->Addr64[2] = d[3];
1401:       p->Addr64[3] = d[4];
1402:       p->Addr64[4] = d[5];
1403:       p->Addr64[5] = d[6];
1404:       p->Addr64[6] = d[7];
1405:       p->Addr64[7] = d[8];
1406:
1407:       p->Addr16[0] = d[9];
1408:       p->Addr16[1] = d[10];
1409:
1410:       p->atCmd[0] = d[11];
1411:       p->atCmd[1] = d[12];
1412:
1413:       p->status = d[13];
1414:
1415:       p->samples = 1;
1416:
1417:       if (p->status == 0x00 && p->atCmd[0] == 'I' && p->atCmd[1] == 'S') {
1418:         /* parse the io data */
1419:         if (xbee.log) xbee_log("--- Sample -----------------");
1420:         xbee_parse_io(p, d, 15, 17, 0);
1421:         if (xbee.log) xbee_log("----------------------------");
1422:       } else {
1423:         /* copy in the data */
1424:         p->datalen = i-13;
1425:         for (;i>13;i--) p->data[i-14] = d[i];
1426:       }
1427:
1428:       /* ######################################### */
1429:       /* if: TX status */
1430:     } else if (t == 0x89) {
1431:       if (xbee.log) {
1432:         xbee_log("Packet type: TX Status Report (0x89)");
1433:         xbee_log("FrameID: 0x%02X",d[0]);
1434:         xbee_logc("Status: ");
1435:         if (d[1] == 0) fprintf(xbee.log,"Success");
1436:         else if (d[1] == 1) fprintf(xbee.log,"No ACK");
1437:         else if (d[1] == 2) fprintf(xbee.log,"CCA Failure");
1438:         else if (d[1] == 3) fprintf(xbee.log,"Purged");
1439:         fprintf(xbee.log," (0x%02X)\n",d[1]);
1440:       }
1441:       p->type = xbee_txStatus;
1442:
1443:       p->sAddr64 = FALSE;
1444:       p->dataPkt = FALSE;
1445:       p->txStatusPkt = TRUE;
```

```
1446:       p->modemStatusPkt = FALSE;
1447:       p->remoteATPkt = FALSE;
1448:       p->IOPkt = FALSE;
1449:
1450:       p->frameID = d[0];
1451:
1452:       p->status = d[1];
1453:
1454:       /* never returns data */
1455:       p->datalen = 0;
1456:
1457:       /* ######################################### */
1458:       /* if: 16 / 64bit data recieve */
1459:     } else if ((t == 0x80) ||
1460:                (t == 0x81)) {
1461:       int offset;
1462:       if (t == 0x80) { /* 64bit */
1463:         offset = 8;
1464:       } else { /* 16bit */
1465:         offset = 2;
1466:       }
1467:       if (xbee.log) {
1468:         xbee_log("Packet type: %d-bit RX Data (0x%02X)",((t == 0x80)?64:16),t);
1469:         xbee_logc("%d-bit Address: ",((t == 0x80)?64:16));
1470:         for (j=0;j<offset;j++) {
1471:           fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
1472:         }
1473:         fprintf(xbee.log,"\n");
1474:         xbee_log("RSSI: -%ddB",d[offset]);
1475:         if (d[offset + 1] & 0x02) xbee_log("Options: Address Broadcast");
1476:         if (d[offset + 1] & 0x03) xbee_log("Options: PAN Broadcast");
1477:       }
1478:       p->dataPkt = TRUE;
1479:       p->txStatusPkt = FALSE;
1480:       p->modemStatusPkt = FALSE;
1481:       p->remoteATPkt = FALSE;
1482:       p->IOPkt = FALSE;
1483:
1484:       if (t == 0x80) { /* 64bit */
1485:         p->type = xbee_64bitData;
1486:
1487:         p->sAddr64 = TRUE;
1488:
1489:         p->Addr64[0] = d[0];
1490:         p->Addr64[1] = d[1];
1491:         p->Addr64[2] = d[2];
1492:         p->Addr64[3] = d[3];
1493:         p->Addr64[4] = d[4];
1494:         p->Addr64[5] = d[5];
1495:         p->Addr64[6] = d[6];
1496:         p->Addr64[7] = d[7];
1497:       } else { /* 16bit */
1498:         p->type = xbee_16bitData;
1499:
1500:         p->sAddr64 = FALSE;
1501:
1502:         p->Addr16[0] = d[0];
1503:         p->Addr16[1] = d[1];
1504:       }
1505:
1506:       /* save the RSSI / signal strength
1507:          this can be used with printf as:
1508:          printf("-%ddB\n",p->RSSI); */
1509:       p->RSSI = d[offset];
1510:
1511:       p->status = d[offset + 1];
1512:
1513:       /* copy in the data */
1514:       p->datalen = i-(offset + 1);
1515:       for (;i>offset + 1;i--) p->data[i-(offset + 2)] = d[i];
1516:
1517:       /* ######################################### */
1518:       /* if: 16 / 64bit I/O recieve */
1519:     } else if ((t == 0x82) ||
1520:                (t == 0x83)) {
1521:       int offset;
1522:       if (t == 0x82) { /* 64bit */
1523:         p->type = xbee_64bitIO;
1524:
1525:         p->sAddr64 = TRUE;
1526:
1527:         p->Addr64[0] = d[0];
1528:         p->Addr64[1] = d[1];
1529:         p->Addr64[2] = d[2];
1530:         p->Addr64[3] = d[3];
```

```
1531:         p->Addr64[4] = d[4];
1532:         p->Addr64[5] = d[5];
1533:         p->Addr64[6] = d[6];
1534:         p->Addr64[7] = d[7];
1535:
1536:         offset = 8;
1537:         p->samples = d[10];
1538:       } else { /* 16bit */
1539:         p->type = xbee_16bitIO;
1540:
1541:         p->sAddr64 = FALSE;
1542:
1543:         p->Addr16[0] = d[0];
1544:         p->Addr16[1] = d[1];
1545:
1546:         offset = 2;
1547:         p->samples = d[4];
1548:       }
1549:       if (p->samples > 1) {
1550:         p = Xrealloc(p, sizeof(xbee_pkt) + (sizeof(xbee_sample) * (p->samples - 1)));
1551:       }
1552:       if (xbee.log) {
1553:         xbee_logc("Packet type: %d-bit RX I/O Data (0x%02X)\n",((t == 0x82)?64:16),t);
1554:         xbee_logc("%d-bit Address: ",((t == 0x82)?64:16));
1555:         for (j = 0; j < offset; j++) {
1556:           fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
1557:         }
1558:         fprintf(xbee.log,"\n");
1559:         xbee_log("RSSI: -%ddB",d[offset]);
1560:         if (d[9] & 0x02) xbee_log("Options: Address Broadcast");
1561:         if (d[9] & 0x02) xbee_log("Options: PAN Broadcast");
1562:         xbee_log("Samples: %d",d[offset + 2]);
1563:       }
1564:       i = offset + 5;
1565:
1566:       /* never returns data */
1567:       p->datalen = 0;
1568:
1569:       p->dataPkt = FALSE;
1570:       p->txStatusPkt = FALSE;
1571:       p->modemStatusPkt = FALSE;
1572:       p->remoteATPkt = FALSE;
1573:       p->IOPkt = TRUE;
1574:
1575:       /* save the RSSI / signal strength
1576:          this can be used with printf as:
1577:          printf("-%ddB\n",p->RSSI); */
1578:       p->RSSI = d[offset];
1579:
1580:       p->status = d[offset + 1];
1581:
1582:       /* each sample is split into its own packet here, for simplicity */
1583:       for (o = 0; o < p->samples; o++) {
1584:         if (xbee.log) {
1585:           xbee_log("--- Sample %3d ------------", o);
1586:         }
1587:
1588:         /* parse the io data */
1589:         i = xbee_parse_io(p, d, offset + 3, i, o);
1590:       }
1591:       if (xbee.log) {
1592:         xbee_log("----------------------------");
1593:       }
1594:
1595:       /* ########################################## */
1596:       /* if: Unknown */
1597:     } else {
1598:       if (xbee.log) {
1599:         xbee_log("Packet type: Unknown (0x%02X)",t);
1600:       }
1601:       p->type = xbee_unknown;
1602:     }
1603:     p->next = NULL;
1604:
1605:     /* lock the connection mutex */
1606:     xbee_mutex_lock(xbee.conmutex);
1607:
1608:     con = xbee.conlist;
1609:     hasCon = 0;
1610:     while (con) {
1611:       if (xbee_matchpktcon(p,con)) {
1612:         hasCon = 1;
1613:         break;
1614:       }
1615:       con = con->next;
```

```
1616:     }
1617:
1618:     /* unlock the connection mutex */
1619:     xbee_mutex_unlock(xbee.conmutex);
1620:
1621:     /* if the packet doesn't have a connection, don't add it! */
1622:     if (!hasCon) {
1623:       Xfree(p);
1624:       if (xbee.log) {
1625:         xbee_log("Connectionless packet... discarding!");
1626:       }
1627:       continue;
1628:     }
1629:
1630:     /* lock the packet mutex, so we can safely add the packet to the list */
1631:     xbee_mutex_lock(xbee.pktmutex);
1632:
1633:     /* if: the list is empty */
1634:     if (!xbee.pktlist) {
1635:       /* start the list! */
1636:       xbee.pktlist = p;
1637:     } else if (xbee.pktlast) {
1638:       /* add the packet to the end */
1639:       xbee.pktlast->next = p;
1640:     } else {
1641:       /* pktlast wasnt set... look for the end and then set it */
1642:       i = 0;
1643:       q = xbee.pktlist;
1644:       while (q->next) {
1645:         q = q->next;
1646:         i++;
1647:       }
1648:       q->next = p;
1649:       xbee.pktcount = i;
1650:     }
1651:     xbee.pktlast = p;
1652:     xbee.pktcount++;
1653:
1654:     /* unlock the packet mutex */
1655:     xbee_mutex_unlock(xbee.pktmutex);
1656:
1657:     if (xbee.log) {
1658:       xbee_log("--========================--");
1659:       xbee_log("Packets: %d",xbee.pktcount);
1660:     }
1661:
1662:     p = q = NULL;
1663:   }
1664:   return 0;
1665: }
1666:
1667: /* ########################################################################
1668:    xbee_getbyte - INTERNAL
1669:    waits for an escaped byte of data */
1670: static unsigned char xbee_getbyte(void) {
1671:   unsigned char c;
1672:
1673:   ISREADY;
1674:
1675:   /* take a byte */
1676:   c = xbee_getrawbyte();
1677:   /* if its escaped, take another and un-escape */
1678:   if (c == 0x7D) c = xbee_getrawbyte() ^ 0x20;
1679:
1680:   return (c & 0xFF);
1681: }
1682:
1683: /* ########################################################################
1684:    xbee_getrawbyte - INTERNAL
1685:    waits for a raw byte of data */
1686: static unsigned char xbee_getrawbyte(void) {
1687:   int ret;
1688:   unsigned char c = 0x00;
1689:
1690:   ISREADY;
1691:
1692:   /* the loop is just incase there actually isnt a byte there to be read... */
1693:   do {
1694:     /* wait for a read to be possible */
1695:     if ((ret = xbee_select(NULL)) == -1) {
1696:       perror("libxbee:xbee_getrawbyte()");
1697:       exit(1);
1698:     }
1699:     if (!xbee.listenrun) break;
1700:     if (ret == 0) continue;
```

```
1701:
1702:     /* read 1 character */
1703:     xbee_read(&c,1);
1704: #ifdef _WIN32 /* ---- */
1705:     ret = xbee.ttyr;
1706:     if (ret == 0) {
1707:       usleep(10);
1708:       continue;
1709:     }
1710: #endif /* ----------- */
1711:   } while (0);
1712:
1713:   return (c & 0xFF);
1714: }
1715:
1716: /* ####################################################################
1717:    xbee_send_pkt - INTERNAL
1718:    sends a complete packet of data */
1719: static void xbee_send_pkt(t_data *pkt) {
1720:   ISREADY;
1721:
1722:   /* lock the send mutex */
1723:   xbee_mutex_lock(xbee.sendmutex);
1724:
1725:   /* write and flush the data */
1726:   xbee_write(pkt->data,pkt->length);
1727:
1728:   /* unlock the mutex */
1729:   xbee_mutex_unlock(xbee.sendmutex);
1730:
1731:   if (xbee.log) {
1732:     int i,x,y;
1733:     /* prints packet in hex byte-by-byte */
1734:     xbee_logc("TX Packet:");
1735:     for (i=0,x=0,y=0;i<pkt->length;i++,x--) {
1736:       if (x == 0) {
1737:         fprintf(xbee.log,"\n  0x%04X | ",y);
1738:         x = 0x8;
1739:         y += x;
1740:       }
1741:       if (x == 4) {
1742:         fprintf(xbee.log,"  ");
1743:       }
1744:       fprintf(xbee.log,"0x%02X ",pkt->data[i]);
1745:     }
1746:     fprintf(xbee.log,"\n");
1747:   }
1748:
1749:   /* free the packet */
1750:   Xfree(pkt);
1751: }
1752:
1753: /* ####################################################################
1754:    xbee_make_pkt - INTERNAL
1755:    adds delimiter field
1756:    calculates length and checksum
1757:    escapes bytes */
1758: static t_data *xbee_make_pkt(unsigned char *data, int length) {
1759:   t_data *pkt;
1760:   unsigned int l, i, o, t, x, m;
1761:   char d = 0;
1762:
1763:   ISREADY;
1764:
1765:   /* check the data given isnt too long
1766:      100 bytes maximum payload + 12 bytes header information */
1767:   if (length > 100 + 12) return NULL;
1768:
1769:   /* calculate the length of the whole packet
1770:      start, length (MSB), length (LSB), DATA, checksum */
1771:   l = 3 + length + 1;
1772:
1773:   /* prepare memory */
1774:   pkt = Xcalloc(sizeof(t_data));
1775:
1776:   /* put start byte on */
1777:   pkt->data[0] = 0x7E;
1778:
1779:   /* copy data into packet */
1780:   for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
1781:     /* if: its time for the checksum */
1782:     if (i == length) d = M8((0xFF - M8(t)));
1783:     /* if: its time for the high length byte */
1784:     else if (m == 1) d = M8(length >> 8);
1785:     /* if: its time for the low length byte */
```

```
1786:     else if (m == 2) d = M8(length);
1787:     /* if: its time for the normal data */
1788:     else if (m > 2) d = data[i];
1789:
1790:     x = 0;
1791:     /* check for any escapes needed */
1792:     if ((d == 0x11) || /* XON */
1793:         (d == 0x13) || /* XOFF */
1794:         (d == 0x7D) || /* Escape */
1795:         (d == 0x7E)) { /* Frame Delimiter */
1796:       l++;
1797:       pkt->data[o++] = 0x7D;
1798:       x = 1;
1799:     }
1800:
1801:     /* move data in */
1802:     pkt->data[o] = ((!x)?d:d^0x20);
1803:     if (m > 2) {
1804:       i++;
1805:       t += d;
1806:     }
1807:   }
1808:
1809:   /* remember the length */
1810:   pkt->length = l;
1811:
1812:   return pkt;
1813: }
```