

```
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 <stdio.h>
22: #include <stdlib.h>
23:
24: #include <stdarg.h>
25:
26: #include <string.h>
27: #include <fcntl.h>
28: #include <errno.h>
29: #include <signal.h>
30:
31: #ifdef __GNUC__ /* ---- */
32: #include <unistd.h>
33: #include <termios.h>
34: #include <pthread.h>
35: #include <sys/time.h>
36: #else /* ----- */
37: #include <Windows.h>
38: #include <io.h>
39: #include <time.h>
40: #endif /* ----- */
41:
42: #include "xbee.h"
43: #include "api.h"
44:
45: #ifdef __GNUC__ /* ---- */
46: #include "xsys/linux.c"
47: #else /* ----- */
48: #include "xsys/win32.c"
49: #endif /* ----- */
50:
51:
52: #ifdef __UMAKEFILE
53: /* for embedded compiling */
54: const char *xbee_svn_version(void) {
55:     return "Embedded";
56: }
57: #endif
58:
59: /* ##### */
60: /* ### Memory Handling ##### */
61: /* ##### */
62:
63: /* malloc wrapper function */
64: static void *Xmalloc(size_t size) {
65:     void *t;
66:     t = malloc(size);
67:     if (!t) {
68:         /* uhoh... thats pretty bad... */
69:         perror("libxbee:malloc()");
70:         exit(1);
71:     }
72:     return t;
73: }
74:
75: /* calloc wrapper function */
76: static void *Xcalloc(size_t size) {
77:     void *t;
78:     t = calloc(1, size);
79:     if (!t) {
80:         /* uhoh... thats pretty bad... */
81:         perror("libxbee:calloc()");
82:         exit(1);
83:     }
84:     return t;
85: }
```

```

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

```

```
171:     xbee_sendAT - INTERNAL
172:     allows for an at command to be send, and the reply to be captured */
173: static int xbee_sendAT(char *command, char *retBuf, int retBuflen) {
174:     return xbee_sendATdelay(0,0,command,retBuf, retBuflen);
175: }
176: static int xbee_sendATdelay(int preDelay, int postDelay, char *command, char *retBuf, int retBuflen) {
177:     struct timeval to;
178:
179:     int ret;
180:     int bufi = 0;
181:
182:     /* if there is a preDelay given, then use it and a bit more */
183:     if (preDelay) usleep(preDelay * 1200);
184:
185:     /* get rid of any pre-command sludge... */
186:     memset(&to, 0, sizeof(to));
187:     ret = xbee_select(&to);
188:     if (ret > 0) {
189:         char t[128];
190:         while (xbee_read(t,127));
191:     }
192:
193:     /* send the requested command */
194:     if (xbee.log) xbee_log("sendATdelay: Sending '%s'", command);
195:     xbee_write(command, strlen(command));
196:
197:     /* if there is a postDelay, then use it */
198:     if (postDelay) {
199:         usleep(postDelay * 900);
200:
201:         /* get rid of any post-command sludge... */
202:         memset(&to, 0, sizeof(to));
203:         ret = xbee_select(&to);
204:         if (ret > 0) {
205:             char t[128];
206:             while (xbee_read(t,127));
207:         }
208:     }
209:
210:     /* retrieve the data */
211:     memset(retBuf, 0, retBuflen);
212:     memset(&to, 0, sizeof(to));
213:     /* select on the xbee fd... wait at most 200ms for the response */
214:     to.tv_usec = 200000;
215:     if ((ret = xbee_select(&to)) == -1) {
216:         perror("libxbee:xbee_sendATdelay()");
217:         exit(1);
218:     }
219:
220:     if (!ret) {
221:         /* timed out, and there is nothing to be read */
222:         if (xbee.log) xbee_log("sendATdelay: No Data to read - Timeout...");
223:         return 1;
224:     }
225:
226:     /* check for any dribble... */
227:     do {
228:         /* if there is actually no space in the retBuf then break out */
229:         if (bufi >= retBuflen - 1) {
230:             break;
231:         }
232:
233:         /* read as much data as is possible into retBuf */
234:         if ((ret = xbee_read(&retBuf[bufi], retBuflen - bufi - 1)) == 0) {
235:             break;
236:         }
237:
238:         /* advance the 'end of string' pointer */
239:         bufi += ret;
240:
241:         /* wait at most 100ms for any more data */
242:         memset(&to, 0, sizeof(to));
243:         to.tv_usec = 100000;
244:         if ((ret = xbee_select(&to)) == -1) {
245:             perror("libxbee:xbee_sendATdelay()");
246:             exit(1);
247:         }
248:
249:         /* loop while data was read */
250:     } while (ret);
251:
252:     if (!bufi) {
253:         if (xbee.log) xbee_log("sendATdelay: No response...");
254:         return 1;
255:     }
```

```

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

```

```

341:
342:     if (xbee.log) fflush(xbee.log);
343:
344:     /* nullify everything */
345:
346:     /* free all connections */
347:     con = xbee.conlist;
348:     xbee.conlist = NULL;
349:     while (con) {
350:         ncon = con->next;
351:         Xfree(con);
352:         con = ncon;
353:     }
354:
355:     /* free all packets */
356:     xbee.pktlast = NULL;
357:     pkt = xbee.pktlist;
358:     xbee.pktlist = NULL;
359:     while (pkt) {
360:         npkt = pkt->next;
361:         Xfree(pkt);
362:         pkt = npkt;
363:     }
364:
365:     /* destroy mutexes */
366:     xbee_mutex_destroy(xbee.conmutex);
367:     xbee_mutex_destroy(xbee.pktmutex);
368:     xbee_mutex_destroy(xbee.sendmutex);
369:
370:     /* close the serial port */
371:     Xfree(xbee.path);
372:     #ifdef __GNUC__ /* ---- */
373:     if (xbee.tty) fclose(xbee.tty);
374:     if (xbee.ttyfd) close(xbee.ttyfd);
375:     #else /* ----- */
376:     if (xbee.tty) CloseHandle(xbee.tty);
377:     #endif /* ----- */
378:
379:     /* close log and tty */
380:     if (xbee.log) {
381:         fprintf(xbee.log, "libxbee: Stopped! (%s)\n", xbee_svn_version());
382:         fflush(xbee.log);
383:         fclose(xbee.log);
384:     }
385:
386:     /* wipe everything else... */
387:     memset(&xbee, 0, sizeof(xbee));
388:
389:     return ret;
390: }
391:
392: /* #####
393: xbee_setup
394: opens xbee serial port & creates xbee listen thread
395: the xbee must be configured for API mode 2
396: THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
397: int xbee_setup(char *path, int baudrate) {
398:     return xbee_setuplogAPI(path, baudrate, 0, 0, 0);
399: }
400: int xbee_setuplog(char *path, int baudrate, int logfd) {
401:     return xbee_setuplogAPI(path, baudrate, logfd, 0, 0);
402: }
403: int xbee_setupAPI(char *path, int baudrate, char cmdSeq, int cmdTime) {
404:     return xbee_setuplogAPI(path, baudrate, 0, cmdSeq, cmdTime);
405: }
406: int xbee_setuplogAPI(char *path, int baudrate, int logfd, char cmdSeq, int cmdTime) {
407:     #ifdef __GNUC__ /* ---- */
408:     struct flock fl;
409:     struct termios tc;
410:     speed_t chosenbaud;
411:     #else /* ----- */
412:     int chosenbaud;
413:     DCB tc;
414:     int evtMask;
415:     COMMTIMEOUTS timeouts;
416:     #endif /* ----- */
417:     t_info info;
418:
419:     memset(&xbee, 0, sizeof(xbee));
420:
421:     #ifdef DEBUG
422:     /* logfd or stderr */
423:     xbee.logfd = ((logfd)?logfd:2);
424:     #else
425:     xbee.logfd = logfd;

```

```

426: #endif
427:     if (xbee.logfd) {
428:         xbee.log = fdopen(xbee.logfd, "w");
429:         if (!xbee.log) {
430:             /* errno == 9 is bad file descriptor (probably not provided) */
431:             if (errno != 9) perror("xbee_setup(): Failed opening logfile");
432:             xbee.logfd = 0;
433:         } else {
434:             /* set to line buffer - ensure lines are written to file when complete */
435: #ifdef __GNUC__ /* ---- */
436:             setvbuf(xbee.log, NULL, _IOLBF, BUFSIZ);
437: #else /* ----- */
438:             /* Win32 is rubbish... so we have to completely disable buffering... */
439:             setvbuf(xbee.log, NULL, _IONBF, BUFSIZ);
440: #endif /* ----- */
441:         }
442:     }
443:
444:     if (xbee.log) fprintf(xbee.log, "libxbee: Starting (%s)...\n", xbee_svn_version());
445:
446: #ifdef __GNUC__ /* ---- */
447:     /* select the baud rate */
448:     switch (baudrate) {
449:         case 1200: chosenbaud = B1200; break;
450:         case 2400: chosenbaud = B2400; break;
451:         case 4800: chosenbaud = B4800; break;
452:         case 9600: chosenbaud = B9600; break;
453:         case 19200: chosenbaud = B19200; break;
454:         case 38400: chosenbaud = B38400; break;
455:         case 57600: chosenbaud = B57600; break;
456:         case 115200: chosenbaud = B115200; break;
457:         default:
458:             fprintf(stderr, "%s(): Unknown or incompatiable baud rate specified... (%d)\n", __FUNCTION__, baudrate);
459:             return -1;
460:     };
461: #endif /* ----- */
462:
463:     /* setup the connection stuff */
464:     xbee.conlist = NULL;
465:
466:     /* setup the packet stuff */
467:     xbee.pktlist = NULL;
468:     xbee.pktlast = NULL;
469:     xbee.pktcount = 0;
470:     xbee.listenrun = 1;
471:
472:     /* setup the mutexes */
473:     if (xbee_mutex_init(xbee.conmutex)) {
474:         perror("xbee_setup():xbee_mutex_init(conmutex)");
475:         return -1;
476:     }
477:     if (xbee_mutex_init(xbee.pktmutex)) {
478:         perror("xbee_setup():xbee_mutex_init(pktmutex)");
479:         xbee_mutex_destroy(xbee.conmutex);
480:         return -1;
481:     }
482:     if (xbee_mutex_init(xbee.sendmutex)) {
483:         perror("xbee_setup():xbee_mutex_init(sendmutex)");
484:         xbee_mutex_destroy(xbee.conmutex);
485:         xbee_mutex_destroy(xbee.pktmutex);
486:         return -1;
487:     }
488:
489:     /* take a copy of the XBee device path */
490:     if ((xbee.path = Xmalloc(sizeof(char) * (strlen(path) + 1))) == NULL) {
491:         perror("xbee_setup():Xmalloc(path)");
492:         xbee_mutex_destroy(xbee.conmutex);
493:         xbee_mutex_destroy(xbee.pktmutex);
494:         xbee_mutex_destroy(xbee.sendmutex);
495:         return -1;
496:     }
497:     strcpy(xbee.path, path);
498:
499: #ifdef __GNUC__ /* ---- */
500:     /* open the serial port as a file descriptor */
501:     if ((xbee.ttyfd = open(path, O_RDWR | O_NOCTTY | O_NONBLOCK)) == -1) {
502:         perror("xbee_setup():open()");
503:         xbee_mutex_destroy(xbee.conmutex);
504:         xbee_mutex_destroy(xbee.pktmutex);
505:         xbee_mutex_destroy(xbee.sendmutex);
506:         Xfree(xbee.path);
507:         return -1;
508:     }
509:
510:     /* lock the file */

```

```

511: fl.l_type = F_WRLCK | F_RDLCK;
512: fl.l_whence = SEEK_SET;
513: fl.l_start = 0;
514: fl.l_len = 0;
515: fl.l_pid = getpid();
516: if (fcntl(xbee.ttyfd, F_SETLK, &fl) == -1) {
517:     perror("xbee_setup():fcntl()");
518:     xbee_mutex_destroy(xbee.conmutex);
519:     xbee_mutex_destroy(xbee.pktmutex);
520:     xbee_mutex_destroy(xbee.sendmutex);
521:     Xfree(xbee.path);
522:     close(xbee.ttyfd);
523:     return -1;
524: }
525:
526: /* open the serial port as a FILE* */
527: if ((xbee.tty = fdopen(xbee.ttyfd, "r+")) == NULL) {
528:     perror("xbee_setup():fdopen()");
529:     xbee_mutex_destroy(xbee.conmutex);
530:     xbee_mutex_destroy(xbee.pktmutex);
531:     xbee_mutex_destroy(xbee.sendmutex);
532:     Xfree(xbee.path);
533:     close(xbee.ttyfd);
534:     return -1;
535: }
536:
537: /* flush the serial port */
538: fflush(xbee.tty);
539:
540: /* disable buffering */
541: setvbuf(xbee.tty, NULL, _IONBF, BUFSIZ);
542:
543: /* setup the baud rate and other io attributes */
544: tcgetattr(xbee.ttyfd, &tc);
545: /* input flags */
546: tc.c_iflag &= ~ IGNBRK; /* enable ignoring break */
547: tc.c_iflag &= ~(IGNPAR | PARMRK); /* disable parity checks */
548: tc.c_iflag &= ~ INPCK; /* disable parity checking */
549: tc.c_iflag &= ~ ISTRIP; /* disable stripping 8th bit */
550: tc.c_iflag &= ~(INLCR | ICRNL); /* disable translating NL <-> CR */
551: tc.c_iflag &= ~ IGNCR; /* disable ignoring CR */
552: tc.c_iflag &= ~(IXON | IXOFF); /* disable XON/XOFF flow control */
553: /* output flags */
554: tc.c_oflag &= ~ OPOST; /* disable output processing */
555: tc.c_oflag &= ~(ONLCR | OCRNL); /* disable translating NL <-> CR */
556: tc.c_oflag &= ~ OFILL; /* disable fill characters */
557: /* control flags */
558: tc.c_cflag |= CREAD; /* enable reciever */
559: tc.c_cflag &= ~ PARENB; /* disable parity */
560: tc.c_cflag &= ~ CSTOPB; /* disable 2 stop bits */
561: tc.c_cflag &= ~ CSIZE; /* remove size flag... */
562: tc.c_cflag |= CS8; /* ...enable 8 bit characters */
563: tc.c_cflag |= HUPCL; /* enable lower control lines on close - hang up */
564: /* local flags */
565: tc.c_lflag &= ~ ISIG; /* disable generating signals */
566: tc.c_lflag &= ~ ICANON; /* disable canonical mode - line by line */
567: tc.c_lflag &= ~ ECHO; /* disable echoing characters */
568: tc.c_lflag &= ~ ECHONL; /* ??? */
569: tc.c_lflag &= ~ NOFLSH; /* disable flushing on SIGINT */
570: tc.c_lflag &= ~ IEXTEN; /* disable input processing */
571: /* control characters */
572: memset(tc.c_cc, 0, sizeof(tc.c_cc));
573: /* i/o rates */
574: cfsetspeed(&tc, chosenbaud); /* set i/o baud rate */
575: tcsetattr(xbee.ttyfd, TCSANOW, &tc);
576: tcflow(xbee.ttyfd, TCOON|TCION); /* enable input & output transmission */
577: #else /* ----- */
578: /* open the serial port */
579: xbee.tty = CreateFile(TEXT(path),
580:     GENERIC_READ | GENERIC_WRITE,
581:     0, /* exclusive access */
582:     NULL, /* default security attributes */
583:     OPEN_EXISTING,
584:     FILE_FLAG_OVERLAPPED,
585:     NULL);
586: if (xbee.tty == INVALID_HANDLE_VALUE) {
587:     perror("xbee_setup():CreateFile()");
588:     xbee_mutex_destroy(xbee.conmutex);
589:     xbee_mutex_destroy(xbee.pktmutex);
590:     xbee_mutex_destroy(xbee.sendmutex);
591:     Xfree(xbee.path);
592:     return -1;
593: }
594:
595: GetCommState(xbee.tty, &tc);

```



```

596: tc.BaudRate =          baudrate;
597: tc.fBinary =          TRUE;
598: tc.fParity =          FALSE;
599: tc.fOutxCtsFlow =      FALSE;
600: tc.fOutxDsrFlow =      FALSE;
601: tc.fDtrControl =       DTR_CONTROL_DISABLE;
602: tc.fDsrSensitivity =    FALSE;
603: tc.fTXContinueOnXoff =  FALSE;
604: tc.fOutX =             FALSE;
605: tc.fInX =              FALSE;
606: tc.fErrorChar =        FALSE;
607: tc.fNull =             FALSE;
608: tc.fRtsControl =       RTS_CONTROL_DISABLE;
609: tc.fAbortOnError =     FALSE;
610: tc.ByteSize =          8;
611: tc.Parity =            NOPARITY;
612: tc.StopBits =          ONESTOPBIT;
613: SetCommState(xbee.tty, &tc);
614:
615: timeouts.ReadIntervalTimeout = MAXDWORD;
616: timeouts.ReadTotalTimeoutMultiplier = 0;
617: timeouts.ReadTotalTimeoutConstant = 0;
618: timeouts.WriteTotalTimeoutMultiplier = 0;
619: timeouts.WriteTotalTimeoutConstant = 0;
620: SetCommTimeouts(xbee.tty, &timeouts);
621:
622: GetCommMask(xbee.tty, &evtMask);
623: evtMask |= EV_RXCHAR;
624: SetCommMask(xbee.tty, evtMask);
625: #endif /* ----- */
626:
627: /* when xbee_end() is called, if this is not 2 then ATAP will be set to this value */
628: xbee.oldAPI = 2;
629: xbee.cmdSeq = cmdSeq;
630: xbee.cmdTime = cmdTime;
631: if (xbee.cmdSeq && xbee.cmdTime) {
632:     if (xbee_startAPI()) {
633:         if (xbee.log) {
634:             xbee_log("Couldn't communicate with XBee...");
635:         }
636:         xbee_mutex_destroy(xbee.conmutex);
637:         xbee_mutex_destroy(xbee.pktmutex);
638:         xbee_mutex_destroy(xbee.sendmutex);
639:         Xfree(xbee.path);
640: #ifdef __GNUC__ /* ---- */
641:         close(xbee.ttyfd);
642: #endif /* ----- */
643:         fclose(xbee.tty);
644:         return -1;
645:     }
646: }
647:
648: /* allow the listen thread to start */
649: xbee_ready = -1;
650:
651: /* can start xbee_listen thread now */
652: if (xbee_thread_create(xbee.listent, xbee_listen_wrapper, info)) {
653:     perror("xbee_setup(): xbee_thread_create()");
654:     xbee_mutex_destroy(xbee.conmutex);
655:     xbee_mutex_destroy(xbee.pktmutex);
656:     xbee_mutex_destroy(xbee.sendmutex);
657:     Xfree(xbee.path);
658: #ifdef __GNUC__ /* ---- */
659:     close(xbee.ttyfd);
660: #endif /* ----- */
661:     fclose(xbee.tty);
662:     return -1;
663: }
664:
665: usleep(100);
666: while (xbee_ready != -2) {
667:     usleep(100);
668:     if (xbee.log) {
669:         xbee_log("Waiting for xbee_listen() to be ready...");
670:     }
671: }
672:
673: /* allow other functions to be used! */
674: xbee_ready = 1;
675:
676: if (xbee.log) fprintf(xbee.log, "libxbee: Started!\n");
677:
678: return 0;
679: }
680:

```



```

681:  /* #####
682:    xbee_con
683:    produces a connection to the specified device and frameID
684:    if a connection had already been made, then this connection will be returned */
685: xbee_con *xbee_newcon(unsigned char frameID, xbee_types type, ...) {
686:     xbee_con *con, *ocon;
687:     unsigned char tAddr[8];
688:     va_list ap;
689:     int t;
690:     int i;
691:
692:     ISREADY;
693:
694:     if (!type || type == xbee_unknown) type = xbee_localAT; /* default to local AT */
695:     else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */
696:
697:     va_start(ap, type);
698:     /* if: 64 bit address expected (2 ints) */
699:     if ((type == xbee_64bitRemoteAT) ||
700:         (type == xbee_64bitData) ||
701:         (type == xbee_64bitIO)) {
702:         t = va_arg(ap, int);
703:         tAddr[0] = (t >> 24) & 0xFF;
704:         tAddr[1] = (t >> 16) & 0xFF;
705:         tAddr[2] = (t >> 8) & 0xFF;
706:         tAddr[3] = (t >> 0) & 0xFF;
707:         t = va_arg(ap, int);
708:         tAddr[4] = (t >> 24) & 0xFF;
709:         tAddr[5] = (t >> 16) & 0xFF;
710:         tAddr[6] = (t >> 8) & 0xFF;
711:         tAddr[7] = (t >> 0) & 0xFF;
712:
713:         /* if: 16 bit address expected (1 int) */
714:     } else if ((type == xbee_16bitRemoteAT) ||
715:               (type == xbee_16bitData) ||
716:               (type == xbee_16bitIO)) {
717:         t = va_arg(ap, int);
718:         tAddr[0] = (t >> 8) & 0xFF;
719:         tAddr[1] = (t >> 0) & 0xFF;
720:         tAddr[2] = 0;
721:         tAddr[3] = 0;
722:         tAddr[4] = 0;
723:         tAddr[5] = 0;
724:         tAddr[6] = 0;
725:         tAddr[7] = 0;
726:
727:         /* otherwise clear the address */
728:     } else {
729:         memset(tAddr, 0, 8);
730:     }
731:     va_end(ap);
732:
733:     /* lock the connection mutex */
734:     xbee_mutex_lock(xbee.conmutex);
735:
736:     /* are there any connections? */
737:     if (xbee.conlist) {
738:         con = xbee.conlist;
739:         while (con) {
740:             /* if: after a modemStatus, and the types match! */
741:             if ((type == xbee_modemStatus) &&
742:                 (con->type == type)) {
743:                 xbee_mutex_unlock(xbee.conmutex);
744:                 return con;
745:
746:             /* if: after a txStatus and frameIDs match! */
747:             } else if ((type == xbee_txStatus) &&
748:                       (con->type == type) &&
749:                       (frameID == con->frameID)) {
750:                 xbee_mutex_unlock(xbee.conmutex);
751:                 return con;
752:
753:             /* if: after a localAT, and the frameIDs match! */
754:             } else if ((type == xbee_localAT) &&
755:                       (con->type == type) &&
756:                       (frameID == con->frameID)) {
757:                 xbee_mutex_unlock(xbee.conmutex);
758:                 return con;
759:
760:             /* if: connection types match, the frameIDs match, and the addresses match! */
761:             } else if ((type == con->type) &&
762:                       (frameID == con->frameID) &&
763:                       (!memcmp(tAddr, con->tAddr, 8))) {
764:                 xbee_mutex_unlock(xbee.conmutex);
765:                 return con;

```

```

766:     }
767:
768:     /* if there are more, move along, dont want to loose that last item! */
769:     if (con->next == NULL) break;
770:     con = con->next;
771: }
772:
773: /* keep hold of the last connection... we will need to link it up later */
774: ocon = con;
775: }
776:
777: /* create a new connection and set its attributes */
778: con = Xcalloc(sizeof(xbee_con));
779: con->type = type;
780: /* is it a 64bit connection? */
781: if ((type == xbee_64bitRemoteAT) ||
782:     (type == xbee_64bitData) ||
783:     (type == xbee_64bitIO)) {
784:     con->tAddr64 = TRUE;
785: }
786: con->atQueue = 0; /* queue AT commands? */
787: con->txDisableACK = 0; /* disable ACKs? */
788: con->txBroadcast = 0; /* broadcast? */
789: con->frameID = frameID;
790: memcpy(con->tAddr, tAddr, 8); /* copy in the remote address */
791:
792: if (xbee.log) {
793:     switch(type) {
794:     case xbee_localAT:
795:         xbee_log("New local AT connection!");
796:         break;
797:     case xbee_16bitRemoteAT:
798:     case xbee_64bitRemoteAT:
799:         xbee_logc("New %d-bit remote AT connection! (to: ", (con->tAddr64?64:16));
800:         for (i=0; i<(con->tAddr64?8:2); i++) {
801:             fprintf(xbee.log, (i?":%02X": "%02X"), tAddr[i]);
802:         }
803:         fprintf(xbee.log, ")\n");
804:         break;
805:     case xbee_16bitData:
806:     case xbee_64bitData:
807:         xbee_logc("New %d-bit data connection! (to: ", (con->tAddr64?64:16));
808:         for (i=0; i<(con->tAddr64?8:2); i++) {
809:             fprintf(xbee.log, (i?":%02X": "%02X"), tAddr[i]);
810:         }
811:         fprintf(xbee.log, ")\n");
812:         break;
813:     case xbee_16bitIO:
814:     case xbee_64bitIO:
815:         xbee_logc("New %d-bit IO connection! (to: ", (con->tAddr64?64:16));
816:         for (i=0; i<(con->tAddr64?8:2); i++) {
817:             fprintf(xbee.log, (i?":%02X": "%02X"), tAddr[i]);
818:         }
819:         fprintf(xbee.log, ")\n");
820:         break;
821:     case xbee_txStatus:
822:         xbee_log("New Tx status connection!");
823:         break;
824:     case xbee_modemStatus:
825:         xbee_log("New modem status connection!");
826:         break;
827:     case xbee_unknown:
828:     default:
829:         xbee_log("New unknown connection!");
830:     }
831: }
832:
833: /* make it the last in the list */
834: con->next = NULL;
835: /* add it to the list */
836: if (xbee.conlist) {
837:     ocon->next = con;
838: } else {
839:     xbee.conlist = con;
840: }
841:
842: /* unlock the mutex */
843: xbee_mutex_unlock(xbee.conmutex);
844: return con;
845: }
846:
847: /* #####
848: xbee_conflush
849: removes any packets that have been collected for the specified
850: connection */

```

```

851: void xbee_flushcon(xbee_con *con) {
852:     xbee_pkt *r, *p, *n;
853:
854:     /* lock the packet mutex */
855:     xbee_mutex_lock(xbee.pktmutex);
856:
857:     /* if: there are packets */
858:     if ((p = xbee.pktlist) != NULL) {
859:         r = NULL;
860:         /* get all packets for this connection */
861:         do {
862:             /* does the packet match the connection? */
863:             if (xbee_matchpktcon(p,con)) {
864:                 /* if it was the first packet */
865:                 if (!r) {
866:                     /* move the chain along */
867:                     xbee.pktlist = p->next;
868:                 } else {
869:                     /* otherwise relink the list */
870:                     r->next = p->next;
871:                 }
872:                 xbee.pktcount--;
873:
874:                 /* free this packet! */
875:                 n = p->next;
876:                 Xfree(p);
877:                 /* move on */
878:                 p = n;
879:             } else {
880:                 /* move on */
881:                 r = p;
882:                 p = p->next;
883:             }
884:         } while (p);
885:         xbee.pktlast = r;
886:     }
887:
888:     /* unlock the packet mutex */
889:     xbee_mutex_unlock(xbee.pktmutex);
890: }
891:
892: /* #####
893: xbee_endcon
894: close the unwanted connection
895: free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
896: void xbee_endcon2(xbee_con **con) {
897:     xbee_con *t, *u;
898:
899:     /* lock the connection mutex */
900:     xbee_mutex_lock(xbee.conmutex);
901:
902:     u = t = xbee.conlist;
903:     while (t && t != *con) {
904:         u = t;
905:         t = t->next;
906:     }
907:     if (!t) {
908:         /* invalid connection given... */
909:         if (xbee.log) {
910:             xbee_log("Attempted to close invalid connection...");
911:         }
912:         /* unlock the connection mutex */
913:         xbee_mutex_unlock(xbee.conmutex);
914:         return;
915:     }
916:     /* extract this connection from the list */
917:     u->next = (*con)->next;
918:     if (*con == xbee.conlist) xbee.conlist = NULL;
919:
920:     /* unlock the connection mutex */
921:     xbee_mutex_unlock(xbee.conmutex);
922:
923:     /* remove all packets for this connection */
924:     xbee_flushcon(*con);
925:
926:     /* free the connection! */
927:     Xfree(*con);
928: }
929:
930: /* #####
931: xbee_senddata
932: send the specified data to the provided connection */
933: int xbee_senddata(xbee_con *con, char *format, ...) {
934:     int ret;
935:     va_list ap;

```

```

936:
937:     ISREADY;
938:
939:     /* xbee_vsnddata() wants a va_list... */
940:     va_start(ap, format);
941:     /* hand it over :) */
942:     ret = xbee_vsnddata(con,format,ap);
943:     va_end(ap);
944:     return ret;
945: }
946:
947: int xbee_vsnddata(xbee_con *con, char *format, va_list ap) {
948:     unsigned char data[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
949:     int length;
950:
951:     ISREADY;
952:
953:     /* make up the data and keep the length, its possible there are nulls in there */
954:     length = vsnprintf((char *)data,128,format,ap);
955:
956:     /* hand it over :) */
957:     return xbee_nsnddata(con,(char *)data,length);
958: }
959:
960: int xbee_nsnddata(xbee_con *con, char *data, int length) {
961:     t_data *pkt;
962:     int i;
963:     unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
964:
965:     ISREADY;
966:
967:     if (!con) return -1;
968:     if (con->type == xbee_unknown) return -1;
969:     if (length > 127) return -1;
970:
971:
972:     if (xbee.log) {
973:         xbee_log("==== TX Packet =====");
974:         xbee_logc("Connection Type: ");
975:         switch (con->type) {
976:             case xbee_unknown:      fprintf(xbee.log,"Unknown\n"); break;
977:             case xbee_localAT:      fprintf(xbee.log,"Local AT\n"); break;
978:             case xbee_remoteAT:     fprintf(xbee.log,"Remote AT\n"); break;
979:             case xbee_16bitRemoteAT: fprintf(xbee.log,"Remote AT (16-bit)\n"); break;
980:             case xbee_64bitRemoteAT: fprintf(xbee.log,"Remote AT (64-bit)\n"); break;
981:             case xbee_16bitData:    fprintf(xbee.log,"Data (16-bit)\n"); break;
982:             case xbee_64bitData:    fprintf(xbee.log,"Data (64-bit)\n"); break;
983:             case xbee_16bitIO:      fprintf(xbee.log,"IO (16-bit)\n"); break;
984:             case xbee_64bitIO:      fprintf(xbee.log,"IO (64-bit)\n"); break;
985:             case xbee_txStatus:     fprintf(xbee.log,"Tx Status\n"); break;
986:             case xbee_modemStatus:  fprintf(xbee.log,"Modem Status\n"); break;
987:         }
988:         xbee_logc("Destination: ");
989:         for (i=0;i<(con->tAddr64?8:2);i++) {
990:             fprintf(xbee.log,(i?"%02X":"%02X"),con->tAddr[i]);
991:         }
992:         fprintf(xbee.log,"\n");
993:         xbee_log("Length: %d",length);
994:         for (i=0;i<length;i++) {
995:             xbee_logc("%3d | 0x%02X ",i,data[i]);
996:             if ((data[i] > 32) && (data[i] < 127)) {
997:                 fprintf(xbee.log,"%c'\n",data[i]);
998:             } else{
999:                 fprintf(xbee.log," _\n");
1000:             }
1001:         }
1002:     }
1003:
1004:     /* ##### */
1005:     /* if: local AT */
1006:     if (con->type == xbee_localAT) {
1007:         /* AT commands are 2 chars long (plus optional parameter) */
1008:         if (length < 2) return -1;
1009:
1010:         /* use the command? */
1011:         buf[0] = ((!con->atQueue)?0x08:0x09);
1012:         buf[1] = con->frameID;
1013:
1014:         /* copy in the data */
1015:         for (i=0;i<length;i++) {
1016:             buf[i+2] = data[i];
1017:         }
1018:
1019:         /* setup the packet */
1020:         pkt = xbee_make_pkt(buf,i+2);

```

```
1021:     /* send it on */
1022:     xbee_send_pkt(pkt);
1023:
1024:     return 0;
1025:
1026:     /* ##### */
1027:     /* if: remote AT */
1028: } else if ((con->type == xbee_16bitRemoteAT) ||
1029:           (con->type == xbee_64bitRemoteAT)) {
1030:     if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter) */
1031:     buf[0] = 0x17;
1032:     buf[1] = con->frameID;
1033:
1034:     /* copy in the relevant address */
1035:     if (con->tAddr64) {
1036:         memcpy(&buf[2], con->tAddr, 8);
1037:         buf[10] = 0xFF;
1038:         buf[11] = 0xFE;
1039:     } else {
1040:         memset(&buf[2], 0, 8);
1041:         memcpy(&buf[10], con->tAddr, 2);
1042:     }
1043:     /* queue the command? */
1044:     buf[12] = ((!con->atQueue)?0x02:0x00);
1045:
1046:     /* copy in the data */
1047:     for (i=0; i<length; i++) {
1048:         buf[i+13] = data[i];
1049:     }
1050:
1051:     /* setup the packet */
1052:     pkt = xbee_make_pkt(buf, i+13);
1053:     /* send it on */
1054:     xbee_send_pkt(pkt);
1055:
1056:     return 0;
1057:
1058:     /* ##### */
1059:     /* if: 16 or 64bit Data */
1060: } else if ((con->type == xbee_16bitData) ||
1061:           (con->type == xbee_64bitData)) {
1062:     int offset;
1063:
1064:     /* if: 16bit Data */
1065:     if (con->type == xbee_16bitData) {
1066:         buf[0] = 0x01;
1067:         offset = 5;
1068:         /* copy in the address */
1069:         memcpy(&buf[2], con->tAddr, 2);
1070:
1071:         /* if: 64bit Data */
1072:     } else { /* 64bit Data */
1073:         buf[0] = 0x00;
1074:         offset = 11;
1075:         /* copy in the address */
1076:         memcpy(&buf[2], con->tAddr, 8);
1077:     }
1078:
1079:     /* copy frameID */
1080:     buf[1] = con->frameID;
1081:
1082:     /* disable ack? broadcast? */
1083:     buf[offset-1] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
1084:
1085:     /* copy in the data */
1086:     for (i=0; i<length; i++) {
1087:         buf[i+offset] = data[i];
1088:     }
1089:
1090:     /* setup the packet */
1091:     pkt = xbee_make_pkt(buf, i+offset);
1092:     /* send it on */
1093:     xbee_send_pkt(pkt);
1094:
1095:     return 0;
1096:
1097:     /* ##### */
1098:     /* if: I/O */
1099: } else if ((con->type == xbee_64bitIO) ||
1100:           (con->type == xbee_16bitIO)) {
1101:     /* not currently implemented... is it even allowed? */
1102:     if (xbee.log) {
1103:         fprintf(xbee.log, "***** TODO *****\n");
1104:     }
1105: }
```

```

1106:
1107:     return -2;
1108: }
1109:
1110: /* #####
1111: xbee_getpacket
1112: retrieves the next packet destined for the given connection
1113: once the packet has been retrieved, it is removed for the list! */
1114: xbee_pkt *xbee_getpacketwait(xbee_con *con) {
1115:     xbee_pkt *p;
1116:     int i;
1117:
1118:     /* 50ms * 20 = 1 second */
1119:     for (i = 0; i < 20; i++) {
1120:         p = xbee_getpacket(con);
1121:         if (p) break;
1122:         usleep(50000); /* 50ms */
1123:     }
1124:
1125:     return p;
1126: }
1127: xbee_pkt *xbee_getpacket(xbee_con *con) {
1128:     xbee_pkt *l, *p, *q;
1129:     /*if (xbee.log) {
1130:         xbee_log("==== Get Packet =====");
1131:     }*/
1132:
1133:     /* lock the packet mutex */
1134:     xbee_mutex_lock(xbee.pktmutex);
1135:
1136:     /* if: there are no packets */
1137:     if ((p = xbee.pktlist) == NULL) {
1138:         xbee_mutex_unlock(xbee.pktmutex);
1139:         /*if (xbee.log) {
1140:             xbee_log("No packets available...");
1141:         }*/
1142:         return NULL;
1143:     }
1144:
1145:     l = NULL;
1146:     q = NULL;
1147:     /* get the first available packet for this connection */
1148:     do {
1149:         /* does the packet match the connection? */
1150:         if (xbee_matchpktcon(p, con)) {
1151:             q = p;
1152:             break;
1153:         }
1154:         /* move on */
1155:         l = p;
1156:         p = p->next;
1157:     } while (p);
1158:
1159:     /* if: no packet was found */
1160:     if (!q) {
1161:         xbee_mutex_unlock(xbee.pktmutex);
1162:         /*if (xbee.log) {
1163:             xbee_log("No packets available (for connection)...");
1164:         }*/
1165:         return NULL;
1166:     }
1167:
1168:     /* if it was the first packet */
1169:     if (l) {
1170:         /* relink the list */
1171:         l->next = p->next;
1172:         if (!l->next) xbee.pktlast = l;
1173:     } else {
1174:         /* move the chain along */
1175:         xbee.pktlist = p->next;
1176:         if (!xbee.pktlist) {
1177:             xbee.pktlast = NULL;
1178:         } else if (!xbee.pktlist->next) {
1179:             xbee.pktlast = xbee.pktlist;
1180:         }
1181:     }
1182:     xbee.pktcount--;
1183:
1184:     /* unlink this packet from the chain! */
1185:     q->next = NULL;
1186:
1187:     if (xbee.log) {
1188:         xbee_log("==== Get Packet =====");
1189:         xbee_log("Got a packet");
1190:         xbee_log("Packets left: %d", xbee.pktcount);

```

```

1191: }
1192:
1193:  /* unlock the packet mutex */
1194:  xbee_mutex_unlock(xbee.pktmutex);
1195:
1196:  /* and return the packet (must be free'd by caller!) */
1197:  return q;
1198: }
1199:
1200: /* #####
1201:     xbee_matchpktcon - INTERNAL
1202:     checks if the packet matches the connection */
1203: static int xbee_matchpktcon(xbee_pkt *pkt, xbee_con *con) {
1204:  /* if: the connection type matches the packet type OR
1205:      the connection is 16/64bit remote AT, and the packet is a remote AT response */
1206:  if ((pkt->type == con->type) || /* -- */
1207:      ((pkt->type == xbee_remoteAT) && /* -- */
1208:       ((con->type == xbee_16bitRemoteAT) ||
1209:        (con->type == xbee_64bitRemoteAT)))) {
1210:
1211:      /* if: the packet is modem status OR
1212:          the packet is tx status or AT data and the frame IDs match OR
1213:          the addresses match */
1214:      if (pkt->type == xbee_modemStatus) return 1;
1215:
1216:      if ((pkt->type == xbee_txStatus) ||
1217:          (pkt->type == xbee_localAT) ||
1218:          (pkt->type == xbee_remoteAT)) {
1219:          if (pkt->frameID == con->frameID) {
1220:              return 1;
1221:          }
1222:      } else if (pkt->sAddr64 && !memcmp(pkt->Addr64, con->tAddr, 8)) {
1223:          return 1;
1224:      } else if (!pkt->sAddr64 && !memcmp(pkt->Addr16, con->tAddr, 2)) {
1225:          return 1;
1226:      }
1227:  }
1228:  return 0;
1229: }
1230:
1231: /* #####
1232:     xbee_parse_io - INTERNAL
1233:     parses the data given into the packet io information */
1234: static int xbee_parse_io(xbee_pkt *p, unsigned char *d, int maskOffset, int sampleOffset, int sample) {
1235:  xbee_sample *s = &(p->IOdata[sample]);
1236:
1237:  /* copy in the I/O data mask */
1238:  s->IOmask = (((d[maskOffset]<<8) | d[maskOffset + 1]) & 0x7FFF);
1239:
1240:  /* copy in the digital I/O data */
1241:  s->IOdigital = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x01FF);
1242:
1243:  /* advance over the digital data, if its there */
1244:  sampleOffset += ((s->IOmask & 0x01FF)?2:0);
1245:
1246:  /* copy in the analog I/O data */
1247:  if (s->IOmask & 0x0200) {
1248:      s->IOanalog[0] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1249:      sampleOffset+=2;
1250:  }
1251:  if (s->IOmask & 0x0400) {
1252:      s->IOanalog[1] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1253:      sampleOffset+=2;
1254:  }
1255:  if (s->IOmask & 0x0800) {
1256:      s->IOanalog[2] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1257:      sampleOffset+=2;
1258:  }
1259:  if (s->IOmask & 0x1000) {
1260:      s->IOanalog[3] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1261:      sampleOffset+=2;
1262:  }
1263:  if (s->IOmask & 0x2000) {
1264:      s->IOanalog[4] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1265:      sampleOffset+=2;
1266:  }
1267:  if (s->IOmask & 0x4000) {
1268:      s->IOanalog[5] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1269:      sampleOffset+=2;
1270:  }
1271:
1272:  if (xbee.log) {
1273:      if (s->IOmask & 0x0001)
1274:          xbee_log("Digital 0: %c", ((s->IOdigital & 0x0001)?'1':'0'));
1275:      if (s->IOmask & 0x0002)

```



```

1276:     xbee_log("Digital 1: %c",((s->IOdigital & 0x0002)?'1':'0'));
1277:     if (s->IOmask & 0x0004)
1278:         xbee_log("Digital 2: %c",((s->IOdigital & 0x0004)?'1':'0'));
1279:     if (s->IOmask & 0x0008)
1280:         xbee_log("Digital 3: %c",((s->IOdigital & 0x0008)?'1':'0'));
1281:     if (s->IOmask & 0x0010)
1282:         xbee_log("Digital 4: %c",((s->IOdigital & 0x0010)?'1':'0'));
1283:     if (s->IOmask & 0x0020)
1284:         xbee_log("Digital 5: %c",((s->IOdigital & 0x0020)?'1':'0'));
1285:     if (s->IOmask & 0x0040)
1286:         xbee_log("Digital 6: %c",((s->IOdigital & 0x0040)?'1':'0'));
1287:     if (s->IOmask & 0x0080)
1288:         xbee_log("Digital 7: %c",((s->IOdigital & 0x0080)?'1':'0'));
1289:     if (s->IOmask & 0x0100)
1290:         xbee_log("Digital 8: %c",((s->IOdigital & 0x0100)?'1':'0'));
1291:     if (s->IOmask & 0x0200)
1292:         xbee_log("Analog 0: %d (~%.2fv)\n",s->IOanalog[0],(3.3/1023)*s->IOanalog[0]);
1293:     if (s->IOmask & 0x0400)
1294:         xbee_log("Analog 1: %d (~%.2fv)\n",s->IOanalog[1],(3.3/1023)*s->IOanalog[1]);
1295:     if (s->IOmask & 0x0800)
1296:         xbee_log("Analog 2: %d (~%.2fv)\n",s->IOanalog[2],(3.3/1023)*s->IOanalog[2]);
1297:     if (s->IOmask & 0x1000)
1298:         xbee_log("Analog 3: %d (~%.2fv)\n",s->IOanalog[3],(3.3/1023)*s->IOanalog[3]);
1299:     if (s->IOmask & 0x2000)
1300:         xbee_log("Analog 4: %d (~%.2fv)\n",s->IOanalog[4],(3.3/1023)*s->IOanalog[4]);
1301:     if (s->IOmask & 0x4000)
1302:         xbee_log("Analog 5: %d (~%.2fv)\n",s->IOanalog[5],(3.3/1023)*s->IOanalog[5]);
1303: }
1304:
1305: return sampleOffset;
1306: }
1307:
1308: /* #####
1309: xbee_listen_stop
1310: stops the listen thread after the current packet has been processed */
1311: void xbee_listen_stop(void) {
1312:     xbee.listenrun = 0;
1313: }
1314:
1315: /* #####
1316: xbee_listen_wrapper - INTERNAL
1317: the xbee_listen wrapper. Prints an error when xbee_listen ends */
1318: static void xbee_listen_wrapper(t_info *info) {
1319:     int ret;
1320:     /* just falls out if the proper 'go-ahead' isn't given */
1321:     if (xbee_ready != -1) return;
1322:     /* now allow the parent to continue */
1323:     xbee_ready = -2;
1324:
1325: #ifdef _WIN32 /* ---- */
1326:     /* win32 requires this delay... no idea why */
1327:     usleep(1000000);
1328: #endif /* ----- */
1329:
1330:     while (xbee.listenrun) {
1331:         info->i = -1;
1332:         ret = xbee_listen(info);
1333:         if (!xbee.listenrun) break;
1334:         if (xbee.log) {
1335:             xbee_log("xbee_listen() returned [%d]... Restarting in 250ms!",ret);
1336:         }
1337:         usleep(25000);
1338:     }
1339: }
1340:
1341: /* xbee_listen - INTERNAL
1342: the xbee xbee_listen thread
1343: reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
1344: static int xbee_listen(t_info *info) {
1345:     unsigned char c, t, d[1024];
1346:     unsigned int l, i, chksum, o;
1347:     int j;
1348:     xbee_pkt *p, *q;
1349:     xbee_con *con;
1350:     int hasCon;
1351:
1352:     /* just falls out if the proper 'go-ahead' isn't given */
1353:     if (info->i != -1) return -1;
1354:     /* do this forever :) */
1355:     while (xbee.listenrun) {
1356:         /* wait for a valid start byte */
1357:         if (xbee_getrawbyte() != 0x7E) continue;
1358:         if (!xbee.listenrun) return 0;
1359:
1360:         if (xbee.log) {

```

```

1361:         xbee_log("---== RX Packet =====");
1362:         xbee_log("Got a packet!...");
1363:     }
1364:
1365:     /* get the length */
1366:     l = xbee_getbyte() << 8;
1367:     l += xbee_getbyte();
1368:
1369:     /* check it is a valid length... */
1370:     if (!l) {
1371:         if (xbee.log) {
1372:             xbee_log("Recived zero length packet!");
1373:         }
1374:         continue;
1375:     }
1376:     if (l > 100) {
1377:         if (xbee.log) {
1378:             xbee_log("Recived oversized packet! Length: %d", l - 1);
1379:         }
1380:     }
1381:     if (l > sizeof(d) - 1) {
1382:         if (xbee.log) {
1383:             xbee_log("Recived packet larger than buffer! Discarding...");
1384:         }
1385:         continue;
1386:     }
1387:
1388:     if (xbee.log) {
1389:         xbee_log("Length: %d", l - 1);
1390:     }
1391:
1392:     /* get the packet type */
1393:     t = xbee_getbyte();
1394:
1395:     /* start the checksum */
1396:     chksum = t;
1397:
1398:     /* suck in all the data */
1399:     for (i = 0; l > 1 && i < 128; l--, i++) {
1400:         /* get an unescaped byte */
1401:         c = xbee_getbyte();
1402:         d[i] = c;
1403:         chksum += c;
1404:         if (xbee.log) {
1405:             xbee_logc("%3d | 0x%02X | ", i, c);
1406:             if ((c > 32) && (c < 127)) fprintf(xbee.log, "'%c'", c); else fprintf(xbee.log, " _ ");
1407:
1408:             if ((t == 0x80 && i == (8 + 2)) || /* 64-bit Data packet */
1409:                 (t == 0x81 && i == (2 + 2))) { /* 16-bit Data packet */
1410:                 /* mark the beginning of the 'data' bytes */
1411:                 fprintf(xbee.log, " <-- data starts");
1412:             }
1413:
1414:             fprintf(xbee.log, "\n");
1415:         }
1416:     }
1417:     i--; /* it went up too many times!... */
1418:
1419:     /* add the checksum */
1420:     chksum += xbee_getbyte();
1421:
1422:     /* check if the whole packet was recieved, or something else ocured... unlikely... */
1423:     if (l > 1) {
1424:         if (xbee.log) {
1425:             xbee_log("Didn't get whole packet... :(");
1426:         }
1427:         continue;
1428:     }
1429:
1430:     /* check the checksum */
1431:     if ((chksum & 0xFF) != 0xFF) {
1432:         if (xbee.log) {
1433:             xbee_log("Invalid Checksum: 0x%02X", chksum);
1434:         }
1435:         continue;
1436:     }
1437:
1438:     /* make a new packet */
1439:     p = Xcalloc(sizeof(xbee_pkt));
1440:     q = NULL;
1441:     p->datalen = 0;
1442:
1443:     /* ##### */
1444:     /* if: modem status */
1445:     if (t == 0x8A) {

```

```

1446:     if (xbee.log) {
1447:         xbee_log("Packet type: Modem Status (0x8A)");
1448:         xbee_logc("Event: ");
1449:         switch (d[0]) {
1450:             case 0x00: fprintf(xbee.log, "Hardware reset"); break;
1451:             case 0x01: fprintf(xbee.log, "Watchdog timer reset"); break;
1452:             case 0x02: fprintf(xbee.log, "Associated"); break;
1453:             case 0x03: fprintf(xbee.log, "Disassociated"); break;
1454:             case 0x04: fprintf(xbee.log, "Synchronization lost"); break;
1455:             case 0x05: fprintf(xbee.log, "Coordinator realignment"); break;
1456:             case 0x06: fprintf(xbee.log, "Coordinator started"); break;
1457:         }
1458:         fprintf(xbee.log, "... (0x%02X)\n", d[0]);
1459:     }
1460:     p->type = xbee_modemStatus;
1461:
1462:     p->sAddr64 = FALSE;
1463:     p->dataPkt = FALSE;
1464:     p->txStatusPkt = FALSE;
1465:     p->modemStatusPkt = TRUE;
1466:     p->remoteATPkt = FALSE;
1467:     p->IOPkt = FALSE;
1468:
1469:     /* modem status can only ever give 1 'data' byte */
1470:     p->datalen = 1;
1471:     p->data[0] = d[0];
1472:
1473:     /* ##### */
1474:     /* if: local AT response */
1475: } else if (t == 0x88) {
1476:     if (xbee.log) {
1477:         xbee_log("Packet type: Local AT Response (0x88)");
1478:         xbee_log("FrameID: 0x%02X", d[0]);
1479:         xbee_log("AT Command: %c%c", d[1], d[2]);
1480:         xbee_logc("Status: ");
1481:         if (d[3] == 0) fprintf(xbee.log, "OK");
1482:         else if (d[3] == 1) fprintf(xbee.log, "Error");
1483:         else if (d[3] == 2) fprintf(xbee.log, "Invalid Command");
1484:         else if (d[3] == 3) fprintf(xbee.log, "Invalid Parameter");
1485:         fprintf(xbee.log, " (0x%02X)\n", d[3]);
1486:     }
1487:     p->type = xbee_localAT;
1488:
1489:     p->sAddr64 = FALSE;
1490:     p->dataPkt = FALSE;
1491:     p->txStatusPkt = FALSE;
1492:     p->modemStatusPkt = FALSE;
1493:     p->remoteATPkt = FALSE;
1494:     p->IOPkt = FALSE;
1495:
1496:     p->frameID = d[0];
1497:     p->atCmd[0] = d[1];
1498:     p->atCmd[1] = d[2];
1499:
1500:     p->status = d[3];
1501:
1502:     /* copy in the data */
1503:     p->datalen = i-3;
1504:     for (; i>3; i--) p->data[i-4] = d[i];
1505:
1506:     /* ##### */
1507:     /* if: remote AT response */
1508: } else if (t == 0x97) {
1509:     if (xbee.log) {
1510:         xbee_log("Packet type: Remote AT Response (0x97)");
1511:         xbee_log("FrameID: 0x%02X", d[0]);
1512:         xbee_logc("64-bit Address: ");
1513:         for (j=0; j<8; j++) {
1514:             fprintf(xbee.log, (j?"%02X":"%02X"), d[1+j]);
1515:         }
1516:         fprintf(xbee.log, "\n");
1517:         xbee_logc("16-bit Address: ");
1518:         for (j=0; j<2; j++) {
1519:             fprintf(xbee.log, (j?"%02X":"%02X"), d[9+j]);
1520:         }
1521:         fprintf(xbee.log, "\n");
1522:         xbee_log("AT Command: %c%c", d[11], d[12]);
1523:         xbee_logc("Status: ");
1524:         if (d[13] == 0) fprintf(xbee.log, "OK");
1525:         else if (d[13] == 1) fprintf(xbee.log, "Error");
1526:         else if (d[13] == 2) fprintf(xbee.log, "Invalid Command");
1527:         else if (d[13] == 3) fprintf(xbee.log, "Invalid Parameter");
1528:         else if (d[13] == 4) fprintf(xbee.log, "No Response");
1529:         fprintf(xbee.log, " (0x%02X)\n", d[13]);
1530:     }

```

```

1531:     p->type = xbee_remoteAT;
1532:
1533:     p->sAddr64 = FALSE;
1534:     p->dataPkt = FALSE;
1535:     p->txStatusPkt = FALSE;
1536:     p->modemStatusPkt = FALSE;
1537:     p->remoteATPkt = TRUE;
1538:     p->IOPkt = FALSE;
1539:
1540:     p->frameID = d[0];
1541:
1542:     p->Addr64[0] = d[1];
1543:     p->Addr64[1] = d[2];
1544:     p->Addr64[2] = d[3];
1545:     p->Addr64[3] = d[4];
1546:     p->Addr64[4] = d[5];
1547:     p->Addr64[5] = d[6];
1548:     p->Addr64[6] = d[7];
1549:     p->Addr64[7] = d[8];
1550:
1551:     p->Addr16[0] = d[9];
1552:     p->Addr16[1] = d[10];
1553:
1554:     p->atCmd[0] = d[11];
1555:     p->atCmd[1] = d[12];
1556:
1557:     p->status = d[13];
1558:
1559:     p->samples = 1;
1560:
1561:     if (p->status == 0x00 && p->atCmd[0] == 'I' && p->atCmd[1] == 'S') {
1562:         /* parse the io data */
1563:         if (xbee.log) xbee_log("---- Sample -----");
1564:         xbee_parse_io(p, d, 15, 17, 0);
1565:         if (xbee.log) xbee_log("-----");
1566:     } else {
1567:         /* copy in the data */
1568:         p->datalen = i-13;
1569:         for (;i>13;i--) p->data[i-14] = d[i];
1570:     }
1571:
1572:     /* ##### */
1573:     /* if: TX status */
1574: } else if (t == 0x89) {
1575:     if (xbee.log) {
1576:         xbee_log("Packet type: TX Status Report (0x89)");
1577:         xbee_log("FrameID: 0x%02X",d[0]);
1578:         xbee_logc("Status: ");
1579:         if (d[1] == 0) fprintf(xbee.log,"Success");
1580:         else if (d[1] == 1) fprintf(xbee.log,"No ACK");
1581:         else if (d[1] == 2) fprintf(xbee.log,"CCA Failure");
1582:         else if (d[1] == 3) fprintf(xbee.log,"Purged");
1583:         fprintf(xbee.log," (0x%02X)\n",d[1]);
1584:     }
1585:     p->type = xbee_txStatus;
1586:
1587:     p->sAddr64 = FALSE;
1588:     p->dataPkt = FALSE;
1589:     p->txStatusPkt = TRUE;
1590:     p->modemStatusPkt = FALSE;
1591:     p->remoteATPkt = FALSE;
1592:     p->IOPkt = FALSE;
1593:
1594:     p->frameID = d[0];
1595:
1596:     p->status = d[1];
1597:
1598:     /* never returns data */
1599:     p->datalen = 0;
1600:
1601:     /* ##### */
1602:     /* if: 16 / 64bit data recieve */
1603: } else if ((t == 0x80) ||
1604:           (t == 0x81)) {
1605:     int offset;
1606:     if (t == 0x80) { /* 64bit */
1607:         offset = 8;
1608:     } else { /* 16bit */
1609:         offset = 2;
1610:     }
1611:     if (xbee.log) {
1612:         xbee_log("Packet type: %d-bit RX Data (0x%02X)",((t == 0x80)?64:16),t);
1613:         xbee_logc("%d-bit Address: ",((t == 0x80)?64:16));
1614:         for (j=0;j<offset;j++) {
1615:             fprintf(xbee.log,(j?"%02X":"%02X"),d[j]);

```

```

1616:     }
1617:     fprintf(xbee.log, "\n");
1618:     xbee_log("RSSI: -%ddb", d[offset]);
1619:     if (d[offset + 1] & 0x02) xbee_log("Options: Address Broadcast");
1620:     if (d[offset + 1] & 0x03) xbee_log("Options: PAN Broadcast");
1621: }
1622: p->dataPkt = TRUE;
1623: p->txStatusPkt = FALSE;
1624: p->modemStatusPkt = FALSE;
1625: p->remoteATPkt = FALSE;
1626: p->IOPkt = FALSE;
1627:
1628: if (t == 0x80) { /* 64bit */
1629:     p->type = xbee_64bitData;
1630:
1631:     p->sAddr64 = TRUE;
1632:
1633:     p->Addr64[0] = d[0];
1634:     p->Addr64[1] = d[1];
1635:     p->Addr64[2] = d[2];
1636:     p->Addr64[3] = d[3];
1637:     p->Addr64[4] = d[4];
1638:     p->Addr64[5] = d[5];
1639:     p->Addr64[6] = d[6];
1640:     p->Addr64[7] = d[7];
1641: } else { /* 16bit */
1642:     p->type = xbee_16bitData;
1643:
1644:     p->sAddr64 = FALSE;
1645:
1646:     p->Addr16[0] = d[0];
1647:     p->Addr16[1] = d[1];
1648: }
1649:
1650: /* save the RSSI / signal strength
1651:    this can be used with printf as:
1652:    printf("-%ddb\n", p->RSSI); */
1653: p->RSSI = d[offset];
1654:
1655: p->status = d[offset + 1];
1656:
1657: /* copy in the data */
1658: p->datalen = i - (offset + 1);
1659: for (; i > offset + 1; i--) p->data[i - (offset + 2)] = d[i];
1660:
1661: /* ##### */
1662: /* if: 16 / 64bit I/O receive */
1663: } else if ((t == 0x82) ||
1664:            (t == 0x83)) {
1665:     int offset;
1666:     if (t == 0x82) { /* 64bit */
1667:         p->type = xbee_64bitIO;
1668:
1669:         p->sAddr64 = TRUE;
1670:
1671:         p->Addr64[0] = d[0];
1672:         p->Addr64[1] = d[1];
1673:         p->Addr64[2] = d[2];
1674:         p->Addr64[3] = d[3];
1675:         p->Addr64[4] = d[4];
1676:         p->Addr64[5] = d[5];
1677:         p->Addr64[6] = d[6];
1678:         p->Addr64[7] = d[7];
1679:
1680:         offset = 8;
1681:         p->samples = d[10];
1682:     } else { /* 16bit */
1683:         p->type = xbee_16bitIO;
1684:
1685:         p->sAddr64 = FALSE;
1686:
1687:         p->Addr16[0] = d[0];
1688:         p->Addr16[1] = d[1];
1689:
1690:         offset = 2;
1691:         p->samples = d[4];
1692:     }
1693:     if (p->samples > 1) {
1694:         p = Xrealloc(p, sizeof(xbee_pkt) + (sizeof(xbee_sample) * (p->samples - 1)));
1695:     }
1696:     if (xbee.log) {
1697:         xbee_logc("Packet type: %d-bit RX I/O Data (0x%02X)\n", ((t == 0x82)?64:16), t);
1698:         xbee_logc("%d-bit Address: ", ((t == 0x82)?64:16));
1699:         for (j = 0; j < offset; j++) {
1700:             fprintf(xbee.log, (j?"%02X":"%02X"), d[j]);

```

```

1701:     }
1702:     fprintf(xbee.log, "\n");
1703:     xbee_log("RSSI: -%ddb", d[offset]);
1704:     if (d[9] & 0x02) xbee_log("Options: Address Broadcast");
1705:     if (d[9] & 0x02) xbee_log("Options: PAN Broadcast");
1706:     xbee_log("Samples: %d", d[offset + 2]);
1707: }
1708: i = offset + 5;
1709:
1710: /* never returns data */
1711: p->datalen = 0;
1712:
1713: p->dataPkt = FALSE;
1714: p->txStatusPkt = FALSE;
1715: p->modemStatusPkt = FALSE;
1716: p->remoteATPkt = FALSE;
1717: p->IOPkt = TRUE;
1718:
1719: /* save the RSSI / signal strength
1720:    this can be used with printf as:
1721:    printf("-%ddb\n", p->RSSI); */
1722: p->RSSI = d[offset];
1723:
1724: p->status = d[offset + 1];
1725:
1726: /* each sample is split into its own packet here, for simplicity */
1727: for (o = 0; o < p->samples; o++) {
1728:     if (xbee.log) {
1729:         xbee_log("--- Sample %3d -----", o);
1730:     }
1731:
1732:     /* parse the io data */
1733:     i = xbee_parse_io(p, d, offset + 3, i, o);
1734: }
1735: if (xbee.log) {
1736:     xbee_log("-----");
1737: }
1738:
1739: /* ##### */
1740: /* if: Unknown */
1741: } else {
1742:     if (xbee.log) {
1743:         xbee_log("Packet type: Unknown (0x%02X)", t);
1744:     }
1745:     p->type = xbee_unknown;
1746: }
1747: p->next = NULL;
1748:
1749: /* lock the connection mutex */
1750: xbee_mutex_lock(xbee.conmutex);
1751:
1752: con = xbee.conlist;
1753: hasCon = 0;
1754: while (con) {
1755:     if (xbee_matchpktcon(p, con)) {
1756:         hasCon = 1;
1757:         break;
1758:     }
1759:     con = con->next;
1760: }
1761:
1762: /* unlock the connection mutex */
1763: xbee_mutex_unlock(xbee.conmutex);
1764:
1765: /* if the packet doesn't have a connection, don't add it! */
1766: if (!hasCon) {
1767:     xfree(p);
1768:     if (xbee.log) {
1769:         xbee_log("Connectionless packet... discarding!");
1770:     }
1771:     continue;
1772: }
1773:
1774: /* lock the packet mutex, so we can safely add the packet to the list */
1775: xbee_mutex_lock(xbee.pktmutex);
1776:
1777: /* if: the list is empty */
1778: if (!xbee.pktlist) {
1779:     /* start the list! */
1780:     xbee.pktlist = p;
1781: } else if (xbee.pktlast) {
1782:     /* add the packet to the end */
1783:     xbee.pktlast->next = p;
1784: } else {
1785:     /* pktlast wasnt set... look for the end and then set it */

```

```

1786:         i = 0;
1787:         q = xbee.pktlist;
1788:         while (q->next) {
1789:             q = q->next;
1790:             i++;
1791:         }
1792:         q->next = p;
1793:         xbee.pktcount = i;
1794:     }
1795:     xbee.pktlast = p;
1796:     xbee.pktcount++;
1797:
1798:     /* unlock the packet mutex */
1799:     xbee_mutex_unlock(xbee.pktmutex);
1800:
1801:     if (xbee.log) {
1802:         xbee_log("-----");
1803:         xbee_log("Packets: %d", xbee.pktcount);
1804:     }
1805:
1806:     p = q = NULL;
1807: }
1808: return 0;
1809: }
1810:
1811: /* #####
1812:  xbee_getbyte - INTERNAL
1813:  waits for an escaped byte of data */
1814: static unsigned char xbee_getbyte(void) {
1815:     unsigned char c;
1816:
1817:     ISREADY;
1818:
1819:     /* take a byte */
1820:     c = xbee_getrawbyte();
1821:     /* if its escaped, take another and un-escape */
1822:     if (c == 0x7D) c = xbee_getrawbyte() ^ 0x20;
1823:
1824:     return (c & 0xFF);
1825: }
1826:
1827: /* #####
1828:  xbee_getrawbyte - INTERNAL
1829:  waits for a raw byte of data */
1830: static unsigned char xbee_getrawbyte(void) {
1831:     struct timeval to;
1832:     int ret;
1833:     unsigned char c = 0x00;
1834:
1835:     ISREADY;
1836:
1837:     /* the loop is just incase there actually isnt a byte there to be read... */
1838:     do {
1839:         /* wait for a read to be possible */
1840:         /* timeout every 1 second to keep alive */
1841:         memset(&to, 0, sizeof(to));
1842:         to.tv_usec = 1000 * 1000;
1843:         if ((ret = xbee_select(&to)) == -1) {
1844:             perror("libxbee:xbee_getrawbyte()");
1845:             exit(1);
1846:         }
1847:         if (!xbee.listenrun) break;
1848:         if (ret == 0) continue;
1849:
1850:         /* read 1 character */
1851:         xbee_read(&c, 1);
1852: #ifdef _WIN32 /* ---- */
1853:         ret = xbee.ttyr;
1854:         if (ret == 0) {
1855:             usleep(10);
1856:             continue;
1857:         }
1858: #endif /* ----- */
1859:     } while (0);
1860:
1861:     return (c & 0xFF);
1862: }
1863:
1864: /* #####
1865:  xbee_send_pkt - INTERNAL
1866:  sends a complete packet of data */
1867: static void xbee_send_pkt(t_data *pkt) {
1868:     ISREADY;
1869:
1870:     /* lock the send mutex */

```



```

1871: xbee_mutex_lock(xbee.sendmutex);
1872:
1873: /* write and flush the data */
1874: xbee_write(pkt->data,pkt->length);
1875:
1876: /* unlock the mutex */
1877: xbee_mutex_unlock(xbee.sendmutex);
1878:
1879: if (xbee.log) {
1880:     int i,x,y;
1881:     /* prints packet in hex byte-by-byte */
1882:     xbee_logc("TX Packet:");
1883:     for (i=0,x=0,y=0;i<pkt->length;i++,x--) {
1884:         if (x == 0) {
1885:             fprintf(xbee.log,"\n 0x%04X | ",y);
1886:             x = 0x8;
1887:             y += x;
1888:         }
1889:         if (x == 4) {
1890:             fprintf(xbee.log," ");
1891:         }
1892:         fprintf(xbee.log,"0x%02X ",pkt->data[i]);
1893:     }
1894:     fprintf(xbee.log,"\n");
1895: }
1896:
1897: /* free the packet */
1898: Xfree(pkt);
1899: }
1900:
1901: /* #####
1902:     xbee_make_pkt - INTERNAL
1903:     adds delimiter field
1904:     calculates length and checksum
1905:     escapes bytes */
1906: static t_data *xbee_make_pkt(unsigned char *data, int length) {
1907:     t_data *pkt;
1908:     unsigned int l, i, o, t, x, m;
1909:     char d = 0;
1910:
1911:     ISREADY;
1912:
1913:     /* check the data given isnt too long
1914:         100 bytes maximum payload + 12 bytes header information */
1915:     if (length > 100 + 12) return NULL;
1916:
1917:     /* calculate the length of the whole packet
1918:         start, length (MSB), length (LSB), DATA, checksum */
1919:     l = 3 + length + 1;
1920:
1921:     /* prepare memory */
1922:     pkt = Xcalloc(sizeof(t_data));
1923:
1924:     /* put start byte on */
1925:     pkt->data[0] = 0x7E;
1926:
1927:     /* copy data into packet */
1928:     for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
1929:         /* if: its time for the checksum */
1930:         if (i == length) d = M8((0xFF - M8(t)));
1931:         /* if: its time for the high length byte */
1932:         else if (m == 1) d = M8(length >> 8);
1933:         /* if: its time for the low length byte */
1934:         else if (m == 2) d = M8(length);
1935:         /* if: its time for the normal data */
1936:         else if (m > 2) d = data[i];
1937:
1938:         x = 0;
1939:         /* check for any escapes needed */
1940:         if ((d == 0x11) || /* XON */
1941:             (d == 0x13) || /* XOFF */
1942:             (d == 0x7D) || /* Escape */
1943:             (d == 0x7E)) { /* Frame Delimiter */
1944:             l++;
1945:             pkt->data[o++] = 0x7D;
1946:             x = 1;
1947:         }
1948:
1949:         /* move data in */
1950:         pkt->data[o] = ((!x)?d:d^0x20);
1951:         if (m > 2) {
1952:             i++;
1953:             t += d;
1954:         }
1955:     }

```

```
1956:
1957:  /* remember the length */
1958:  pkt->length = 1;
1959:
1960:  return pkt;
1961: }
```