

```
1:  /*
2:      libxbee - a C library to aid the use of Digi's Series 1 XBee modules
3:      running in API mode (AP=2).
4:
5:      Copyright (C) 2009 Attie Grande (attie@attie.co.uk)
6:
7:      This program is free software: you can redistribute it and/or modify
8:      it under the terms of the GNU General Public License as published by
9:      the Free Software Foundation, either version 3 of the License, or
10:     (at your option) any later version.
11:
12:     This program is distributed in the hope that it will be useful,
13:     but WITHOUT ANY WARRANTY; without even the implied warranty of
14:     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
15:     GNU General Public License for more details.
16:
17:     You should have received a copy of the GNU General Public License
18:     along with this program. If not, see <http://www.gnu.org/licenses/>.
19: */
20:
21: #include "globals.h"
22: #include "api.h"
23:
24: /* ready flag.
25:     needs to be set to -1 so that the listen thread can begin.
26:     then 1 so that functions can be used (after setup of course...) */
27: int xbee_ready = 0;
28:
29: /* #####
30: /* ### Memory Handling #####
31: /* #####
32:
33: /* malloc wrapper function */
34: void *Xmalloc(size_t size) {
35:     void *t;
36:     t = malloc(size);
37:     if (!t) {
38:         /* uhoh... thats pretty bad... */
39:         perror("xbee:malloc()");
40:         exit(1);
41:     }
42:     return t;
43: }
44:
45: /* calloc wrapper function */
46: void *Xcalloc(size_t size) {
47:     void *t;
48:     t = calloc(1, size);
49:     if (!t) {
50:         /* uhoh... thats pretty bad... */
51:         perror("xbee:calloc()");
52:         exit(1);
53:     }
54:     return t;
55: }
56:
57: /* realloc wrapper function */
58: void *Xrealloc(void *ptr, size_t size) {
59:     void *t;
60:     t = realloc(ptr, size);
61:     if (!t) {
62:         /* uhoh... thats pretty bad... */
63:         perror("xbee:realloc()");
64:         exit(1);
65:     }
66:     return t;
67: }
68:
69: /* free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
70: void Xfree2(void **ptr) {
71:     free(*ptr);
72:     *ptr = NULL;
73: }
74:
75: /* #####
76: /* ### XBee Functions #####
77: /* #####
78:
79: /* #####
80:     xbee_setup
81:     opens xbee serial port & creates xbee read thread
82:     the xbee must be configured for API mode 2
83:     THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
84: int xbee_setup(char *path, int baudrate) {
85:     t_info info;
```

```

86: struct flock fl;
87: struct termios tc;
88: speed_t chosenbaud;
89:
90: /* select the baud rate */
91: switch (baudrate) {
92:     case 1200: chosenbaud = B1200; break;
93:     case 2400: chosenbaud = B2400; break;
94:     case 4800: chosenbaud = B4800; break;
95:     case 9600: chosenbaud = B9600; break;
96:     case 19200: chosenbaud = B19200; break;
97:     case 38400: chosenbaud = B38400; break;
98:     case 57600: chosenbaud = B57600; break;
99:     case 115200: chosenbaud = B115200; break;
100: default:
101:     fprintf(stderr, "XBee: Unknown or incompatible baud rate specified... (%d)\n", baudrate);
102:     return -1;
103: };
104:
105: /* setup the connection mutex */
106: xbee.conlist = NULL;
107: if (pthread_mutex_init(&xbee.conmutex, NULL)) {
108:     perror("xbee_setup():pthread_mutex_init(conmutex)");
109:     return -1;
110: }
111:
112: /* setup the packet mutex */
113: xbee.pktlist = NULL;
114: if (pthread_mutex_init(&xbee.pktmutex, NULL)) {
115:     perror("xbee_setup():pthread_mutex_init(pktmutex)");
116:     return -1;
117: }
118:
119: /* setup the send mutex */
120: if (pthread_mutex_init(&xbee.sendmutex, NULL)) {
121:     perror("xbee_setup():pthread_mutex_init(sendmutex)");
122:     return -1;
123: }
124:
125: /* take a copy of the XBee device path */
126: if ((xbee.path = malloc(sizeof(char) * (strlen(path) + 1))) == NULL) {
127:     perror("xbee_setup():malloc(path)");
128:     return -1;
129: }
130: strcpy(xbee.path, path);
131:
132: /* open the serial port as a file descriptor */
133: if ((xbee.ttyfd = open(path, O_RDWR | O_NOCTTY | O_NONBLOCK)) == -1) {
134:     perror("xbee_setup():open()");
135:     Xfree(xbee.path);
136:     xbee.ttyfd = -1;
137:     xbee.tty = NULL;
138:     return -1;
139: }
140:
141: /* lock the file */
142: fl.l_type = F_WRLCK | F_RDLCK;
143: fl.l_whence = SEEK_SET;
144: fl.l_start = 0;
145: fl.l_len = 0;
146: fl.l_pid = getpid();
147: if (fcntl(xbee.ttyfd, F_SETLK, &fl) == -1) {
148:     perror("xbee_setup():fcntl()");
149:     Xfree(xbee.path);
150:     close(xbee.ttyfd);
151:     xbee.ttyfd = -1;
152:     xbee.tty = NULL;
153:     return -1;
154: }
155:
156: /* setup the baud rate and other io attributes */
157: tcgetattr(xbee.ttyfd, &tc);
158: cfsetispeed(&tc, chosenbaud); /* set input baud rate */
159: cfsetospeed(&tc, chosenbaud); /* set output baud rate */
160: /* input flags */
161: tc.c_iflag |= IGNBRK; /* enable ignoring break */
162: tc.c_iflag &= ~(IGNPAR | PARMRK); /* disable parity checks */
163: tc.c_iflag &= ~INPCK; /* disable parity checking */
164: tc.c_iflag &= ~ISTRIP; /* disable stripping 8th bit */
165: tc.c_iflag &= ~(INLCR | ICRNL); /* disable translating NL <-> CR */
166: tc.c_iflag &= ~IGNCR; /* disable ignoring CR */
167: tc.c_iflag &= ~(IXON | IXOFF); /* disable XON/XOFF flow control */
168: /* output flags */
169: tc.c_oflag &= ~OPOST; /* disable output processing */
170:

```

```

171: tc.c_oflag &= ~(ONLCR | OCRNL); /* disable translating NL <-> CR */
172: tc.c_oflag &= ~OFILL; /* disable fill characters */
173: /* control flags */
174: tc.c_cflag |= CREAD; /* enable reciever */
175: tc.c_cflag &= ~PARENB; /* disable parity */
176: tc.c_cflag &= ~CSTOPB; /* disable 2 stop bits */
177: tc.c_cflag &= ~CSIZE; /* remove size flag... */
178: tc.c_cflag |= CS8; /* ...enable 8 bit characters */
179: tc.c_cflag |= HUPCL; /* enable lower control lines on close - hang up */
180: /* local flags */
181: tc.c_lflag &= ~ISIG; /* disable generating signals */
182: tc.c_lflag &= ~ICANON; /* disable canonical mode - line by line */
183: tc.c_lflag &= ~ECHO; /* disable echoing characters */
184: tc.c_lflag &= ~NOFLSH; /* disable flushing on SIGINT */
185: tc.c_lflag &= ~IEXTEN; /* disable input processing */
186: tcsetattr(xbee.ttyfd, TCSANOW, &tc);
187:
188: /* open the serial port as a FILE */
189: if ((xbee.tty = fdopen(xbee.ttyfd, "r+")) == NULL) {
190:     perror("xbee_setup():fdopen()");
191:     Xfree(xbee.path);
192:     close(xbee.ttyfd);
193:     xbee.ttyfd = -1;
194:     xbee.tty = NULL;
195:     return -1;
196: }
197:
198: /* flush the serial port */
199: fflush(xbee.tty);
200:
201: /* allow the listen thread to start */
202: xbee_ready = -1;
203:
204: /* can start xbee_listen thread now */
205: if (pthread_create(&xbee.listent, NULL, (void (*)(void *))xbee_listen, (void *)&info) != 0) {
206:     perror("xbee_setup():pthread_create()");
207:     Xfree(xbee.path);
208:     fclose(xbee.tty);
209:     close(xbee.ttyfd);
210:     xbee.ttyfd = -1;
211:     xbee.tty = NULL;
212:     return -1;
213: }
214:
215: /* allow other functions to be used! */
216: xbee_ready = 1;
217:
218: /* make a txStatus connection */
219: xbee.con_txStatus = xbee_newcon("", xbee_txStatus);
220:
221: return 0;
222: }
223:
224: /* #####
225: xbee_con
226: produces a connection to the specified device and frameID
227: if a connection had already been made, then this connection will be returned */
228: xbee_con *xbee_newcon(unsigned char frameID, xbee_types type, ...) {
229:     xbee_con *con, *ocon;
230:     unsigned char tAddr[8];
231:     va_list ap;
232:     int t;
233: #ifdef DEBUG
234:     int i;
235: #endif
236:
237:     ISREADY;
238:
239:     if (!type || type == xbee_unknown) type = xbee_localAT; /* default to local AT */
240:     else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */
241:
242:     va_start(ap, type);
243:     /* if: 64 bit address expected (2 ints) */
244:     if ((type == xbee_64bitRemoteAT) ||
245:         (type == xbee_64bitData) ||
246:         (type == xbee_64bitIO)) {
247:         t = va_arg(ap, int);
248:         tAddr[0] = (t >> 24) & 0xFF;
249:         tAddr[1] = (t >> 16) & 0xFF;
250:         tAddr[2] = (t >> 8) & 0xFF;
251:         tAddr[3] = (t >> 0) & 0xFF;
252:         t = va_arg(ap, int);
253:         tAddr[4] = (t >> 24) & 0xFF;
254:         tAddr[5] = (t >> 16) & 0xFF;
255:         tAddr[6] = (t >> 8) & 0xFF;

```

```

256:     tAddr[7] = (t          ) & 0xFF;
257:
258:     /* if: 16 bit address expected (1 int) */
259: } else if ((type == xbee_16bitRemoteAT) ||
260:           (type == xbee_16bitData) ||
261:           (type == xbee_16bitIO)) {
262:     t = va_arg(ap, int);
263:     tAddr[0] = (t >> 8) & 0xFF;
264:     tAddr[1] = (t          ) & 0xFF;
265:     tAddr[2] = 0;
266:     tAddr[3] = 0;
267:     tAddr[4] = 0;
268:     tAddr[5] = 0;
269:     tAddr[6] = 0;
270:     tAddr[7] = 0;
271:
272:     /* otherwise clear the address */
273: } else {
274:     memset(tAddr, 0, 8);
275: }
276: va_end(ap);
277:
278: /* lock the connection mutex */
279: pthread_mutex_lock(&xbee.conmutex);
280:
281: /* are there any connections? */
282: if (xbee.conlist) {
283:     con = xbee.conlist;
284:     while (con) {
285:         /* if: after a modemStatus, and the types match! */
286:         if ((type == xbee_modemStatus) &&
287:             (con->type == type)) {
288:             pthread_mutex_unlock(&xbee.conmutex);
289:             return con;
290:
291:         /* if: after a txStatus and frameIDs match! */
292:         } else if ((type == xbee_txStatus) &&
293:                    (con->type == type) &&
294:                    (frameID == con->frameID)) {
295:             pthread_mutex_unlock(&xbee.conmutex);
296:             return con;
297:
298:         /* if: after a localAT, and the frameIDs match! */
299:         } else if ((type == xbee_localAT) &&
300:                    (con->type == type) &&
301:                    (frameID == con->frameID)) {
302:             pthread_mutex_unlock(&xbee.conmutex);
303:             return con;
304:
305:         /* if: connection types match, the frameIDs match, and the addresses match! */
306:         } else if ((type == con->type) &&
307:                    (frameID == con->frameID) &&
308:                    (!memcmp(tAddr, con->tAddr, 8))) {
309:             pthread_mutex_unlock(&xbee.conmutex);
310:             return con;
311:         }
312:
313:         /* if there are more, move along, dont want to loose that last item! */
314:         if (con->next == NULL) break;
315:         con = con->next;
316:     }
317:
318:     /* keep hold of the last connection... we will need to link it up later */
319:     ocon = con;
320: }
321:
322: /* create a new connection and set its attributes */
323: con = Xcalloc(sizeof(xbee_con));
324: con->type = type;
325: /* is it a 64bit connection? */
326: if ((type == xbee_64bitRemoteAT) ||
327:     (type == xbee_64bitData) ||
328:     (type == xbee_64bitIO)) {
329:     con->tAddr64 = TRUE;
330: }
331: con->atQueue = 0; /* queue AT commands? */
332: con->txDisableACK = 0; /* disable ACKs? */
333: con->txBroadcast = 0; /* broadcast? */
334: con->frameID = frameID;
335: memcpy(con->tAddr, tAddr, 8); /* copy in the remote address */
336:
337: #ifndef DEBUG
338:     switch(type) {
339:         case xbee_localAT:
340:             fprintf(stderr, "XBee: New local AT connection!\n");

```

```

341:         break;
342:     case xbee_16bitRemoteAT:
343:     case xbee_64bitRemoteAT:
344:         fprintf(stderr, "XBee: New %d-bit remote AT connection! (to: ", (con->tAddr64?64:16));
345:         for (i=0; i<(con->tAddr64?8:2); i++) {
346:             fprintf(stderr, (i?":%02X":"%02X"), tAddr[i]);
347:         }
348:         fprintf(stderr, ")\n");
349:         break;
350:     case xbee_16bitData:
351:     case xbee_64bitData:
352:         fprintf(stderr, "XBee: New %d-bit data connection! (to: ", (con->tAddr64?64:16));
353:         for (i=0; i<(con->tAddr64?8:2); i++) {
354:             fprintf(stderr, (i?":%02X":"%02X"), tAddr[i]);
355:         }
356:         fprintf(stderr, ")\n");
357:         break;
358:     case xbee_16bitIO:
359:     case xbee_64bitIO:
360:         fprintf(stderr, "XBee: New %d-bit IO connection! (to: ", (con->tAddr64?64:16));
361:         for (i=0; i<(con->tAddr64?8:2); i++) {
362:             fprintf(stderr, (i?":%02X":"%02X"), tAddr[i]);
363:         }
364:         fprintf(stderr, ")\n");
365:         break;
366:     case xbee_txStatus:
367:         fprintf(stderr, "XBee: New Tx status connection!\n");
368:         break;
369:     case xbee_modemStatus:
370:         fprintf(stderr, "XBee: New modem status connection!\n");
371:         break;
372:     case xbee_unknown:
373:     default:
374:         fprintf(stderr, "XBee: New unknown connection!\n");
375:     }
376: #endif
377:
378:     /* make it the last in the list */
379:     con->next = NULL;
380:     /* add it to the list */
381:     if (xbee.conlist) {
382:         ocon->next = con;
383:     } else {
384:         xbee.conlist = con;
385:     }
386:
387:     /* unlock the mutex */
388:     pthread_mutex_unlock(&xbee.conmutex);
389:     return con;
390: }
391:
392: /* #####
393: xbee_senddata
394: send the specified data to the provided connection */
395: xbee_pkt *xbee_senddata(xbee_con *con, char *format, ...) {
396:     xbee_pkt *p;
397:     va_list ap;
398:
399:     ISREADY;
400:
401:     /* xbee_vsnddata() wants a va_list... */
402:     va_start(ap, format);
403:     /* hand it over :) */
404:     p = xbee_vsnddata(con, format, ap);
405:     va_end(ap);
406:     return p;
407: }
408:
409: xbee_pkt *xbee_vsnddata(xbee_con *con, char *format, va_list ap) {
410:     t_data *pkt;
411:     int i, length;
412:     unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
413:     unsigned char data[128]; /* ditto */
414:     xbee_pkt *p = NULL; /* response packet */
415:     int to = 100; /* resonse timeout */
416:
417:     ISREADY;
418:
419:     if (!con) return (void *)-1;
420:     if (con->type == xbee_unknown) return (void *)-1;
421:
422:     /* make up the data and keep the length, its possible there are nulls in there */
423:     length = vsnprintf((char *)data, 128, format, ap);
424:
425: #ifdef DEBUG

```

```

426: fprintf(stderr,"XBee: ---- TX Packet =====--\n");
427: fprintf(stderr,"XBee: Length: %d\n",length);
428: for (i=0;i<length;i++) {
429:     fprintf(stderr,"XBee: %3d | 0x%02X ",i,data[i]);
430:     if ((data[i] > 32) && (data[i] < 127)) fprintf(stderr,"%c'\n",data[i]); else fprintf(stderr," _\n");
431: }
432: #endif
433:
434: /* ##### */
435: /* if: local AT */
436: if (con->type == xbee_localAT) {
437:     /* AT commands are 2 chars long (plus optional parameter) */
438:     if (length < 2) return (void *)-1;
439:
440:     /* use the command? */
441:     buf[0] = ((!con->atQueue)?0x08:0x09);
442:     buf[1] = con->frameID;
443:
444:     /* copy in the data */
445:     for (i=0;i<length;i++) {
446:         buf[i+2] = data[i];
447:     }
448:
449:     /* setup the packet */
450:     pkt = xbee_make_pkt(buf,i+2);
451:     /* send it on */
452:     xbee_send_pkt(pkt);
453:
454:     /* wait for a response packet */
455:     for (; p == NULL && to > 0; to--) {
456:         usleep(25400); /* tuned so that hopefully the first time round will catch the response */
457:         p = xbee_getpacket(con);
458:     }
459:
460:     /* if: no txStatus packet was recieved */
461:     if (to == 0) {
462: #ifdef DEBUG
463:         fprintf(stderr,"XBee: No AT status recieved before timeout\n");
464: #endif
465:         return NULL;
466:     }
467:
468: #ifdef DEBUG
469:     switch (p->status) {
470:     case 0x00: fprintf(stderr,"XBee: AT Status: OK!\n"); break;
471:     case 0x01: fprintf(stderr,"XBee: AT Status: Error\n"); break;
472:     case 0x02: fprintf(stderr,"XBee: AT Status: Invalid Command\n"); break;
473:     case 0x03: fprintf(stderr,"XBee: AT Status: Invalid Parameter\n"); break;
474:     }
475: #endif
476:     return p;
477:     /* ##### */
478:     /* if: remote AT */
479: } else if ((con->type == xbee_16bitRemoteAT) ||
480:           (con->type == xbee_64bitRemoteAT)) {
481:     if (length < 2) return (void *)-1; /* at commands are 2 chars long (plus optional parameter) */
482:     buf[0] = 0x17;
483:     buf[1] = con->frameID;
484:
485:     /* copy in the relevant address */
486:     if (con->tAddr64) {
487:         memcpy(&buf[2],con->tAddr,8);
488:         buf[10] = 0xFF;
489:         buf[11] = 0xFE;
490:     } else {
491:         memset(&buf[2],0,8);
492:         memcpy(&buf[10],con->tAddr,2);
493:     }
494:     /* queue the command? */
495:     buf[12] = ((!con->atQueue)?0x02:0x00);
496:
497:     /* copy in the data */
498:     for (i=0;i<length;i++) {
499:         buf[i+13] = data[i];
500:     }
501:
502:     /* setup the packet */
503:     pkt = xbee_make_pkt(buf,i+13);
504:     /* send it on */
505:     xbee_send_pkt(pkt);
506:
507:     /* wait for a response packet */
508:     for (; p == NULL && to > 0; to--) {
509:         usleep(25400); /* tuned so that hopefully the first time round will catch the response */
510:         p = xbee_getpacket(con);

```

```

511:     }
512:
513:     /* if: no txStatus packet was recieved */
514:     if (to == 0) {
515: #ifdef DEBUG
516:         fprintf(stderr, "XBee: No AT status recieved before timeout\n");
517: #endif
518:         return NULL;
519:     }
520:
521: #ifdef DEBUG
522:     switch (p->status) {
523:     case 0x00: fprintf(stderr, "XBee: AT Status: OK!\n"); break;
524:     case 0x01: fprintf(stderr, "XBee: AT Status: Error\n"); break;
525:     case 0x02: fprintf(stderr, "XBee: AT Status: Invalid Command\n"); break;
526:     case 0x03: fprintf(stderr, "XBee: AT Status: Invalid Parameter\n"); break;
527:     case 0x04: fprintf(stderr, "XBee: AT Status: No Response\n"); break;
528:     }
529: #endif
530:     return p;
531:     /* ##### */
532:     /* if: 16 or 64bit Data */
533:     } else if ((con->type == xbee_16bitData) ||
534:                (con->type == xbee_64bitData)) {
535:         int offset;
536:
537:         /* if: 16bit Data */
538:         if (con->type == xbee_16bitData) {
539:             buf[0] = 0x01;
540:             offset = 5;
541:             /* copy in the address */
542:             memcpy(&buf[2], con->tAddr, 2);
543:
544:             /* if: 64bit Data */
545:         } else { /* 64bit Data */
546:             buf[0] = 0x00;
547:             offset = 11;
548:             /* copy in the address */
549:             memcpy(&buf[2], con->tAddr, 8);
550:         }
551:
552:         /* copy frameID */
553:         buf[1] = con->frameID;
554:
555:         /* disable ack? broadcast? */
556:         buf[offset-1] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
557:
558:         /* copy in the data */
559:         for (i=0; i<length; i++) {
560:             buf[i+offset] = data[i];
561:         }
562:
563:         /* setup the packet */
564:         pkt = xbee_make_pkt(buf, i+offset);
565:         /* send it on */
566:         xbee_send_pkt(pkt);
567:
568:         /* wait for a response packet */
569:         for (; p == NULL && to > 0; to--) {
570:             usleep(25400); /* tuned so that hopefully the first time round will catch the response */
571:             p = xbee_getpacket(xbee.con_txStatus);
572:         }
573:
574:         /* if: no txStatus packet was recieved */
575:         if (to == 0) {
576: #ifdef DEBUG
577:             fprintf(stderr, "XBee: No txStatus recieved before timeout\n");
578: #endif
579:             return NULL;
580:         }
581:
582: #ifdef DEBUG
583:         switch (p->status) {
584:         case 0x00: fprintf(stderr, "XBee: txStatus: Success!\n"); break;
585:         case 0x01: fprintf(stderr, "XBee: txStatus: No ACK\n"); break;
586:         case 0x02: fprintf(stderr, "XBee: txStatus: CCA Failure\n"); break;
587:         case 0x03: fprintf(stderr, "XBee: txStatus: Purged\n"); break;
588:         }
589: #endif
590:         /* return the packet */
591:         return p;
592:         /* ##### */
593:         /* if: I/O */
594:     } else if ((con->type == xbee_64bitIO) ||
595:                (con->type == xbee_16bitIO)) {

```



```

596:     /* not currently implemented... is it even allowed? */
597:     fprintf(stderr, "***** TODO *****\n");
598: }
599:
600: return (void *)-1;
601: }
602:
603: /* #####
604: xbee_getpacket
605: retrieves the next packet destined for the given connection
606: once the packet has been retrieved, it is removed for the list! */
607: xbee_pkt *xbee_getpacket(xbee_con *con) {
608:     xbee_pkt *l, *p, *q;
609: #ifdef DEBUG
610:     int c;
611:     fprintf(stderr, "XBee: ---- Get Packet =====\n");
612: #endif
613:
614:     /* lock the packet mutex */
615:     pthread_mutex_lock(&xbee.pktmutex);
616:
617:     /* if: there are no packets */
618:     if ((p = xbee.pktlist) == NULL) {
619:         pthread_mutex_unlock(&xbee.pktmutex);
620: #ifdef DEBUG
621:         fprintf(stderr, "XBee: No packets available...\n");
622: #endif
623:         return NULL;
624:     }
625:
626:     l = NULL;
627:     q = NULL;
628:     /* get the first available packet for this socket */
629:     do {
630:         /* if: the connection type matches the packet type OR
631:          the connection is 16/64bit remote AT, and the packet is a remote AT response */
632:         if ((p->type == con->type) || /* -- */
            ((p->type == xbee_remoteAT) && /* -- */
             ((con->type == xbee_16bitRemoteAT) ||
              (con->type == xbee_64bitRemoteAT)))) {
633:
634:             /* if: the packet is modem status OR
635:              the packet is tx status or AT data and the frame IDs match OR
636:              the addresses match */
637:             if ((p->type == xbee_modemStatus) ||
638:                 ((p->type == xbee_txStatus) ||
639:                  (p->type == xbee_localAT) ||
640:                  (p->type == xbee_remoteAT)) &&
641:                 (con->frameID == p->frameID) ||
642:                 (!memcmp(con->tAddr, p->Addr64, 8))) {
643:                 q = p;
644:                 break;
645:             }
646:         }
647:         p = p->next;
648:     } while (p);
649:
650:     /* move on */
651:     l = p;
652:     p = p->next;
653: } while (p);
654:
655: /* if: no packet was found */
656: if (!q) {
657:     pthread_mutex_unlock(&xbee.pktmutex);
658: #ifdef DEBUG
659:     fprintf(stderr, "XBee: No packets available (for connection)...\n");
660: #endif
661:     return NULL;
662: }
663:
664: /* if it was the first packet */
665: if (!l) {
666:     /* move the chain along */
667:     xbee.pktlist = p->next;
668: } else {
669:     /* otherwise relink the list */
670:     l->next = p->next;
671: }
672:
673: #ifdef DEBUG
674:     fprintf(stderr, "XBee: Got a packet\n");
675:     for (p = xbee.pktlist, c = 0; p; c++, p = p->next);
676:     fprintf(stderr, "XBee: Packets left: %d\n", c);
677: #endif
678:
679: /* unlock the packet mutex */

```



```

681: pthread_mutex_unlock(&xbee.pktmutex);
682:
683:  /* and return the packet (must be freed by caller!) */
684:  return q;
685: }
686:
687: /* #####
688:     xbee_listen - INTERNAL
689:     the xbee xbee_listen thread
690:     reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
691: void xbee_listen(t_info *info) {
692:     unsigned char c, t, d[128];
693:     unsigned int l, i, chksum, o;
694: #ifdef DEBUG
695:     int j;
696: #endif
697:     xbee_pkt *p, *q, *po;
698:
699:     /* just falls out if the proper 'go-ahead' isn't given */
700:     if (xbee_ready != -1) return;
701:
702:     /* do this forever :) */
703:     while(1) {
704:         /* wait for a valid start byte */
705:         if (xbee_getRawByte() != 0x7E) continue;
706:
707: #ifdef DEBUG
708:         fprintf(stderr,"XBee: ---- RX Packet =====\nXBee: Got a packet!...\n");
709: #endif
710:
711:         /* get the length */
712:         l = xbee_getByte() << 8;
713:         l += xbee_getByte();
714:
715:         /* check it is a valid length... */
716:         if (!l) {
717: #ifdef DEBUG
718:             fprintf(stderr,"XBee: Recived zero length packet!\n");
719: #endif
720:             continue;
721:         }
722:         if (l > 100) {
723: #ifdef DEBUG
724:             fprintf(stderr,"XBee: Recived oversized packet! Length: %d\n",l - 1);
725: #endif
726:             continue;
727:         }
728:
729: #ifdef DEBUG
730:         fprintf(stderr,"XBee: Length: %d\n",l - 1);
731: #endif
732:
733:         /* get the packet type */
734:         t = xbee_getByte();
735:
736:         /* start the checksum */
737:         chksum = t;
738:
739:         /* suck in all the data */
740:         for (i = 0; l > 1 && i < 128; l--, i++) {
741:             /* get an unescaped byte */
742:             c = xbee_getByte();
743:             d[i] = c;
744:             chksum += c;
745: #ifdef DEBUG
746:             fprintf(stderr,"XBee: %3d | 0x%02X | ",i,c);
747:             if ((c > 32) && (c < 127)) fprintf(stderr,"%c'\n",c); else fprintf(stderr," _\n");
748: #endif
749:         }
750:         i--; /* it went up too many times!... */
751:
752:         /* add the checksum */
753:         chksum += xbee_getByte();
754:
755:         /* check if the whole packet was recieved, or something else occured... unlikely... */
756:         if (l > 1) {
757: #ifdef DEBUG
758:             fprintf(stderr,"XBee: Didn't get whole packet... :(\n");
759: #endif
760:             continue;
761:         }
762:
763:         /* check the checksum */
764:         if ((chksum & 0xFF) != 0xFF) {
765: #ifdef DEBUG

```

```

766:     fprintf(stderr,"XBee: Invalid Checksum: 0x%02X\n",chksum);
767: #endif
768:     continue;
769: }
770:
771: /* make a new packet */
772: po = p = Xcalloc(sizeof(xbee_pkt));
773: q = NULL;
774: p->datalen = 0;
775:
776: /* ##### */
777: /* if: modem status */
778: if (t == 0x8A) {
779: #ifdef DEBUG
780:     fprintf(stderr,"XBee: Packet type: Modem Status (0x8A)\n");
781:     fprintf(stderr,"XBee: ");
782:     switch (d[0]) {
783:     case 0x00: fprintf(stderr,"Hardware reset"); break;
784:     case 0x01: fprintf(stderr,"Watchdog timer reset"); break;
785:     case 0x02: fprintf(stderr,"Associated"); break;
786:     case 0x03: fprintf(stderr,"Disassociated"); break;
787:     case 0x04: fprintf(stderr,"Synchronization lost"); break;
788:     case 0x05: fprintf(stderr,"Coordinator realignment"); break;
789:     case 0x06: fprintf(stderr,"Coordinator started"); break;
790:     }
791:     fprintf(stderr,"...\n");
792: #endif
793:     p->type = xbee_modemStatus;
794:
795:     p->sAddr64 = FALSE;
796:     p->dataPkt = FALSE;
797:     p->txStatusPkt = FALSE;
798:     p->modemStatusPkt = TRUE;
799:     p->remoteATPkt = FALSE;
800:     p->IOPkt = FALSE;
801:
802:     /* modem status can only ever give 1 'data' byte */
803:     p->datalen = 1;
804:     p->data[0] = d[0];
805:
806:     /* ##### */
807:     /* if: local AT response */
808:     } else if (t == 0x88) {
809: #ifdef DEBUG
810:         fprintf(stderr,"XBee: Packet type: Local AT Response (0x88)\n");
811:         fprintf(stderr,"XBee: FrameID: 0x%02X\n",d[0]);
812:         fprintf(stderr,"XBee: AT Command: %c%c\n",d[1],d[2]);
813:         if (d[3] == 0) fprintf(stderr,"XBee: Status: OK\n");
814:         else if (d[3] == 1) fprintf(stderr,"XBee: Status: Error\n");
815:         else if (d[3] == 2) fprintf(stderr,"XBee: Status: Invalid Command\n");
816:         else if (d[3] == 3) fprintf(stderr,"XBee: Status: Invalid Parameter\n");
817: #endif
818:         p->type = xbee_localAT;
819:
820:         p->sAddr64 = FALSE;
821:         p->dataPkt = FALSE;
822:         p->txStatusPkt = FALSE;
823:         p->modemStatusPkt = FALSE;
824:         p->remoteATPkt = FALSE;
825:         p->IOPkt = FALSE;
826:
827:         p->frameID = d[0];
828:         p->atCmd[0] = d[1];
829:         p->atCmd[1] = d[2];
830:
831:         p->status = d[3];
832:
833:         /* copy in the data */
834:         p->datalen = i-3;
835:         for (;i>3;i--) p->data[i-4] = d[i];
836:
837:         /* ##### */
838:         /* if: remote AT response */
839:     } else if (t == 0x97) {
840: #ifdef DEBUG
841:         fprintf(stderr,"XBee: Packet type: Remote AT Response (0x97)\n");
842:         fprintf(stderr,"XBee: FrameID: 0x%02X\n",d[0]);
843:         fprintf(stderr,"XBee: 64-bit Address: ");
844:         for (j=0;j<8;j++) {
845:             fprintf(stderr,(j?"%02X":"%02X"),d[1+j]);
846:         }
847:         fprintf(stderr,"\n");
848:         fprintf(stderr,"XBee: 16-bit Address: ");
849:         for (j=0;j<2;j++) {
850:             fprintf(stderr,(j?"%02X":"%02X"),d[9+j]);

```

```

851:     }
852:     fprintf(stderr, "\n");
853:     fprintf(stderr, "XBee: AT Command: %c%c\n", d[11], d[12]);
854:     if (d[13] == 0) fprintf(stderr, "XBee: Status: OK\n");
855:     else if (d[13] == 1) fprintf(stderr, "XBee: Status: Error\n");
856:     else if (d[13] == 2) fprintf(stderr, "XBee: Status: Invalid Command\n");
857:     else if (d[13] == 3) fprintf(stderr, "XBee: Status: Invalid Parameter\n");
858:     else if (d[13] == 4) fprintf(stderr, "XBee: Status: No Response\n");
859: #endif
860:     p->type = xbee_remoteAT;
861:
862:     p->sAddr64 = FALSE;
863:     p->dataPkt = FALSE;
864:     p->txStatusPkt = FALSE;
865:     p->modemStatusPkt = FALSE;
866:     p->remoteATPkt = TRUE;
867:     p->IOPkt = FALSE;
868:
869:     p->frameID = d[0];
870:
871:     p->Addr64[0] = d[1];
872:     p->Addr64[1] = d[2];
873:     p->Addr64[2] = d[3];
874:     p->Addr64[3] = d[4];
875:     p->Addr64[4] = d[5];
876:     p->Addr64[5] = d[6];
877:     p->Addr64[6] = d[7];
878:     p->Addr64[7] = d[8];
879:
880:     p->Addr16[0] = d[9];
881:     p->Addr16[1] = d[10];
882:
883:     p->atCmd[0] = d[11];
884:     p->atCmd[1] = d[12];
885:
886:     p->status = d[13];
887:
888:     /* copy in the data */
889:     p->datalen = i-13;
890:     for (; i>13; i--) p->data[i-14] = d[i];
891:
892:     /* ##### */
893:     /* if: TX status */
894:     } else if (t == 0x89) {
895: #ifdef DEBUG
896:     fprintf(stderr, "XBee: Packet type: TX Status Report (0x89)\n");
897:     fprintf(stderr, "XBee: FrameID: 0x%02X\n", d[0]);
898:     if (d[1] == 0) fprintf(stderr, "XBee: Status: Success\n");
899:     else if (d[1] == 1) fprintf(stderr, "XBee: Status: No ACK\n");
900:     else if (d[1] == 2) fprintf(stderr, "XBee: Status: CCA Failure\n");
901:     else if (d[1] == 3) fprintf(stderr, "XBee: Status: Purged\n");
902: #endif
903:     p->type = xbee_txStatus;
904:
905:     p->sAddr64 = FALSE;
906:     p->dataPkt = FALSE;
907:     p->txStatusPkt = TRUE;
908:     p->modemStatusPkt = FALSE;
909:     p->remoteATPkt = FALSE;
910:     p->IOPkt = FALSE;
911:
912:     p->frameID = d[0];
913:
914:     p->status = d[1];
915:
916:     /* never returns data */
917:     p->datalen = 0;
918:
919:     /* ##### */
920:     /* if: 16 / 64bit data receive */
921:     } else if ((t == 0x80) ||
922:                (t == 0x81)) {
923:         int offset;
924:         if (t == 0x80) { /* 64bit */
925:             offset = 8;
926:         } else { /* 16bit */
927:             offset = 2;
928:         }
929: #ifdef DEBUG
930:         fprintf(stderr, "XBee: Packet type: %d-bit RX Data (0x%02X)\n", ((t == 0x80)?64:16), t);
931:         fprintf(stderr, "XBee: %d-bit Address: ", ((t == 0x80)?64:16));
932:         for (j=0; j<offset; j++) {
933:             fprintf(stderr, (j?"%02X":"%02X"), d[j]);
934:         }
935:         fprintf(stderr, "\n");

```

```

936:     fprintf(stderr,"XBee: RSSI: -%ddb\n",d[offset]);
937:     if (d[offset + 1] & 0x02) fprintf(stderr,"XBee: Options: Address Broadcast\n");
938:     if (d[offset + 1] & 0x03) fprintf(stderr,"XBee: Options: PAN Broadcast\n");
939: #endif
940:     p->dataPkt = TRUE;
941:     p->txStatusPkt = FALSE;
942:     p->modemStatusPkt = FALSE;
943:     p->remoteATPkt = FALSE;
944:     p->IOPkt = FALSE;
945:
946:     if (t == 0x82) { /* 64bit */
947:         p->type = xbee_64bitData;
948:
949:         p->sAddr64 = TRUE;
950:
951:         p->Addr64[0] = d[0];
952:         p->Addr64[1] = d[1];
953:         p->Addr64[2] = d[2];
954:         p->Addr64[3] = d[3];
955:         p->Addr64[4] = d[4];
956:         p->Addr64[5] = d[5];
957:         p->Addr64[6] = d[6];
958:         p->Addr64[7] = d[7];
959:     } else { /* 16bit */
960:         p->type = xbee_16bitData;
961:
962:         p->sAddr64 = FALSE;
963:
964:         p->Addr16[0] = d[0];
965:         p->Addr16[1] = d[1];
966:     }
967:
968:     /* save the RSSI / signal strength
969:      this can be used with printf as:
970:      printf("-%ddb\n",p->RSSI); */
971:     p->RSSI = d[offset];
972:
973:     p->status = d[offset + 1];
974:
975:     /* copy in the data */
976:     p->datalen = i-(offset + 1);
977:     for (;i>offset + 1;i--) p->data[i-(offset + 2)] = d[i];
978:
979:     /* ##### */
980:     /* if: 16 / 64bit I/O recieve */
981:     } else if ((t == 0x82) ||
982:               (t == 0x83)) {
983:         int offset, samples;
984:         if (t == 0x82) { /* 64bit */
985:             offset = 8;
986:             samples = d[10];
987:         } else { /* 16bit */
988:             offset = 2;
989:             samples = d[4];
990:         }
991: #ifdef DEBUG
992:         fprintf(stderr,"XBee: Packet type: %d-bit RX I/O Data (0x%02X)\n",((t == 0x82)?64:16),t);
993:         fprintf(stderr,"XBee: %d-bit Address: ",((t == 0x82)?64:16));
994:         for (j = 0; j < offset; j++) {
995:             fprintf(stderr,(j?"%02X":"%02X"),d[j]);
996:         }
997:         fprintf(stderr,"\n");
998:         fprintf(stderr,"XBee: RSSI: -%ddb\n",d[offset]);
999:         if (d[9] & 0x02) fprintf(stderr,"XBee: Options: Address Broadcast\n");
1000:        if (d[9] & 0x03) fprintf(stderr,"XBee: Options: PAN Broadcast\n");
1001:        fprintf(stderr,"XBee: Samples: %d\n",d[offset + 2]);
1002: #endif
1003:        i = offset + 5;
1004:
1005:        /* each sample is split into its own packet here, for simplicity */
1006:        for (o = samples; o > 0; o--) {
1007: #ifdef DEBUG
1008:            fprintf(stderr,"XBee: --- Sample %3d ----- \n", o - samples + 1);
1009: #endif
1010:            /* if we arent still using the original packet */
1011:            if (o < samples) {
1012:                /* make a new one and link it up! */
1013:                q = Xcalloc(sizeof(xbee_pkt));
1014:                p->next = q;
1015:                p = q;
1016:            }
1017:
1018:            /* never returns data */
1019:            p->datalen = 0;
1020:

```

```

1021:         p->dataPkt = FALSE;
1022:         p->txStatusPkt = FALSE;
1023:         p->modemStatusPkt = FALSE;
1024:         p->remoteATPkt = FALSE;
1025:         p->IOPkt = TRUE;
1026:
1027:         if (t == 0x82) { /* 64bit */
1028:             p->type = xbee_64bitIO;
1029:
1030:             p->sAddr64 = TRUE;
1031:
1032:             p->Addr64[0] = d[0];
1033:             p->Addr64[1] = d[1];
1034:             p->Addr64[2] = d[2];
1035:             p->Addr64[3] = d[3];
1036:             p->Addr64[4] = d[4];
1037:             p->Addr64[5] = d[5];
1038:             p->Addr64[6] = d[6];
1039:             p->Addr64[7] = d[7];
1040:         } else { /* 16bit */
1041:             p->type = xbee_16bitIO;
1042:
1043:             p->sAddr64 = FALSE;
1044:
1045:             p->Addr16[0] = d[0];
1046:             p->Addr16[1] = d[1];
1047:         }
1048:
1049:         /* save the RSSI / signal strength
1050:            this can be used with printf as:
1051:            printf("-%ddb\n",p->RSSI); */
1052:         p->RSSI = d[offset];
1053:
1054:         p->status = d[offset + 1];
1055:
1056:         /* copy in the I/O data mask */
1057:         p->IOMask = (((d[offset + 3]<<8) | d[offset + 4]) & 0x7FFF);
1058:
1059:         /* copy in the digital I/O data */
1060:         p->IOdata = (((d[i]<<8) | d[i+1]) & 0x01FF);
1061:
1062:         /* advance over the digital data, if its there */
1063:         i += (((d[offset + 3]&0x01)|((d[offset + 4]))?2:0);
1064:
1065:         /* copy in the analog I/O data */
1066:         if (d[i]&0x02) {p->IOanalog[0] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1067:         if (d[i]&0x04) {p->IOanalog[1] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1068:         if (d[i]&0x08) {p->IOanalog[2] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1069:         if (d[i]&0x10) {p->IOanalog[3] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1070:         if (d[i]&0x20) {p->IOanalog[4] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1071:         if (d[i]&0x40) {p->IOanalog[5] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1072: #ifndef DEBUG
1073:         if (p->IOMask & 0x0001) fprintf(stderr,"XBee: Digital 0: %c\n",((p->IOdata & 0x0001)?'1':'0'));
1074:         if (p->IOMask & 0x0002) fprintf(stderr,"XBee: Digital 1: %c\n",((p->IOdata & 0x0002)?'1':'0'));
1075:         if (p->IOMask & 0x0004) fprintf(stderr,"XBee: Digital 2: %c\n",((p->IOdata & 0x0004)?'1':'0'));
1076:         if (p->IOMask & 0x0008) fprintf(stderr,"XBee: Digital 3: %c\n",((p->IOdata & 0x0008)?'1':'0'));
1077:         if (p->IOMask & 0x0010) fprintf(stderr,"XBee: Digital 4: %c\n",((p->IOdata & 0x0010)?'1':'0'));
1078:         if (p->IOMask & 0x0020) fprintf(stderr,"XBee: Digital 5: %c\n",((p->IOdata & 0x0020)?'1':'0'));
1079:         if (p->IOMask & 0x0040) fprintf(stderr,"XBee: Digital 6: %c\n",((p->IOdata & 0x0040)?'1':'0'));
1080:         if (p->IOMask & 0x0080) fprintf(stderr,"XBee: Digital 7: %c\n",((p->IOdata & 0x0080)?'1':'0'));
1081:         if (p->IOMask & 0x0100) fprintf(stderr,"XBee: Digital 8: %c\n",((p->IOdata & 0x0100)?'1':'0'));
1082:         if (p->IOMask & 0x0200) fprintf(stderr,"XBee: Analog 0: %.2fv\n",(3.3/1023)*p->IOanalog[0]);
1083:         if (p->IOMask & 0x0400) fprintf(stderr,"XBee: Analog 1: %.2fv\n",(3.3/1023)*p->IOanalog[1]);
1084:         if (p->IOMask & 0x0800) fprintf(stderr,"XBee: Analog 2: %.2fv\n",(3.3/1023)*p->IOanalog[2]);
1085:         if (p->IOMask & 0x1000) fprintf(stderr,"XBee: Analog 3: %.2fv\n",(3.3/1023)*p->IOanalog[3]);
1086:         if (p->IOMask & 0x2000) fprintf(stderr,"XBee: Analog 4: %.2fv\n",(3.3/1023)*p->IOanalog[4]);
1087:         if (p->IOMask & 0x4000) fprintf(stderr,"XBee: Analog 5: %.2fv\n",(3.3/1023)*p->IOanalog[5]);
1088: #endif
1089:     }
1090: #ifdef DEBUG
1091:     fprintf(stderr,"XBee: ----- \n");
1092: #endif
1093:
1094:     /* ##### */
1095:     /* if: Unknown */
1096:     } else {
1097: #ifdef DEBUG
1098:         fprintf(stderr,"XBee: Packet type: Unknown (0x%02X)\n",t);
1099: #endif
1100:         p->type = xbee_unknown;
1101:     }
1102:     p->next = NULL;
1103:
1104:     /* lock the packet mutex, so we can safely add the packet to the list */
1105:     pthread_mutex_lock(&xbee.pktmutex);

```

```

1106:     i = 1;
1107:     /* if: the list is empty */
1108:     if (!xbee.pktlist) {
1109:         /* start the list! */
1110:         xbee.pktlist = po;
1111:     } else {
1112:         /* add the packet to the end */
1113:         q = xbee.pktlist;
1114:         while (q->next) {
1115:             q = q->next;
1116:             i++;
1117:         }
1118:         q->next = po;
1119:     }
1120:
1121: #ifdef DEBUG
1122:     while (q && q->next) {
1123:         q = q->next;
1124:         i++;
1125:     }
1126:     fprintf(stderr,"XBee: -----\\n");
1127:     fprintf(stderr,"XBee: Packets: %d\\n",i);
1128: #endif
1129:
1130:     po = p = q = NULL;
1131:
1132:     /* unlock the packet mutex */
1133:     pthread_mutex_unlock(&xbee.pktmutex);
1134: }
1135: }
1136:
1137: /* #####
1138: xbee_getByte - INTERNAL
1139: waits for an escaped byte of data */
1140: unsigned char xbee_getByte(void) {
1141:     unsigned char c;
1142:
1143:     ISREADY;
1144:
1145:     /* take a byte */
1146:     c = xbee_getRawByte();
1147:     /* if its escaped, take another and un-escape */
1148:     if (c == 0x7D) c = xbee_getRawByte() ^ 0x20;
1149:
1150:     return (c & 0xFF);
1151: }
1152:
1153: /* #####
1154: xbee_getRawByte - INTERNAL
1155: waits for a raw byte of data */
1156: unsigned char xbee_getRawByte(void) {
1157:     unsigned char c;
1158:     fd_set fds;
1159:
1160:     ISREADY;
1161:
1162:     /* wait for a read to be possible */
1163:     FD_ZERO(&fds);
1164:     FD_SET(xbee.ttyfd,&fds);
1165:     if (select(xbee.ttyfd+1,&fds,NULL,NULL,NULL) == -1) {
1166:         perror("xbee:xbee_listen():xbee_getByte()");
1167:         exit(1);
1168:     }
1169:
1170:     /* read 1 character
1171:     the loop is just incase there actually isnt a byte there to be read... */
1172:     do {
1173:         if (read(xbee.ttyfd,&c,1) == 0) {
1174:             usleep(10);
1175:             continue;
1176:         }
1177:     } while (0);
1178:
1179:     return (c & 0xFF);
1180: }
1181:
1182: /* #####
1183: xbee_send_pkt - INTERNAL
1184: sends a complete packet of data */
1185: void xbee_send_pkt(t_data *pkt) {
1186:     ISREADY;
1187:
1188:
1189:     /* lock the send mutex */
1190:     pthread_mutex_lock(&xbee.sendmutex);

```

```

1191:
1192:  /* write and flush the data */
1193:  fwrite(pkt->data,pkt->length,1,xbee.tty);
1194:  fflush(xbee.tty);
1195:
1196:  /* unlock the mutex */
1197:  pthread_mutex_unlock(&xbee.sendmutex);
1198:
1199: #ifdef DEBUG
1200: {
1201:     int i;
1202:     /* prints packet in hex byte-by-byte */
1203:     fprintf(stderr,"XBee: TX Packet - ");
1204:     for (i=0;i<pkt->length;i++) {
1205:         fprintf(stderr,"0x%02X ",pkt->data[i]);
1206:     }
1207:     fprintf(stderr,"\n");
1208: }
1209: #endif
1210:
1211:  /* free the packet */
1212:  Xfree(pkt);
1213: }
1214:
1215: /* #####
1216:  xbee_make_pkt - INTERNAL
1217:  adds delimiter field
1218:  calculates length and checksum
1219:  escapes bytes */
1220: t_data *xbee_make_pkt(unsigned char *data, int length) {
1221:     t_data *pkt;
1222:     unsigned int l, i, o, t, x, m;
1223:     char d = 0;
1224:
1225:     ISREADY;
1226:
1227:     /* check the data given isnt too long
1228:      100 bytes maximum payload + 12 bytes header information */
1229:     if (length > 100 + 12) return NULL;
1230:
1231:     /* calculate the length of the whole packet
1232:      start, length (MSB), length (LSB), DATA, checksum */
1233:     l = 3 + length + 1;
1234:
1235:     /* prepare memory */
1236:     pkt = Xcalloc(sizeof(t_data));
1237:
1238:     /* put start byte on */
1239:     pkt->data[0] = 0x7E;
1240:
1241:     /* copy data into packet */
1242:     for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
1243:         /* if: its time for the checksum */
1244:         if (i == length) d = M8((0xFF - M8(t)));
1245:         /* if: its time for the high length byte */
1246:         else if (m == 1) d = M8(length >> 8);
1247:         /* if: its time for the low length byte */
1248:         else if (m == 2) d = M8(length);
1249:         /* if: its time for the normal data */
1250:         else if (m > 2) d = data[i];
1251:
1252:         x = 0;
1253:         /* check for any escapes needed */
1254:         if ((d == 0x11) || /* XON */
1255:             (d == 0x13) || /* XOFF */
1256:             (d == 0x7D) || /* Escape */
1257:             (d == 0x7E)) { /* Frame Delimiter */
1258:             l++;
1259:             pkt->data[o++] = 0x7D;
1260:             x = 1;
1261:         }
1262:
1263:         /* move data in */
1264:         pkt->data[o] = ((!x)?d:d^0x20);
1265:         if (m > 2) {
1266:             i++;
1267:             t += d;
1268:         }
1269:     }
1270:
1271:     /* remember the length */
1272:     pkt->length = l;
1273:
1274:     return pkt;
1275: }

```