```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 "globals.h"
#include "api.h"

/* ready flag.
   needs to be set to -1 so that the listen thread can begin.
   then 1 so that functions can be used (after setup of course...) */
volatile int xbee_ready = 0;

#ifdef __GNUC__ /* ---- */
/* ######################################################################## */
/* ### Unix Functions ##################################################### */
/* ######################################################################## */
static int xbee_select(struct timeval *timeout) {
  fd_set fds;

  FD_ZERO(&fds);
  FD_SET(xbee.ttyfd, &fds);

  return select(xbee.ttyfd+1, &fds, NULL, NULL, timeout);
}

#else           /* ---- */
/* ######################################################################## */
/* ### Win32 Functions #################################################### */
/* ######################################################################## */
BOOL APIENTRY DllMain(HANDLE hModule, DWORD dwReason, LPVOID lpReserved) {
  if ((dwReason == DLL_PROCESS_DETACH || dwReason == DLL_THREAD_DETACH) && xbee_ready == 1) {
    xbee_end();
  } else if ((dwReason == DLL_PROCESS_ATTACH || dwReason == DLL_THREAD_ATTACH) && xbee_ready == 0) {
    memset(&xbee,0,sizeof(xbee));
  }
  return TRUE;
}

void xbee_free(void *ptr) {
  if (!ptr) return;
  free(ptr);
}

/* These silly little functions are required for VB6
   - it freaks out when you call a function that uses va_args... */
xbee_con *xbee_newcon_simple(unsigned char frameID, xbee_types type) {
  return xbee_newcon(frameID,type);
}
xbee_con *xbee_newcon_16bit(unsigned char frameID, xbee_types type, int addr) {
  return xbee_newcon(frameID,type, addr);
}
xbee_con *xbee_newcon_64bit(unsigned char frameID, xbee_types type, int addrL, int addrH) {
  return xbee_newcon(frameID,type,addrL,addrH);
}

static int xbee_select(struct timeval *timeout) {
  int evtMask = 0;
  COMSTAT status;
  int ret;

  for (;;) {
    /* find out how many bytes are in the Rx buffer... */
    if (ClearCommError(xbee.tty,NULL,&status) && (status.cbInQue > 0)) {
      /* if there is data... return! */
      return 1; /*status.cbInQue;*/
    } else if (timeout && timeout->tv_sec == 0 && timeout->tv_usec == 0) {
      return 0;
    }

```

```
  86:     /* otherwise wait for an Rx event... */
  87:     xbee.ttyovrs.hEvent = CreateEvent(NULL,TRUE,FALSE,NULL);
  88:     if (!WaitCommEvent(xbee.tty,&evtMask,&xbee.ttyovrs)) {
  89:       if (GetLastError() == ERROR_IO_PENDING) {
  90:         DWORD timeoutval;
  91:         if (!timeout) {
  92:           timeoutval = INFINITE;
  93:         } else {
  94:           timeoutval = (timeout->tv_sec * 1000) + (timeout->tv_usec / 1000);
  95:         }
  96:         ret = WaitForSingleObject(xbee.ttyovrs.hEvent,timeoutval);
  97:         if (ret == WAIT_TIMEOUT) return 0;
  98:       } else {
  99:         usleep(1000); /* 1 ms */
 100:       }
 101:     }
 102:     CloseHandle(xbee.ttyovrs.hEvent);
 103:   }
 104:
 105:   /* always return -1 for now... */
 106:   return -1;
 107: }
 108:
 109: int xbee_write(const void *ptr, size_t size) {
 110:   if (!WriteFile(xbee.tty, ptr, size, NULL, &xbee.ttyovrw) &&
 111:       (GetLastError() != ERROR_IO_PENDING)) return -1;
 112:
 113:   if (!GetOverlappedResult(xbee.tty, &xbee.ttyovrw, &xbee.ttyw, TRUE)) return -1;
 114:   return xbee.ttyw;
 115: }
 116:
 117: int xbee_read(void *ptr, size_t size) {
 118:   if (!ReadFile(xbee.tty, ptr, size, NULL, &xbee.ttyovrr) &&
 119:       (GetLastError() != ERROR_IO_PENDING)) return -1;
 120:   if (!GetOverlappedResult(xbee.tty, &xbee.ttyovrr, &xbee.ttyr, TRUE)) return -1;
 121:
 122:   return xbee.ttyr;
 123: }
 124:
 125: const char *xbee_svn_version(void) {
 126:   return "Win32";
 127: }
 128:
 129: #endif          /* ---- */
 130:
 131: #ifdef __UMAKEFILE
 132: /* for embedded compiling */
 133: const char *xbee_svn_version(void) {
 134:   return "Embedded";
 135: }
 136: #endif
 137:
 138: /* ################################################################### */
 139: /* ### Memory Handling ############################################## */
 140: /* ################################################################### */
 141:
 142: /* malloc wrapper function */
 143: static void *Xmalloc(size_t size) {
 144:   void *t;
 145:   t = malloc(size);
 146:   if (!t) {
 147:     /* uhoh... thats pretty bad... */
 148:     perror("libxbee:malloc()");
 149:     exit(1);
 150:   }
 151:   return t;
 152: }
 153:
 154: /* calloc wrapper function */
 155: static void *Xcalloc(size_t size) {
 156:   void *t;
 157:   t = calloc(1, size);
 158:   if (!t) {
 159:     /* uhoh... thats pretty bad... */
 160:     perror("libxbee:calloc()");
 161:     exit(1);
 162:   }
 163:   return t;
 164: }
 165:
 166: /* realloc wrapper function */
 167: static void *Xrealloc(void *ptr, size_t size) {
 168:   void *t;
 169:   t = realloc(ptr,size);
 170:   if (!t) {
```

```
171:     /* uhoh... thats pretty bad... */
172:     perror("libxbee:realloc()");
173:     exit(1);
174:   }
175:   return t;
176: }
177:
178: /* free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
179: static void Xfree2(void **ptr) {
180:   if (!*ptr) return;
181:   free(*ptr);
182:   *ptr = NULL;
183: }
184:
185: /* ######################################################################## */
186: /* ### Helper Functions ################################################### */
187: /* ######################################################################## */
188:
189: /* ########################################################################
190:    returns 1 if the packet has data for the digital input else 0 */
191: int xbee_hasdigital(xbee_pkt *pkt, int sample, int input) {
192:   int mask = 0x0001;
193:   if (input < 0 || input > 7) return 0;
194:   if (sample >= pkt->samples) return 0;
195:
196:   mask <<= input;
197:   return !!(pkt->IOdata[sample].IOmask & mask);
198: }
199:
200: /* ########################################################################
201:    returns 1 if the digital input is high else 0 (or 0 if no digital data present) */
202: int xbee_getdigital(xbee_pkt *pkt, int sample, int input) {
203:   int mask = 0x0001;
204:   if (!xbee_hasdigital(pkt,sample,input)) return 0;
205:
206:   mask <<= input;
207:   return !!(pkt->IOdata[sample].IOdigital & mask);
208: }
209:
210: /* ########################################################################
211:    returns 1 if the packet has data for the analog input else 0 */
212: int xbee_hasanalog(xbee_pkt *pkt, int sample, int input) {
213:   int mask = 0x0200;
214:   if (input < 0 || input > 5) return 0;
215:   if (sample >= pkt->samples) return 0;
216:
217:   mask <<= input;
218:   return !!(pkt->IOdata[sample].IOmask & mask);
219: }
220:
221: /* ########################################################################
222:    returns analog input as a voltage if vRef is non-zero, else raw value (or 0 if no analog data present) */
223: double xbee_getanalog(xbee_pkt *pkt, int sample, int input, double Vref) {
224:   if (!xbee_hasanalog(pkt,sample,input)) return 0;
225:
226:   if (Vref) return (Vref / 1023) * pkt->IOdata[sample].IOanalog[input];
227:   return pkt->IOdata[sample].IOanalog[input];
228: }
229:
230: /* ######################################################################## */
231: /* ### XBee Functions ##################################################### */
232: /* ######################################################################## */
233:
234: static void xbee_logf(const char *logformat, const char *function, char *format, ...) {
235:   char buf[128];
236:   va_list ap;
237:   FILE *log;
238:   va_start(ap,format);
239:   vsnprintf(buf,127,format,ap);
240:   va_end(ap);
241:   if (xbee.log) {
242:     log = xbee.log;
243:   } else {
244:     log = stderr;
245:   }
246:   fprintf(log,logformat,function,buf);
247: }
248:
249: /* ########################################################################
250:    xbee_sendAT - INTERNAL
251:    allows for an at command to be send, and the reply to be captured */
252: static int xbee_sendAT(char *command, char *retBuf, int retBuflen) {
253:   return xbee_sendATdelay(0,0,command,retBuf, retBuflen);
254: }
255: static int xbee_sendATdelay(int preDelay, int postDelay, char *command, char *retBuf, int retBuflen) {
```

```
256:   struct timeval to;
257:
258:   int ret;
259:   int bufi = 0;
260:
261:   /* if there is a preDelay given, then use it and a bit more */
262:   if (preDelay) usleep(preDelay * 1200);
263:
264:   /* get rid of any pre-command sludge... */
265:   memset(&to, 0, sizeof(to));
266:   ret = xbee_select(&to);
267:   if (ret > 0) {
268:     char t[128];
269:     while (xbee_read(t,127));
270:   }
271:
272:   /* send the requested command */
273:   if (xbee.log) xbee_log("sendATdelay: Sending '%s'", command);
274:   xbee_write(command, strlen(command));
275:
276:   /* if there is a postDelay, then use it */
277:   if (postDelay) {
278:     usleep(postDelay * 900);
279:
280:     /* get rid of any post-command sludge... */
281:     memset(&to, 0, sizeof(to));
282:     ret = xbee_select(&to);
283:     if (ret > 0) {
284:       char t[128];
285:       while (xbee_read(t,127));
286:     }
287:   }
288:
289:   /* retrieve the data */
290:   memset(retBuf, 0, retBuflen);
291:   memset(&to, 0, sizeof(to));
292:   /* select on the xbee fd... wait at most 200ms for the response */
293:   to.tv_usec = 200000;
294:   if ((ret = xbee_select(&to)) == -1) {
295:     perror("libxbee:xbee_sendATdelay()");
296:     exit(1);
297:   }
298:
299:   if (!ret) {
300:     /* timed out, and there is nothing to be read */
301:     if (xbee.log) xbee_log("sendATdelay: No Data to read - Timeout...");
302:     return 1;
303:   }
304:
305:   /* check for any dribble... */
306:   do {
307:     /* if there is actually no space in the retBuf then break out */
308:     if (bufi >= retBuflen - 1) {
309:       break;
310:     }
311:
312:     /* read as much data as is possible into retBuf */
313:     if ((ret = xbee_read(&retBuf[bufi], retBuflen - bufi - 1)) == 0) {
314:       break;
315:     }
316:
317:     /* advance the 'end of string' pointer */
318:     bufi += ret;
319:
320:     /* wait at most 100ms for any more data */
321:     memset(&to, 0, sizeof(to));
322:     to.tv_usec = 100000;
323:     if ((ret = xbee_select(&to)) == -1) {
324:       perror("libxbee:xbee_sendATdelay()");
325:       exit(1);
326:     }
327:
328:     /* loop while data was read */
329:   } while (ret);
330:
331:   if (!bufi) {
332:     if (xbee.log) xbee_log("sendATdelay: No response...");
333:     return 1;
334:   }
335:
336:   /* terminate the string */
337:   retBuf[bufi] = '\0';
338:
339:   if (xbee.log) xbee_log("sendATdelay: Recieved '%s'",retBuf);
340:   return 0;
```

```
341: }
342:
343:
344: /* ###################################################################
345:    xbee_start
346:    sets up the correct API mode for the xbee
347:    cmdSeq  = CC
348:    cmdTime = GT */
349: static int xbee_startAPI(void) {
350:   char buf[256];
351:
352:   if (xbee.cmdSeq == 0 || xbee.cmdTime == 0) return 1;
353:
354:   /* setup the command sequence string */
355:   memset(buf,xbee.cmdSeq,3);
356:   buf[3] = '\0';
357:
358:   /* try the command sequence */
359:   if (xbee_sendATdelay(xbee.cmdTime, xbee.cmdTime, buf, buf, sizeof(buf))) {
360:     /* if it failed... try just entering 'AT' which should return OK */
361:     if (xbee_sendAT("AT\r\n", buf, sizeof(buf)) || strncmp(buf,"OK\r",3)) return 1;
362:   } else if (strncmp(&buf[strlen(buf)-3],"OK\r",3)) {
363:     /* if data was returned, but it wasn't OK... then something went wrong! */
364:     return 1;
365:   }
366:
367:   /* get the current API mode */
368:   if (xbee_sendAT("ATAP\r\n", buf, sizeof(buf))) return 1;
369:   buf[1] = '\0';
370:   xbee.oldAPI = atoi(buf);
371:
372:   if (xbee.oldAPI != 2) {
373:     /* if it wasnt set to mode 2 already, then set it to mode 2 */
374:     if (xbee_sendAT("ATAP2\r\n", buf, sizeof(buf)) || strncmp(buf,"OK\r",3)) return 1;
375:   }
376:
377:   /* quit from command mode, ready for some packets! :) */
378:   if (xbee_sendAT("ATCN\r\n", buf, 4) || strncmp(buf,"OK\r",3)) return 1;
379:
380:   return 0;
381: }
382:
383: /* ###################################################################
384:    xbee_end
385:    resets the API mode to the saved value - you must have called xbee_setup[log]API */
386: int xbee_end(void) {
387:   int ret = 1;
388:   xbee_con *con, *ncon;
389:   xbee_pkt *pkt, *npkt;
390:
391:   ISREADY;
392:   if (xbee.log) fprintf(xbee.log,"libxbee: Stopping...\n");
393:
394:   /* if the api mode was not 2 to begin with then put it back */
395:   if (xbee.oldAPI == 2) {
396:     ret = 0;
397:   } else {
398:     int to = 5;
399:
400:     con = xbee_newcon('I',xbee_localAT);
401:     xbee_senddata(con,"AP%c",xbee.oldAPI);
402:
403:     pkt = NULL;
404:
405:     while (!pkt && to--) {
406:       pkt = xbee_getpacketwait(con);
407:     }
408:     if (pkt) {
409:       ret = pkt->status;
410:       Xfree(pkt);
411:     }
412:     xbee_endcon(con);
413:   }
414:
415:   /* stop listening for data... either after timeout or next char read which ever is first */
416:   xbee.listenrun = 0;
417:   xbee_thread_kill(xbee.listent,0);
418:   /* xbee_* functions may no longer run... */
419:   xbee_ready = 0;
420:
421:   if (xbee.log) fflush(xbee.log);
422:
423:   /* nullify everything */
424:
425:   /* free all connections */
```

```
426:   con = xbee.conlist;
427:   xbee.conlist = NULL;
428:   while (con) {
429:     ncon = con->next;
430:     Xfree(con);
431:     con = ncon;
432:   }
433:
434:   /* free all packets */
435:   xbee.pktlast = NULL;
436:   pkt = xbee.pktlist;
437:   xbee.pktlist = NULL;
438:   while (pkt) {
439:     npkt = pkt->next;
440:     Xfree(pkt);
441:     pkt = npkt;
442:   }
443:
444:   /* destroy mutexes */
445:   xbee_mutex_destroy(xbee.conmutex);
446:   xbee_mutex_destroy(xbee.pktmutex);
447:   xbee_mutex_destroy(xbee.sendmutex);
448:
449:   /* close the serial port */
450:   Xfree(xbee.path);
451: #ifdef __GNUC__
452:   if (xbee.tty) fclose(xbee.tty);
453:   if (xbee.ttyfd) close(xbee.ttyfd);
454: #else
455:   if (xbee.tty) CloseHandle(xbee.tty);
456: #endif
457:
458:   /* close log and tty */
459:   if (xbee.log) {
460:     fprintf(xbee.log,"libxbee: Stopped! (%s)\n",xbee_svn_version());
461:     fflush(xbee.log);
462:     fclose(xbee.log);
463:   }
464:
465:   /* wipe everything else... */
466:   memset(&xbee,0,sizeof(xbee));
467:
468:   return ret;
469: }
470:
471: /* ####################################################################
472:    xbee_setup
473:    opens xbee serial port & creates xbee listen thread
474:    the xbee must be configured for API mode 2
475:    THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
476: int xbee_setup(char *path, int baudrate) {
477:   return xbee_setuplogAPI(path,baudrate,0,0,0);
478: }
479: int xbee_setuplog(char *path, int baudrate, int logfd) {
480:   return xbee_setuplogAPI(path,baudrate,logfd,0,0);
481: }
482: int xbee_setupAPI(char *path, int baudrate, char cmdSeq, int cmdTime) {
483:   return xbee_setuplogAPI(path,baudrate,0,cmdSeq,cmdTime);
484: }
485: int xbee_setuplogAPI(char *path, int baudrate, int logfd, char cmdSeq, int cmdTime) {
486: #ifdef __GNUC__ /* ---- */
487:   struct flock fl;
488:   struct termios tc;
489:   speed_t chosenbaud;
490: #else           /* ---- */
491:   int chosenbaud;
492:   DCB tc;
493:   int evtMask;
494:   COMMTIMEOUTS timeouts;
495: #endif          /* ---- */
496:   t_info info;
497:
498:   memset(&xbee,0,sizeof(xbee));
499:
500: #ifdef DEBUG
501:   /* logfd or stdout */
502:   xbee.logfd = ((logfd)?logfd:1);
503: #else
504:   xbee.logfd = logfd;
505: #endif
506:   if (xbee.logfd) {
507:     xbee.log = fdopen(xbee.logfd,"w");
508:     if (!xbee.log) {
509:       /* errno == 9 is bad file descriptor (probrably not provided) */
510:       if (errno != 9) perror("xbee_setup(): Failed opening logfile");
```

```
511:       xbee.logfd = 0;
512:     } else {
513:       /* set to line buffer - ensure lines are written to file when complete */
514: #ifdef __GNUC__ /* ---- */
515:       setvbuf(xbee.log,NULL,_IOLBF,BUFSIZ);
516: #else          /* ---- */
517:       /* Win32 is rubbish... so we have to completely disable buffering... */
518:       setvbuf(xbee.log,NULL,_IONBF,BUFSIZ);
519: #endif         /* ---- */
520:     }
521:   }
522:
523:   if (xbee.log) fprintf(xbee.log,"libxbee: Starting (%s)...\n",xbee_svn_version());
524:
525: #ifdef __GNUC__ /* ---- */
526:   /* select the baud rate */
527:   switch (baudrate) {
528:   case 1200:  chosenbaud = B1200;   break;
529:   case 2400:  chosenbaud = B2400;   break;
530:   case 4800:  chosenbaud = B4800;   break;
531:   case 9600:  chosenbaud = B9600;   break;
532:   case 19200: chosenbaud = B19200;  break;
533:   case 38400: chosenbaud = B38400;  break;
534:   case 57600: chosenbaud = B57600;  break;
535:   case 115200:chosenbaud = B115200; break;
536:   default:
537:     fprintf(stderr,"%s(): Unknown or incompatiable baud rate specified... (%d)\n",__FUNCTION__,baudrate);
538:     return -1;
539:   };
540: #endif         /* ---- */
541:
542:   /* setup the connection stuff */
543:   xbee.conlist = NULL;
544:
545:   /* setup the packet stuff */
546:   xbee.pktlist = NULL;
547:   xbee.pktlast = NULL;
548:   xbee.pktcount = 0;
549:   xbee.listenrun = 1;
550:
551:   /* setup the mutexes */
552:   if (xbee_mutex_init(xbee.conmutex)) {
553:     perror("xbee_setup():xbee_mutex_init(conmutex)");
554:     return -1;
555:   }
556:   if (xbee_mutex_init(xbee.pktmutex)) {
557:     perror("xbee_setup():xbee_mutex_init(pktmutex)");
558:     xbee_mutex_destroy(xbee.conmutex);
559:     return -1;
560:   }
561:   if (xbee_mutex_init(xbee.sendmutex)) {
562:     perror("xbee_setup():xbee_mutex_init(sendmutex)");
563:     xbee_mutex_destroy(xbee.conmutex);
564:     xbee_mutex_destroy(xbee.pktmutex);
565:     return -1;
566:   }
567:
568:   /* take a copy of the XBee device path */
569:   if ((xbee.path = Xmalloc(sizeof(char) * (strlen(path) + 1))) == NULL) {
570:     perror("xbee_setup():Xmalloc(path)");
571:     xbee_mutex_destroy(xbee.conmutex);
572:     xbee_mutex_destroy(xbee.pktmutex);
573:     xbee_mutex_destroy(xbee.sendmutex);
574:     return -1;
575:   }
576:   strcpy(xbee.path,path);
577:
578: #ifdef __GNUC__ /* ---- */
579:   /* open the serial port as a file descriptor */
580:   if ((xbee.ttyfd = open(path,O_RDWR | O_NOCTTY | O_NONBLOCK)) == -1) {
581:     perror("xbee_setup():open()");
582:     xbee_mutex_destroy(xbee.conmutex);
583:     xbee_mutex_destroy(xbee.pktmutex);
584:     xbee_mutex_destroy(xbee.sendmutex);
585:     Xfree(xbee.path);
586:     return -1;
587:   }
588:
589:   /* lock the file */
590:   fl.l_type = F_WRLCK | F_RDLCK;
591:   fl.l_whence = SEEK_SET;
592:   fl.l_start = 0;
593:   fl.l_len = 0;
594:   fl.l_pid = getpid();
595:   if (fcntl(xbee.ttyfd, F_SETLK, &fl) == -1) {
```

```
596:     perror("xbee_setup():fcntl()");
597:     xbee_mutex_destroy(xbee.conmutex);
598:     xbee_mutex_destroy(xbee.pktmutex);
599:     xbee_mutex_destroy(xbee.sendmutex);
600:     Xfree(xbee.path);
601:     close(xbee.ttyfd);
602:     return -1;
603:   }
604:
605:   /* open the serial port as a FILE* */
606:   if ((xbee.tty = fdopen(xbee.ttyfd,"r+")) == NULL) {
607:     perror("xbee_setup():fdopen()");
608:     xbee_mutex_destroy(xbee.conmutex);
609:     xbee_mutex_destroy(xbee.pktmutex);
610:     xbee_mutex_destroy(xbee.sendmutex);
611:     Xfree(xbee.path);
612:     close(xbee.ttyfd);
613:     return -1;
614:   }
615:
616:   /* flush the serial port */
617:   fflush(xbee.tty);
618:
619:   /* disable buffering */
620:   setvbuf(xbee.tty,NULL,_IONBF,BUFSIZ);
621:
622:   /* setup the baud rate and other io attributes */
623:   tcgetattr(xbee.ttyfd, &tc);
624:   /* input flags */
625:   tc.c_iflag &= ~ IGNBRK;           /* enable ignoring break */
626:   tc.c_iflag &= ~(IGNPAR | PARMRK); /* disable parity checks */
627:   tc.c_iflag &= ~ INPCK;            /* disable parity checking */
628:   tc.c_iflag &= ~ ISTRIP;           /* disable stripping 8th bit */
629:   tc.c_iflag &= ~(INLCR | ICRNL);   /* disable translating NL <-> CR */
630:   tc.c_iflag &= ~ IGNCR;            /* disable ignoring CR */
631:   tc.c_iflag &= ~(IXON | IXOFF);    /* disable XON/XOFF flow control */
632:   /* output flags */
633:   tc.c_oflag &= ~ OPOST;            /* disable output processing */
634:   tc.c_oflag &= ~(ONLCR | OCRNL);   /* disable translating NL <-> CR */
635:   tc.c_oflag &= ~ OFILL;            /* disable fill characters */
636:   /* control flags */
637:   tc.c_cflag |=   CREAD;            /* enable reciever */
638:   tc.c_cflag &= ~ PARENB;           /* disable parity */
639:   tc.c_cflag &= ~ CSTOPB;           /* disable 2 stop bits */
640:   tc.c_cflag &= ~ CSIZE;            /* remove size flag... */
641:   tc.c_cflag |=   CS8;              /* ...enable 8 bit characters */
642:   tc.c_cflag |=   HUPCL;            /* enable lower control lines on close - hang up */
643:   /* local flags */
644:   tc.c_lflag &= ~ ISIG;             /* disable generating signals */
645:   tc.c_lflag &= ~ ICANON;           /* disable canonical mode - line by line */
646:   tc.c_lflag &= ~ ECHO;             /* disable echoing characters */
647:   tc.c_lflag &= ~ ECHONL;           /* ??? */
648:   tc.c_lflag &= ~ NOFLSH;           /* disable flushing on SIGINT */
649:   tc.c_lflag &= ~ IEXTEN;           /* disable input processing */
650:   /* control characters */
651:   memset(tc.c_cc,0,sizeof(tc.c_cc));
652:   /* i/o rates */
653:   cfsetspeed(&tc, chosenbaud);     /* set i/o baud rate */
654:   tcsetattr(xbee.ttyfd, TCSANOW, &tc);
655:   tcflow(xbee.ttyfd, TCOON|TCION); /* enable input & output transmission */
656: #else
657:   /* open the serial port */
658:   xbee.tty = CreateFile(TEXT(path),
659:                         GENERIC_READ | GENERIC_WRITE,
660:                         0,    /* exclusive access */
661:                         NULL, /* default security attributes */
662:                         OPEN_EXISTING,
663:                         FILE_FLAG_OVERLAPPED,
664:                         NULL);
665:   if (xbee.tty == INVALID_HANDLE_VALUE) {
666:     perror("xbee_setup():CreateFile()");
667:     xbee_mutex_destroy(xbee.conmutex);
668:     xbee_mutex_destroy(xbee.pktmutex);
669:     xbee_mutex_destroy(xbee.sendmutex);
670:     Xfree(xbee.path);
671:     return -1;
672:   }
673:
674:   GetCommState(xbee.tty, &tc);
675:   tc.BaudRate =          baudrate;
676:   tc.fBinary =           TRUE;
677:   tc.fParity =           FALSE;
678:   tc.fOutxCtsFlow =      FALSE;
679:   tc.fOutxDsrFlow =      FALSE;
680:   tc.fDtrControl =       DTR_CONTROL_DISABLE;
```

```
681:   tc.fDsrSensitivity =   FALSE;
682:   tc.fTXContinueOnXoff = FALSE;
683:   tc.fOutX =             FALSE;
684:   tc.fInX =              FALSE;
685:   tc.fErrorChar =        FALSE;
686:   tc.fNull =             FALSE;
687:   tc.fRtsControl =       RTS_CONTROL_DISABLE;
688:   tc.fAbortOnError =     FALSE;
689:   tc.ByteSize =          8;
690:   tc.Parity =            NOPARITY;
691:   tc.StopBits =          ONESTOPBIT;
692:   SetCommState(xbee.tty, &tc);
693:
694:   timeouts.ReadIntervalTimeout = MAXDWORD;
695:   timeouts.ReadTotalTimeoutMultiplier = 0;
696:   timeouts.ReadTotalTimeoutConstant = 0;
697:   timeouts.WriteTotalTimeoutMultiplier = 0;
698:   timeouts.WriteTotalTimeoutConstant = 0;
699:   SetCommTimeouts(xbee.tty, &timeouts);
700:
701:   GetCommMask(xbee.tty, &evtMask);
702:   evtMask |= EV_RXCHAR;
703:   SetCommMask(xbee.tty, evtMask);
704: #endif          /* ---- */
705:
706:   /* when xbee_end() is called, if this is not 2 then ATAP will be set to this value */
707:   xbee.oldAPI = 2;
708:   xbee.cmdSeq = cmdSeq;
709:   xbee.cmdTime = cmdTime;
710:   if (xbee.cmdSeq && xbee.cmdTime) {
711:     if (xbee_startAPI()) {
712:       if (xbee.log) {
713:         xbee_log("Couldn't communicate with XBee...");
714:       }
715:       xbee_mutex_destroy(xbee.conmutex);
716:       xbee_mutex_destroy(xbee.pktmutex);
717:       xbee_mutex_destroy(xbee.sendmutex);
718:       Xfree(xbee.path);
719: #ifdef __GNUC__
720:       close(xbee.ttyfd);
721: #endif
722:       fclose(xbee.tty);
723:       return -1;
724:     }
725:   }
726:
727:   /* allow the listen thread to start */
728:   xbee_ready = -1;
729:
730:   /* can start xbee_listen thread now */
731:   if (xbee_thread_create(xbee.listent,xbee_listen_wrapper,info)) {
732:     perror("xbee_setup():xbee_thread_create()");
733:     xbee_mutex_destroy(xbee.conmutex);
734:     xbee_mutex_destroy(xbee.pktmutex);
735:     xbee_mutex_destroy(xbee.sendmutex);
736:     Xfree(xbee.path);
737: #ifdef __GNUC__
738:     close(xbee.ttyfd);
739: #endif
740:     fclose(xbee.tty);
741:     return -1;
742:   }
743:
744:   usleep(100);
745:   while (xbee_ready != -2) {
746:     usleep(100);
747:     if (xbee.log) {
748:       xbee_log("Waiting for xbee_listen() to be ready...");
749:     }
750:   }
751:
752:   /* allow other functions to be used! */
753:   xbee_ready = 1;
754:
755:   if (xbee.log) fprintf(xbee.log,"libxbee: Started!\n");
756:
757:   return 0;
758: }
759:
760: /* ################################################################### */
761:    xbee_con
762:    produces a connection to the specified device and frameID
763:    if a connection had already been made, then this connection will be returned */
764: xbee_con *xbee_newcon(unsigned char frameID, xbee_types type, ...) {
765:   xbee_con *con, *ocon;
```

```
766:   unsigned char tAddr[8];
767:   va_list ap;
768:   int t;
769:   int i;
770:
771:   ISREADY;
772:
773:   if (!type || type == xbee_unknown) type = xbee_localAT; /* default to local AT */
774:   else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */
775:
776:   va_start(ap,type);
777:   /* if: 64 bit address expected (2 ints) */
778:   if ((type == xbee_64bitRemoteAT) ||
779:       (type == xbee_64bitData) ||
780:       (type == xbee_64bitIO)) {
781:     t = va_arg(ap, int);
782:     tAddr[0] = (t >> 24) & 0xFF;
783:     tAddr[1] = (t >> 16) & 0xFF;
784:     tAddr[2] = (t >>  8) & 0xFF;
785:     tAddr[3] = (t      ) & 0xFF;
786:     t = va_arg(ap, int);
787:     tAddr[4] = (t >> 24) & 0xFF;
788:     tAddr[5] = (t >> 16) & 0xFF;
789:     tAddr[6] = (t >>  8) & 0xFF;
790:     tAddr[7] = (t      ) & 0xFF;
791:
792:     /* if: 16 bit address expected (1 int) */
793:   } else if ((type == xbee_16bitRemoteAT) ||
794:              (type == xbee_16bitData) ||
795:              (type == xbee_16bitIO)) {
796:     t = va_arg(ap, int);
797:     tAddr[0] = (t >>  8) & 0xFF;
798:     tAddr[1] = (t      ) & 0xFF;
799:     tAddr[2] = 0;
800:     tAddr[3] = 0;
801:     tAddr[4] = 0;
802:     tAddr[5] = 0;
803:     tAddr[6] = 0;
804:     tAddr[7] = 0;
805:
806:     /* otherwise clear the address */
807:   } else {
808:     memset(tAddr,0,8);
809:   }
810:   va_end(ap);
811:
812:   /* lock the connection mutex */
813:   xbee_mutex_lock(xbee.conmutex);
814:
815:   /* are there any connections? */
816:   if (xbee.conlist) {
817:     con = xbee.conlist;
818:     while (con) {
819:       /* if: after a modemStatus, and the types match! */
820:       if ((type == xbee_modemStatus) &&
821:           (con->type == type)) {
822:         xbee_mutex_unlock(xbee.conmutex);
823:         return con;
824:
825:         /* if: after a txStatus and frameIDs match! */
826:       } else if ((type == xbee_txStatus) &&
827:                  (con->type == type) &&
828:                  (frameID == con->frameID)) {
829:         xbee_mutex_unlock(xbee.conmutex);
830:         return con;
831:
832:         /* if: after a localAT, and the frameIDs match! */
833:       } else if ((type == xbee_localAT) &&
834:                  (con->type == type) &&
835:                  (frameID == con->frameID)) {
836:         xbee_mutex_unlock(xbee.conmutex);
837:         return con;
838:
839:         /* if: connection types match, the frameIDs match, and the addresses match! */
840:       } else if ((type == con->type) &&
841:                  (frameID == con->frameID) &&
842:                  (!memcmp(tAddr,con->tAddr,8))) {
843:         xbee_mutex_unlock(xbee.conmutex);
844:         return con;
845:       }
846:
847:       /* if there are more, move along, dont want to loose that last item! */
848:       if (con->next == NULL) break;
849:       con = con->next;
850:     }
```

```
851:
852:     /* keep hold of the last connection... we will need to link it up later */
853:     ocon = con;
854:   }
855:
856:   /* create a new connection and set its attributes */
857:   con = Xcalloc(sizeof(xbee_con));
858:   con->type = type;
859:   /* is it a 64bit connection? */
860:   if ((type == xbee_64bitRemoteAT) ||
861:       (type == xbee_64bitData) ||
862:       (type == xbee_64bitIO)) {
863:     con->tAddr64 = TRUE;
864:   }
865:   con->atQueue = 0; /* queue AT commands? */
866:   con->txDisableACK = 0; /* disable ACKs? */
867:   con->txBroadcast = 0; /* broadcast? */
868:   con->frameID = frameID;
869:   memcpy(con->tAddr,tAddr,8); /* copy in the remote address */
870:
871:   if (xbee.log) {
872:     switch(type) {
873:     case xbee_localAT:
874:       xbee_log("New local AT connection!");
875:       break;
876:     case xbee_16bitRemoteAT:
877:     case xbee_64bitRemoteAT:
878:       xbee_logc("New %d-bit remote AT connection! (to: ",(con->tAddr64?64:16));
879:       for (i=0;i<(con->tAddr64?8:2);i++) {
880:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
881:       }
882:       fprintf(xbee.log,")\n");
883:       break;
884:     case xbee_16bitData:
885:     case xbee_64bitData:
886:       xbee_logc("New %d-bit data connection! (to: ",(con->tAddr64?64:16));
887:       for (i=0;i<(con->tAddr64?8:2);i++) {
888:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
889:       }
890:       fprintf(xbee.log,")\n");
891:       break;
892:     case xbee_16bitIO:
893:     case xbee_64bitIO:
894:       xbee_logc("New %d-bit IO connection! (to: ",(con->tAddr64?64:16));
895:       for (i=0;i<(con->tAddr64?8:2);i++) {
896:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
897:       }
898:       fprintf(xbee.log,")\n");
899:       break;
900:     case xbee_txStatus:
901:       xbee_log("New Tx status connection!");
902:       break;
903:     case xbee_modemStatus:
904:       xbee_log("New modem status connection!");
905:       break;
906:     case xbee_unknown:
907:     default:
908:       xbee_log("New unknown connection!");
909:     }
910:   }
911:
912:   /* make it the last in the list */
913:   con->next = NULL;
914:   /* add it to the list */
915:   if (xbee.conlist) {
916:     ocon->next = con;
917:   } else {
918:     xbee.conlist = con;
919:   }
920:
921:   /* unlock the mutex */
922:   xbee_mutex_unlock(xbee.conmutex);
923:   return con;
924: }
925:
926: /* ################################################################
927:    xbee_conflush
928:    removes any packets that have been collected for the specified
929:    connection */
930: void xbee_flushcon(xbee_con *con) {
931:   xbee_pkt *r, *p, *n;
932:
933:   /* lock the packet mutex */
934:   xbee_mutex_lock(xbee.pktmutex);
935:
```

```
936:   /* if: there are packets */
937:   if ((p = xbee.pktlist) != NULL) {
938:     r = NULL;
939:     /* get all packets for this connection */
940:     do {
941:       /* does the packet match the connection? */
942:       if (xbee_matchpktcon(p,con)) {
943:         /* if it was the first packet */
944:         if (!r) {
945:           /* move the chain along */
946:           xbee.pktlist = p->next;
947:         } else {
948:           /* otherwise relink the list */
949:           r->next = p->next;
950:         }
951:         xbee.pktcount--;
952:
953:         /* free this packet! */
954:         n = p->next;
955:         Xfree(p);
956:         /* move on */
957:         p = n;
958:       } else {
959:         /* move on */
960:         r = p;
961:         p = p->next;
962:       }
963:     } while (p);
964:     xbee.pktlast = r;
965:   }
966:
967:   /* unlock the packet mutex */
968:   xbee_mutex_unlock(xbee.pktmutex);
969: }
970:
971: /* ########################################################################
972:    xbee_endcon
973:    close the unwanted connection
974:    free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
975: void xbee_endcon2(xbee_con **con) {
976:   xbee_con *t, *u;
977:
978:   /* lock the connection mutex */
979:   xbee_mutex_lock(xbee.conmutex);
980:
981:   u = t = xbee.conlist;
982:   while (t && t != *con) {
983:     u = t;
984:     t = t->next;
985:   }
986:   if (!t) {
987:     /* invalid connection given... */
988:     if (xbee.log) {
989:       xbee_log("Attempted to close invalid connection...");
990:     }
991:     /* unlock the connection mutex */
992:     xbee_mutex_unlock(xbee.conmutex);
993:     return;
994:   }
995:   /* extract this connection from the list */
996:   u->next = (*con)->next;
997:   if (*con == xbee.conlist) xbee.conlist = NULL;
998:
999:   /* unlock the connection mutex */
1000:   xbee_mutex_unlock(xbee.conmutex);
1001:
1002:   /* remove all packets for this connection */
1003:   xbee_flushcon(*con);
1004:
1005:   /* free the connection! */
1006:   Xfree(*con);
1007: }
1008:
1009: /* ########################################################################
1010:    xbee_senddata
1011:    send the specified data to the provided connection */
1012: int xbee_senddata(xbee_con *con, char *format, ...) {
1013:   int ret;
1014:   va_list ap;
1015:
1016:   ISREADY;
1017:
1018:   /* xbee_vsenddata() wants a va_list... */
1019:   va_start(ap, format);
1020:   /* hand it over :) */
```

```
1021:   ret = xbee_vsenddata(con,format,ap);
1022:   va_end(ap);
1023:   return ret;
1024: }
1025:
1026: int xbee_vsenddata(xbee_con *con, char *format, va_list ap) {
1027:   unsigned char data[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
1028:   int length;
1029:
1030:   ISREADY;
1031:
1032:   /* make up the data and keep the length, its possible there are nulls in there */
1033:   length = vsnprintf((char *)data,128,format,ap);
1034:
1035:   /* hand it over :) */
1036:   return xbee_nsenddata(con,(char *)data,length);
1037: }
1038:
1039: int xbee_nsenddata(xbee_con *con, char *data, int length) {
1040:   t_data *pkt;
1041:   int i;
1042:   unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
1043:
1044:   ISREADY;
1045:
1046:   if (!con) return -1;
1047:   if (con->type == xbee_unknown) return -1;
1048:   if (length > 127) return -1;
1049:
1050:
1051:   if (xbee.log) {
1052:     xbee_log("--== TX Packet ============--");
1053:     xbee_logc("Connection Type: ");
1054:     switch (con->type) {
1055:     case xbee_unknown:       fprintf(xbee.log,"Unknown\n"); break;
1056:     case xbee_localAT:       fprintf(xbee.log,"Local AT\n"); break;
1057:     case xbee_remoteAT:      fprintf(xbee.log,"Remote AT\n"); break;
1058:     case xbee_16bitRemoteAT: fprintf(xbee.log,"Remote AT (16-bit)\n"); break;
1059:     case xbee_64bitRemoteAT: fprintf(xbee.log,"Remote AT (64-bit)\n"); break;
1060:     case xbee_16bitData:     fprintf(xbee.log,"Data (16-bit)\n"); break;
1061:     case xbee_64bitData:     fprintf(xbee.log,"Data (64-bit)\n"); break;
1062:     case xbee_16bitIO:       fprintf(xbee.log,"IO (16-bit)\n"); break;
1063:     case xbee_64bitIO:       fprintf(xbee.log,"IO (64-bit)\n"); break;
1064:     case xbee_txStatus:      fprintf(xbee.log,"Tx Status\n"); break;
1065:     case xbee_modemStatus:   fprintf(xbee.log,"Modem Status\n"); break;
1066:     }
1067:     xbee_logc("Destination: ");
1068:     for (i=0;i<(con->tAddr64?8:2);i++) {
1069:       fprintf(xbee.log,(i?":%02X":"%02X"),con->tAddr[i]);
1070:     }
1071:     fprintf(xbee.log,"\n");
1072:     xbee_log("Length: %d",length);
1073:     for (i=0;i<length;i++) {
1074:       xbee_logc("%3d | 0x%02X ",i,data[i]);
1075:       if ((data[i] > 32) && (data[i] < 127)) {
1076:         fprintf(xbee.log,"'%c'\n",data[i]);
1077:       } else{
1078:         fprintf(xbee.log," _\n");
1079:       }
1080:     }
1081:   }
1082:
1083:   /* ########################################## */
1084:   /* if: local AT */
1085:   if (con->type == xbee_localAT) {
1086:     /* AT commands are 2 chars long (plus optional parameter) */
1087:     if (length < 2) return -1;
1088:
1089:     /* use the command? */
1090:     buf[0] = ((!con->atQueue)?0x08:0x09);
1091:     buf[1] = con->frameID;
1092:
1093:     /* copy in the data */
1094:     for (i=0;i<length;i++) {
1095:       buf[i+2] = data[i];
1096:     }
1097:
1098:     /* setup the packet */
1099:     pkt = xbee_make_pkt(buf,i+2);
1100:     /* send it on */
1101:     xbee_send_pkt(pkt);
1102:
1103:     return 0;
1104:
1105:     /* ########################################## */
```

```
1106:     /* if: remote AT */
1107:   } else if ((con->type == xbee_16bitRemoteAT) ||
1108:              (con->type == xbee_64bitRemoteAT)) {
1109:     if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter) */
1110:     buf[0] = 0x17;
1111:     buf[1] = con->frameID;
1112:
1113:     /* copy in the relevant address */
1114:     if (con->tAddr64) {
1115:       memcpy(&buf[2],con->tAddr,8);
1116:       buf[10] = 0xFF;
1117:       buf[11] = 0xFE;
1118:     } else {
1119:       memset(&buf[2],0,8);
1120:       memcpy(&buf[10],con->tAddr,2);
1121:     }
1122:     /* queue the command? */
1123:     buf[12] = ((!con->atQueue)?0x02:0x00);
1124:
1125:     /* copy in the data */
1126:     for (i=0;i<length;i++) {
1127:       buf[i+13] = data[i];
1128:     }
1129:
1130:     /* setup the packet */
1131:     pkt = xbee_make_pkt(buf,i+13);
1132:     /* send it on */
1133:     xbee_send_pkt(pkt);
1134:
1135:     return 0;
1136:
1137:     /* ######################################### */
1138:     /* if: 16 or 64bit Data */
1139:   } else if ((con->type == xbee_16bitData) ||
1140:              (con->type == xbee_64bitData)) {
1141:     int offset;
1142:
1143:     /* if: 16bit Data */
1144:     if (con->type == xbee_16bitData) {
1145:       buf[0] = 0x01;
1146:       offset = 5;
1147:       /* copy in the address */
1148:       memcpy(&buf[2],con->tAddr,2);
1149:
1150:       /* if: 64bit Data */
1151:     } else { /* 64bit Data */
1152:       buf[0] = 0x00;
1153:       offset = 11;
1154:       /* copy in the address */
1155:       memcpy(&buf[2],con->tAddr,8);
1156:     }
1157:
1158:     /* copy frameID */
1159:     buf[1] = con->frameID;
1160:
1161:     /* disable ack? broadcast? */
1162:     buf[offset-1] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
1163:
1164:     /* copy in the data */
1165:     for (i=0;i<length;i++) {
1166:       buf[i+offset] = data[i];
1167:     }
1168:
1169:     /* setup the packet */
1170:     pkt = xbee_make_pkt(buf,i+offset);
1171:     /* send it on */
1172:     xbee_send_pkt(pkt);
1173:
1174:     return 0;
1175:
1176:     /* ######################################### */
1177:     /* if: I/O */
1178:   } else if ((con->type == xbee_64bitIO) ||
1179:              (con->type == xbee_16bitIO)) {
1180:     /* not currently implemented... is it even allowed? */
1181:     if (xbee.log) {
1182:       fprintf(xbee.log,"******* TODO ********\n");
1183:     }
1184:   }
1185:
1186:   return -2;
1187: }
1188:
1189: /* ################################################################### */
1190:    xbee_getpacket
```

```
1191:    retrieves the next packet destined for the given connection
1192:    once the packet has been retrieved, it is removed for the list! */
1193: xbee_pkt *xbee_getpacketwait(xbee_con *con) {
1194:   xbee_pkt *p;
1195:   int i;
1196:
1197:   /* 50ms * 20 = 1 second */
1198:   for (i = 0; i < 20; i++) {
1199:     p = xbee_getpacket(con);
1200:     if (p) break;
1201:     usleep(50000); /* 50ms */
1202:   }
1203:
1204:   return p;
1205: }
1206: xbee_pkt *xbee_getpacket(xbee_con *con) {
1207:   xbee_pkt *l, *p, *q;
1208:   /*if (xbee.log) {
1209:     xbee_log("--== Get Packet ==========--");
1210:     }*/
1211:
1212:   /* lock the packet mutex */
1213:   xbee_mutex_lock(xbee.pktmutex);
1214:
1215:   /* if: there are no packets */
1216:   if ((p = xbee.pktlist) == NULL) {
1217:     xbee_mutex_unlock(xbee.pktmutex);
1218:     /*if (xbee.log) {
1219:       xbee_log("No packets avaliable...");
1220:       }*/
1221:     return NULL;
1222:   }
1223:
1224:   l = NULL;
1225:   q = NULL;
1226:   /* get the first avaliable packet for this connection */
1227:   do {
1228:     /* does the packet match the connection? */
1229:     if (xbee_matchpktcon(p,con)) {
1230:       q = p;
1231:       break;
1232:     }
1233:     /* move on */
1234:     l = p;
1235:     p = p->next;
1236:   } while (p);
1237:
1238:   /* if: no packet was found */
1239:   if (!q) {
1240:     xbee_mutex_unlock(xbee.pktmutex);
1241:     /*if (xbee.log) {
1242:       xbee_log("No packets avaliable (for connection)...");
1243:       }*/
1244:     return NULL;
1245:   }
1246:
1247:   /* if it was the first packet */
1248:   if (l) {
1249:     /* relink the list */
1250:     l->next = p->next;
1251:     if (!l->next) xbee.pktlast = l;
1252:   } else {
1253:     /* move the chain along */
1254:     xbee.pktlist = p->next;
1255:     if (!xbee.pktlist) {
1256:       xbee.pktlast = NULL;
1257:     } else if (!xbee.pktlist->next) {
1258:       xbee.pktlast = xbee.pktlist;
1259:     }
1260:   }
1261:   xbee.pktcount--;
1262:
1263:   /* unlink this packet from the chain! */
1264:   q->next = NULL;
1265:
1266:   if (xbee.log) {
1267:     xbee_log("--== Get Packet ==========--");
1268:     xbee_log("Got a packet");
1269:     xbee_log("Packets left: %d",xbee.pktcount);
1270:   }
1271:
1272:   /* unlock the packet mutex */
1273:   xbee_mutex_unlock(xbee.pktmutex);
1274:
1275:   /* and return the packet (must be free'd by caller!) */
```

```
1276:   return q;
1277: }
1278:
1279: /* ###################################################################
1280:    xbee_matchpktcon - INTERNAL
1281:    checks if the packet matches the connection */
1282: static int xbee_matchpktcon(xbee_pkt *pkt, xbee_con *con) {
1283:   /* if: the connection type matches the packet type OR
1284:      the connection is 16/64bit remote AT, and the packet is a remote AT response */
1285:   if ((pkt->type == con->type) || /* -- */
1286:       ((pkt->type == xbee_remoteAT) && /* -- */
1287:        ((con->type == xbee_16bitRemoteAT) ||
1288:         (con->type == xbee_64bitRemoteAT)))) {
1289:
1290:     /* if: the packet is modem status OR
1291:        the packet is tx status or AT data and the frame IDs match OR
1292:        the addresses match */
1293:     if (pkt->type == xbee_modemStatus) return 1;
1294:
1295:     if ((pkt->type == xbee_txStatus) ||
1296:         (pkt->type == xbee_localAT) ||
1297:         (pkt->type == xbee_remoteAT)) {
1298:       if (pkt->frameID == con->frameID) {
1299:         return 1;
1300:       }
1301:     } else if (pkt->sAddr64 && !memcmp(pkt->Addr64,con->tAddr,8)) {
1302:       return 1;
1303:     } else if (!pkt->sAddr64 && !memcmp(pkt->Addr16,con->tAddr,2)) {
1304:       return 1;
1305:     }
1306:   }
1307:   return 0;
1308: }
1309:
1310: /* ###################################################################
1311:    xbee_parse_io - INTERNAL
1312:    parses the data given into the packet io information */
1313: static int xbee_parse_io(xbee_pkt *p, unsigned char *d, int maskOffset, int sampleOffset, int sample) {
1314:   xbee_sample *s = &(p->IOdata[sample]);
1315:
1316:   /* copy in the I/O data mask */
1317:   s->IOmask = (((d[maskOffset]<<8) | d[maskOffset + 1]) & 0x7FFF);
1318:
1319:   /* copy in the digital I/O data */
1320:   s->IOdigital = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x01FF);
1321:
1322:   /* advance over the digital data, if its there */
1323:   sampleOffset += ((s->IOmask & 0x01FF)?2:0);
1324:
1325:   /* copy in the analog I/O data */
1326:   if (s->IOmask & 0x0200) {
1327:     s->IOanalog[0] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1328:     sampleOffset+=2;
1329:   }
1330:   if (s->IOmask & 0x0400) {
1331:     s->IOanalog[1] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1332:     sampleOffset+=2;
1333:   }
1334:   if (s->IOmask & 0x0800) {
1335:     s->IOanalog[2] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1336:     sampleOffset+=2;
1337:   }
1338:   if (s->IOmask & 0x1000) {
1339:     s->IOanalog[3] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1340:     sampleOffset+=2;
1341:   }
1342:   if (s->IOmask & 0x2000) {
1343:     s->IOanalog[4] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1344:     sampleOffset+=2;
1345:   }
1346:   if (s->IOmask & 0x4000) {
1347:     s->IOanalog[5] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1348:     sampleOffset+=2;
1349:   }
1350:
1351:   if (xbee.log) {
1352:     if (s->IOmask & 0x0001)
1353:       xbee_log("Digital 0: %c",((s->IOdigital & 0x0001)?'1':'0'));
1354:     if (s->IOmask & 0x0002)
1355:       xbee_log("Digital 1: %c",((s->IOdigital & 0x0002)?'1':'0'));
1356:     if (s->IOmask & 0x0004)
1357:       xbee_log("Digital 2: %c",((s->IOdigital & 0x0004)?'1':'0'));
1358:     if (s->IOmask & 0x0008)
1359:       xbee_log("Digital 3: %c",((s->IOdigital & 0x0008)?'1':'0'));
1360:     if (s->IOmask & 0x0010)
```

```
1361:       xbee_log("Digital 4: %c",((s->IOdigital & 0x0010)?'1':'0'));
1362:     if (s->IOmask & 0x0020)
1363:       xbee_log("Digital 5: %c",((s->IOdigital & 0x0020)?'1':'0'));
1364:     if (s->IOmask & 0x0040)
1365:       xbee_log("Digital 6: %c",((s->IOdigital & 0x0040)?'1':'0'));
1366:     if (s->IOmask & 0x0080)
1367:       xbee_log("Digital 7: %c",((s->IOdigital & 0x0080)?'1':'0'));
1368:     if (s->IOmask & 0x0100)
1369:       xbee_log("Digital 8: %c",((s->IOdigital & 0x0100)?'1':'0'));
1370:     if (s->IOmask & 0x0200)
1371:       xbee_log("Analog  0: %d (~%.2fv)\n",s->IOanalog[0],(3.3/1023)*s->IOanalog[0]);
1372:     if (s->IOmask & 0x0400)
1373:       xbee_log("Analog  1: %d (~%.2fv)\n",s->IOanalog[1],(3.3/1023)*s->IOanalog[1]);
1374:     if (s->IOmask & 0x0800)
1375:       xbee_log("Analog  2: %d (~%.2fv)\n",s->IOanalog[2],(3.3/1023)*s->IOanalog[2]);
1376:     if (s->IOmask & 0x1000)
1377:       xbee_log("Analog  3: %d (~%.2fv)\n",s->IOanalog[3],(3.3/1023)*s->IOanalog[3]);
1378:     if (s->IOmask & 0x2000)
1379:       xbee_log("Analog  4: %d (~%.2fv)\n",s->IOanalog[4],(3.3/1023)*s->IOanalog[4]);
1380:     if (s->IOmask & 0x4000)
1381:       xbee_log("Analog  5: %d (~%.2fv)\n",s->IOanalog[5],(3.3/1023)*s->IOanalog[5]);
1382:   }
1383:
1384:   return sampleOffset;
1385: }
1386:
1387: /* ####################################################################
1388:    xbee_listen_stop
1389:    stops the listen thread after the current packet has been processed */
1390: void xbee_listen_stop(void) {
1391:   xbee.listenrun = 0;
1392: }
1393:
1394: /* ####################################################################
1395:    xbee_listen_wrapper - INTERNAL
1396:    the xbee_listen wrapper. Prints an error when xbee_listen ends */
1397: static void xbee_listen_wrapper(t_info *info) {
1398:   int ret;
1399:   /* just falls out if the proper 'go-ahead' isn't given */
1400:   if (xbee_ready != -1) return;
1401:   /* now allow the parent to continue */
1402:   xbee_ready = -2;
1403:
1404: #ifdef _WIN32
1405:   /* win32 requires this delay... no idea why */
1406:   usleep(1000000);
1407: #endif
1408:
1409:   while (xbee.listenrun) {
1410:     info->i = -1;
1411:     ret = xbee_listen(info);
1412:     if (!xbee.listenrun) break;
1413:     if (xbee.log) {
1414:       xbee_log("xbee_listen() returned [%d]... Restarting in 250ms!",ret);
1415:     }
1416:     usleep(25000);
1417:   }
1418: }
1419:
1420: /* xbee_listen - INTERNAL
1421:    the xbee xbee_listen thread
1422:    reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
1423: static int xbee_listen(t_info *info) {
1424:   unsigned char c, t, d[1024];
1425:   unsigned int l, i, chksum, o;
1426:   int j;
1427:   xbee_pkt *p, *q;
1428:   xbee_con *con;
1429:   int hasCon;
1430:
1431:   /* just falls out if the proper 'go-ahead' isn't given */
1432:   if (info->i != -1) return -1;
1433:   /* do this forever :) */
1434:   while (xbee.listenrun) {
1435:     /* wait for a valid start byte */
1436:     if (xbee_getrawbyte() != 0x7E) continue;
1437:     if (!xbee.listenrun) return 0;
1438:
1439:     if (xbee.log) {
1440:       xbee_log("--== RX Packet ==========--");
1441:       xbee_log("Got a packet!...");
1442:     }
1443:
1444:     /* get the length */
1445:     l = xbee_getbyte() << 8;
```

```
1446:     l += xbee_getbyte();
1447:
1448:     /* check it is a valid length... */
1449:     if (!l) {
1450:       if (xbee.log) {
1451:         xbee_log("Recived zero length packet!");
1452:       }
1453:       continue;
1454:     }
1455:     if (l > 100) {
1456:       if (xbee.log) {
1457:         xbee_log("Recived oversized packet! Length: %d",l - 1);
1458:       }
1459:     }
1460:     if (l > sizeof(d) - 1) {
1461:       if (xbee.log) {
1462:         xbee_log("Recived packet larger than buffer! Discarding...");
1463:       }
1464:       continue;
1465:     }
1466:
1467:     if (xbee.log) {
1468:       xbee_log("Length: %d",l - 1);
1469:     }
1470:
1471:     /* get the packet type */
1472:     t = xbee_getbyte();
1473:
1474:     /* start the checksum */
1475:     chksum = t;
1476:
1477:     /* suck in all the data */
1478:     for (i = 0; l > 1 && i < 128; l--, i++) {
1479:       /* get an unescaped byte */
1480:       c = xbee_getbyte();
1481:       d[i] = c;
1482:       chksum += c;
1483:       if (xbee.log) {
1484:         xbee_logc("%3d | 0x%02X | ",i,c);
1485:         if ((c > 32) && (c < 127)) fprintf(xbee.log,"'%c'",c); else fprintf(xbee.log," _ ");
1486:
1487:         if ((t == 0x80 && i == (8 + 2)) || /* 64-bit Data packet */
1488:             (t == 0x81 && i == (2 + 2))) { /* 16-bit Data packet */
1489:           /* mark the beginning of the 'data' bytes */
1490:           fprintf(xbee.log,"   <-- data starts");
1491:         }
1492:
1493:         fprintf(xbee.log,"\n");
1494:       }
1495:     }
1496:     i--; /* it went up too many times!... */
1497:
1498:     /* add the checksum */
1499:     chksum += xbee_getbyte();
1500:
1501:     /* check if the whole packet was recieved, or something else occured... unlikely... */
1502:     if (l>1) {
1503:       if (xbee.log) {
1504:         xbee_log("Didn't get whole packet... :(");
1505:       }
1506:       continue;
1507:     }
1508:
1509:     /* check the checksum */
1510:     if ((chksum & 0xFF) != 0xFF) {
1511:       if (xbee.log) {
1512:         xbee_log("Invalid Checksum: 0x%02X",chksum);
1513:       }
1514:       continue;
1515:     }
1516:
1517:     /* make a new packet */
1518:     p = Xcalloc(sizeof(xbee_pkt));
1519:     q = NULL;
1520:     p->datalen = 0;
1521:
1522:     /* ######################################### */
1523:     /* if: modem status */
1524:     if (t == 0x8A) {
1525:       if (xbee.log) {
1526:         xbee_log("Packet type: Modem Status (0x8A)");
1527:         xbee_logc("Event: ");
1528:         switch (d[0]) {
1529:         case 0x00: fprintf(xbee.log,"Hardware reset"); break;
1530:         case 0x01: fprintf(xbee.log,"Watchdog timer reset"); break;
```

```
1531:         case 0x02: fprintf(xbee.log,"Associated"); break;
1532:         case 0x03: fprintf(xbee.log,"Disassociated"); break;
1533:         case 0x04: fprintf(xbee.log,"Synchronization lost"); break;
1534:         case 0x05: fprintf(xbee.log,"Coordinator realignment"); break;
1535:         case 0x06: fprintf(xbee.log,"Coordinator started"); break;
1536:         }
1537:         fprintf(xbee.log,"... (0x%02X)\n",d[0]);
1538:       }
1539:       p->type = xbee_modemStatus;
1540:
1541:       p->sAddr64 = FALSE;
1542:       p->dataPkt = FALSE;
1543:       p->txStatusPkt = FALSE;
1544:       p->modemStatusPkt = TRUE;
1545:       p->remoteATPkt = FALSE;
1546:       p->IOPkt = FALSE;
1547:
1548:       /* modem status can only ever give 1 'data' byte */
1549:       p->datalen = 1;
1550:       p->data[0] = d[0];
1551:
1552:       /* ######################################### */
1553:       /* if: local AT response */
1554:     } else if (t == 0x88) {
1555:       if (xbee.log) {
1556:         xbee_log("Packet type: Local AT Response (0x88)");
1557:         xbee_log("FrameID: 0x%02X",d[0]);
1558:         xbee_log("AT Command: %c%c",d[1],d[2]);
1559:         xbee_logc("Status: ");
1560:         if (d[3] == 0) fprintf(xbee.log,"OK");
1561:         else if (d[3] == 1) fprintf(xbee.log,"Error");
1562:         else if (d[3] == 2) fprintf(xbee.log,"Invalid Command");
1563:         else if (d[3] == 3) fprintf(xbee.log,"Invalid Parameter");
1564:         fprintf(xbee.log," (0x%02X)\n",d[3]);
1565:       }
1566:       p->type = xbee_localAT;
1567:
1568:       p->sAddr64 = FALSE;
1569:       p->dataPkt = FALSE;
1570:       p->txStatusPkt = FALSE;
1571:       p->modemStatusPkt = FALSE;
1572:       p->remoteATPkt = FALSE;
1573:       p->IOPkt = FALSE;
1574:
1575:       p->frameID = d[0];
1576:       p->atCmd[0] = d[1];
1577:       p->atCmd[1] = d[2];
1578:
1579:       p->status = d[3];
1580:
1581:       /* copy in the data */
1582:       p->datalen = i-3;
1583:       for (;i>3;i--) p->data[i-4] = d[i];
1584:
1585:       /* ######################################### */
1586:       /* if: remote AT response */
1587:     } else if (t == 0x97) {
1588:       if (xbee.log) {
1589:         xbee_log("Packet type: Remote AT Response (0x97)");
1590:         xbee_log("FrameID: 0x%02X",d[0]);
1591:         xbee_logc("64-bit Address: ");
1592:         for (j=0;j<8;j++) {
1593:           fprintf(xbee.log,(j?":%02X":"%02X"),d[1+j]);
1594:         }
1595:         fprintf(xbee.log,"\n");
1596:         xbee_logc("16-bit Address: ");
1597:         for (j=0;j<2;j++) {
1598:           fprintf(xbee.log,(j?":%02X":"%02X"),d[9+j]);
1599:         }
1600:         fprintf(xbee.log,"\n");
1601:         xbee_log("AT Command: %c%c",d[11],d[12]);
1602:         xbee_logc("Status: ");
1603:         if (d[13] == 0) fprintf(xbee.log,"OK");
1604:         else if (d[13] == 1) fprintf(xbee.log,"Error");
1605:         else if (d[13] == 2) fprintf(xbee.log,"Invalid Command");
1606:         else if (d[13] == 3) fprintf(xbee.log,"Invalid Parameter");
1607:         else if (d[13] == 4) fprintf(xbee.log,"No Response");
1608:         fprintf(xbee.log," (0x%02X)\n",d[13]);
1609:       }
1610:       p->type = xbee_remoteAT;
1611:
1612:       p->sAddr64 = FALSE;
1613:       p->dataPkt = FALSE;
1614:       p->txStatusPkt = FALSE;
1615:       p->modemStatusPkt = FALSE;
```

```
1616:       p->remoteATPkt = TRUE;
1617:       p->IOPkt = FALSE;
1618:
1619:       p->frameID = d[0];
1620:
1621:       p->Addr64[0] = d[1];
1622:       p->Addr64[1] = d[2];
1623:       p->Addr64[2] = d[3];
1624:       p->Addr64[3] = d[4];
1625:       p->Addr64[4] = d[5];
1626:       p->Addr64[5] = d[6];
1627:       p->Addr64[6] = d[7];
1628:       p->Addr64[7] = d[8];
1629:
1630:       p->Addr16[0] = d[9];
1631:       p->Addr16[1] = d[10];
1632:
1633:       p->atCmd[0] = d[11];
1634:       p->atCmd[1] = d[12];
1635:
1636:       p->status = d[13];
1637:
1638:       p->samples = 1;
1639:
1640:       if (p->status == 0x00 && p->atCmd[0] == 'I' && p->atCmd[1] == 'S') {
1641:         /* parse the io data */
1642:         if (xbee.log) xbee_log("--- Sample -----------------");
1643:         xbee_parse_io(p, d, 15, 17, 0);
1644:         if (xbee.log) xbee_log("----------------------------");
1645:       } else {
1646:         /* copy in the data */
1647:         p->datalen = i-13;
1648:         for (;i>13;i--) p->data[i-14] = d[i];
1649:       }
1650:
1651:       /* ############################################ */
1652:       /* if: TX status */
1653:     } else if (t == 0x89) {
1654:       if (xbee.log) {
1655:         xbee_log("Packet type: TX Status Report (0x89)");
1656:         xbee_log("FrameID: 0x%02X",d[0]);
1657:         xbee_logc("Status: ");
1658:         if (d[1] == 0) fprintf(xbee.log,"Success");
1659:         else if (d[1] == 1) fprintf(xbee.log,"No ACK");
1660:         else if (d[1] == 2) fprintf(xbee.log,"CCA Failure");
1661:         else if (d[1] == 3) fprintf(xbee.log,"Purged");
1662:         fprintf(xbee.log," (0x%02X)\n",d[1]);
1663:       }
1664:       p->type = xbee_txStatus;
1665:
1666:       p->sAddr64 = FALSE;
1667:       p->dataPkt = FALSE;
1668:       p->txStatusPkt = TRUE;
1669:       p->modemStatusPkt = FALSE;
1670:       p->remoteATPkt = FALSE;
1671:       p->IOPkt = FALSE;
1672:
1673:       p->frameID = d[0];
1674:
1675:       p->status = d[1];
1676:
1677:       /* never returns data */
1678:       p->datalen = 0;
1679:
1680:       /* ############################################ */
1681:       /* if: 16 / 64bit data recieve */
1682:     } else if ((t == 0x80) ||
1683:                (t == 0x81)) {
1684:       int offset;
1685:       if (t == 0x80) { /* 64bit */
1686:         offset = 8;
1687:       } else { /* 16bit */
1688:         offset = 2;
1689:       }
1690:       if (xbee.log) {
1691:         xbee_log("Packet type: %d-bit RX Data (0x%02X)",((t == 0x80)?64:16),t);
1692:         xbee_logc("%d-bit Address: ",((t == 0x80)?64:16));
1693:         for (j=0;j<offset;j++) {
1694:           fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
1695:         }
1696:         fprintf(xbee.log,"\n");
1697:         xbee_log("RSSI: -%ddB",d[offset]);
1698:         if (d[offset + 1] & 0x02) xbee_log("Options: Address Broadcast");
1699:         if (d[offset + 1] & 0x03) xbee_log("Options: PAN Broadcast");
1700:       }
```

```
1701:       p->dataPkt = TRUE;
1702:       p->txStatusPkt = FALSE;
1703:       p->modemStatusPkt = FALSE;
1704:       p->remoteATPkt = FALSE;
1705:       p->IOPkt = FALSE;
1706:
1707:       if (t == 0x80) { /* 64bit */
1708:         p->type = xbee_64bitData;
1709:
1710:         p->sAddr64 = TRUE;
1711:
1712:         p->Addr64[0] = d[0];
1713:         p->Addr64[1] = d[1];
1714:         p->Addr64[2] = d[2];
1715:         p->Addr64[3] = d[3];
1716:         p->Addr64[4] = d[4];
1717:         p->Addr64[5] = d[5];
1718:         p->Addr64[6] = d[6];
1719:         p->Addr64[7] = d[7];
1720:       } else { /* 16bit */
1721:         p->type = xbee_16bitData;
1722:
1723:         p->sAddr64 = FALSE;
1724:
1725:         p->Addr16[0] = d[0];
1726:         p->Addr16[1] = d[1];
1727:       }
1728:
1729:       /* save the RSSI / signal strength
1730:          this can be used with printf as:
1731:          printf("-%ddB\n",p->RSSI); */
1732:       p->RSSI = d[offset];
1733:
1734:       p->status = d[offset + 1];
1735:
1736:       /* copy in the data */
1737:       p->datalen = i-(offset + 1);
1738:       for (;i>offset + 1;i--) p->data[i-(offset + 2)] = d[i];
1739:
1740:       /* ############################################ */
1741:       /* if: 16 / 64bit I/O recieve */
1742:     } else if ((t == 0x82) ||
1743:                (t == 0x83)) {
1744:       int offset;
1745:       if (t == 0x82) { /* 64bit */
1746:         p->type = xbee_64bitIO;
1747:
1748:         p->sAddr64 = TRUE;
1749:
1750:         p->Addr64[0] = d[0];
1751:         p->Addr64[1] = d[1];
1752:         p->Addr64[2] = d[2];
1753:         p->Addr64[3] = d[3];
1754:         p->Addr64[4] = d[4];
1755:         p->Addr64[5] = d[5];
1756:         p->Addr64[6] = d[6];
1757:         p->Addr64[7] = d[7];
1758:
1759:         offset = 8;
1760:         p->samples = d[10];
1761:       } else { /* 16bit */
1762:         p->type = xbee_16bitIO;
1763:
1764:         p->sAddr64 = FALSE;
1765:
1766:         p->Addr16[0] = d[0];
1767:         p->Addr16[1] = d[1];
1768:
1769:         offset = 2;
1770:         p->samples = d[4];
1771:       }
1772:       if (p->samples > 1) {
1773:         p = Xrealloc(p, sizeof(xbee_pkt) + (sizeof(xbee_sample) * (p->samples - 1)));
1774:       }
1775:       if (xbee.log) {
1776:         xbee_logc("Packet type: %d-bit RX I/O Data (0x%02X)\n",((t == 0x82)?64:16),t);
1777:         xbee_logc("%d-bit Address: ",((t == 0x82)?64:16));
1778:         for (j = 0; j < offset; j++) {
1779:           fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
1780:         }
1781:         fprintf(xbee.log,"\n");
1782:         xbee_log("RSSI: -%ddB",d[offset]);
1783:         if (d[9] & 0x02) xbee_log("Options: Address Broadcast");
1784:         if (d[9] & 0x02) xbee_log("Options: PAN Broadcast");
1785:         xbee_log("Samples: %d",d[offset + 2]);
```

```
1786:       }
1787:       i = offset + 5;
1788:
1789:       /* never returns data */
1790:       p->datalen = 0;
1791:
1792:       p->dataPkt = FALSE;
1793:       p->txStatusPkt = FALSE;
1794:       p->modemStatusPkt = FALSE;
1795:       p->remoteATPkt = FALSE;
1796:       p->IOPkt = TRUE;
1797:
1798:       /* save the RSSI / signal strength
1799:          this can be used with printf as:
1800:          printf("-%ddB\n",p->RSSI); */
1801:       p->RSSI = d[offset];
1802:
1803:       p->status = d[offset + 1];
1804:
1805:       /* each sample is split into its own packet here, for simplicity */
1806:       for (o = 0; o < p->samples; o++) {
1807:         if (xbee.log) {
1808:           xbee_log("--- Sample %3d -------------", o);
1809:         }
1810:
1811:         /* parse the io data */
1812:         i = xbee_parse_io(p, d, offset + 3, i, o);
1813:       }
1814:       if (xbee.log) {
1815:         xbee_log("----------------------------");
1816:       }
1817:
1818:       /* ########################################## */
1819:       /* if: Unknown */
1820:     } else {
1821:       if (xbee.log) {
1822:         xbee_log("Packet type: Unknown (0x%02X)",t);
1823:       }
1824:       p->type = xbee_unknown;
1825:     }
1826:     p->next = NULL;
1827:
1828:     /* lock the connection mutex */
1829:     xbee_mutex_lock(xbee.conmutex);
1830:
1831:     con = xbee.conlist;
1832:     hasCon = 0;
1833:     while (con) {
1834:       if (xbee_matchpktcon(p,con)) {
1835:         hasCon = 1;
1836:         break;
1837:       }
1838:       con = con->next;
1839:     }
1840:
1841:     /* unlock the connection mutex */
1842:     xbee_mutex_unlock(xbee.conmutex);
1843:
1844:     /* if the packet doesn't have a connection, don't add it! */
1845:     if (!hasCon) {
1846:       Xfree(p);
1847:       if (xbee.log) {
1848:         xbee_log("Connectionless packet... discarding!");
1849:       }
1850:       continue;
1851:     }
1852:
1853:     /* lock the packet mutex, so we can safely add the packet to the list */
1854:     xbee_mutex_lock(xbee.pktmutex);
1855:
1856:     /* if: the list is empty */
1857:     if (!xbee.pktlist) {
1858:       /* start the list! */
1859:       xbee.pktlist = p;
1860:     } else if (xbee.pktlast) {
1861:       /* add the packet to the end */
1862:       xbee.pktlast->next = p;
1863:     } else {
1864:       /* pktlast wasnt set... look for the end and then set it */
1865:       i = 0;
1866:       q = xbee.pktlist;
1867:       while (q->next) {
1868:         q = q->next;
1869:         i++;
1870:       }
```

```
1871:       q->next = p;
1872:       xbee.pktcount = i;
1873:     }
1874:     xbee.pktlast = p;
1875:     xbee.pktcount++;
1876:
1877:     /* unlock the packet mutex */
1878:     xbee_mutex_unlock(xbee.pktmutex);
1879:
1880:     if (xbee.log) {
1881:       xbee_log("--========================--");
1882:       xbee_log("Packets: %d",xbee.pktcount);
1883:     }
1884:
1885:     p = q = NULL;
1886:   }
1887:   return 0;
1888: }
1889:
1890: /* ################################################################
1891:    xbee_getbyte - INTERNAL
1892:    waits for an escaped byte of data */
1893: static unsigned char xbee_getbyte(void) {
1894:   unsigned char c;
1895:
1896:   ISREADY;
1897:
1898:   /* take a byte */
1899:   c = xbee_getrawbyte();
1900:   /* if its escaped, take another and un-escape */
1901:   if (c == 0x7D) c = xbee_getrawbyte() ^ 0x20;
1902:
1903:   return (c & 0xFF);
1904: }
1905:
1906: /* ################################################################
1907:    xbee_getrawbyte - INTERNAL
1908:    waits for a raw byte of data */
1909: static unsigned char xbee_getrawbyte(void) {
1910:   struct timeval to;
1911:   int ret;
1912:   unsigned char c = 0x00;
1913:
1914:   ISREADY;
1915:
1916:   /* the loop is just incase there actually isnt a byte there to be read... */
1917:   do {
1918:     /* wait for a read to be possible */
1919:     /* timeout every 1 second to keep alive */
1920:     memset(&to, 0, sizeof(to));
1921:     to.tv_usec = 1000 * 1000;
1922:     if ((ret = xbee_select(&to)) == -1) {
1923:       perror("libxbee:xbee_getrawbyte()");
1924:       exit(1);
1925:     }
1926:     if (!xbee.listenrun) break;
1927:     if (ret == 0) continue;
1928:
1929:     /* read 1 character */
1930:     xbee_read(&c,1);
1931: #ifdef _WIN32
1932:     ret = xbee.ttyr;
1933:     if (ret == 0) {
1934:       usleep(10);
1935:       continue;
1936:     }
1937: #endif
1938:   } while (0);
1939:
1940:   return (c & 0xFF);
1941: }
1942:
1943: /* ################################################################
1944:    xbee_send_pkt - INTERNAL
1945:    sends a complete packet of data */
1946: static void xbee_send_pkt(t_data *pkt) {
1947:   ISREADY;
1948:
1949:   /* lock the send mutex */
1950:   xbee_mutex_lock(xbee.sendmutex);
1951:
1952:   /* write and flush the data */
1953:   xbee_write(pkt->data,pkt->length);
1954:
1955:   /* unlock the mutex */
```

```
1956:   xbee_mutex_unlock(xbee.sendmutex);
1957:
1958:   if (xbee.log) {
1959:     int i,x,y;
1960:     /* prints packet in hex byte-by-byte */
1961:     xbee_logc("TX Packet:");
1962:     for (i=0,x=0,y=0;i<pkt->length;i++,x--) {
1963:       if (x == 0) {
1964:         fprintf(xbee.log,"\n  0x%04X | ",y);
1965:         x = 0x8;
1966:         y += x;
1967:       }
1968:       if (x == 4) {
1969:         fprintf(xbee.log,"  ");
1970:       }
1971:       fprintf(xbee.log,"0x%02X ",pkt->data[i]);
1972:     }
1973:     fprintf(xbee.log,"\n");
1974:   }
1975:
1976:   /* free the packet */
1977:   Xfree(pkt);
1978: }
1979:
1980: /* ###################################################################
1981:    xbee_make_pkt - INTERNAL
1982:    adds delimiter field
1983:    calculates length and checksum
1984:    escapes bytes */
1985: static t_data *xbee_make_pkt(unsigned char *data, int length) {
1986:   t_data *pkt;
1987:   unsigned int l, i, o, t, x, m;
1988:   char d = 0;
1989:
1990:   ISREADY;
1991:
1992:   /* check the data given isnt too long
1993:      100 bytes maximum payload + 12 bytes header information */
1994:   if (length > 100 + 12) return NULL;
1995:
1996:   /* calculate the length of the whole packet
1997:      start, length (MSB), length (LSB), DATA, checksum */
1998:   l = 3 + length + 1;
1999:
2000:   /* prepare memory */
2001:   pkt = Xcalloc(sizeof(t_data));
2002:
2003:   /* put start byte on */
2004:   pkt->data[0] = 0x7E;
2005:
2006:   /* copy data into packet */
2007:   for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
2008:     /* if: its time for the checksum */
2009:     if (i == length) d = M8((0xFF - M8(t)));
2010:     /* if: its time for the high length byte */
2011:     else if (m == 1) d = M8(length >> 8);
2012:     /* if: its time for the low length byte */
2013:     else if (m == 2) d = M8(length);
2014:     /* if: its time for the normal data */
2015:     else if (m > 2) d = data[i];
2016:
2017:     x = 0;
2018:     /* check for any escapes needed */
2019:     if ((d == 0x11) || /* XON */
2020:         (d == 0x13) || /* XOFF */
2021:         (d == 0x7D) || /* Escape */
2022:         (d == 0x7E)) { /* Frame Delimiter */
2023:       l++;
2024:       pkt->data[o++] = 0x7D;
2025:       x = 1;
2026:     }
2027:
2028:     /* move data in */
2029:     pkt->data[o] = ((!x)?d:d^0x20);
2030:     if (m > 2) {
2031:       i++;
2032:       t += d;
2033:     }
2034:   }
2035:
2036:   /* remember the length */
2037:   pkt->length = l;
2038:
2039:   return pkt;
2040: }
```