

```
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: const char *SVN_REV = "$Id: api.c 401 2010-07-30 18:09:12Z attie.co.uk $";
21: char svn_rev[128] = "\0";
22:
23: #include "api.h"
24:
25: #ifdef __GNUC__ /* ---- */
26: #include "xsys/linux.c"
27: #else /* ----- */
28: #include "xsys/win32.c"
29: #endif /* ----- */
30:
31: const char *xbee_svn_version(void) {
32:     if (svn_rev[0] == '\0') {
33:         char *t;
34:         sprintf(svn_rev, "r%s", &SVN_REV[11]);
35:         t = strrchr(svn_rev, ' ');
36:         if (t) {
37:             t[0] = '\0';
38:         }
39:     }
40:     return svn_rev;
41: }
42:
43: const char *xbee_build_info(void) {
44:     return "Built on " __DATE__ " @ " __TIME__ " for " HOST_OS;
45: }
46:
47: /* ##### */
48: /* ### Memory Handling ##### */
49: /* ##### */
50:
51: /* malloc wrapper function */
52: static void *Xmalloc(size_t size) {
53:     void *t;
54:     t = malloc(size);
55:     if (!t) {
56:         /* uhoh... thats pretty bad... */
57:         perror("libxbee:malloc()");
58:         exit(1);
59:     }
60:     return t;
61: }
62:
63: /* calloc wrapper function */
64: static void *Xcalloc(size_t size) {
65:     void *t;
66:     t = calloc(1, size);
67:     if (!t) {
68:         /* uhoh... thats pretty bad... */
69:         perror("libxbee:calloc()");
70:         exit(1);
71:     }
72:     return t;
73: }
74:
75: /* realloc wrapper function */
76: static void *Xrealloc(void *ptr, size_t size) {
77:     void *t;
78:     t = realloc(ptr, size);
79:     if (!t) {
80:         /* uhoh... thats pretty bad... */
81:         perror("libxbee:realloc()");
82:         exit(1);
83:     }
84:     return t;
85: }
```

```

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

```

```

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

```

```

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

```

```

341:
342:  /* free all connections */
343:  con = xbee.conlist;
344:  xbee.conlist = NULL;
345:  while (con) {
346:      ncon = con->next;
347:      Xfree(con);
348:      con = ncon;
349:  }
350:
351:  /* free all packets */
352:  xbee.pktlast = NULL;
353:  pkt = xbee.pktlist;
354:  xbee.pktlist = NULL;
355:  while (pkt) {
356:      npkt = pkt->next;
357:      Xfree(pkt);
358:      pkt = npkt;
359:  }
360:
361:  /* destroy mutexes */
362:  xbee_mutex_destroy(xbee.conmutex);
363:  xbee_mutex_destroy(xbee.pktmutex);
364:  xbee_mutex_destroy(xbee.sendmutex);
365:
366:  /* close the serial port */
367:  Xfree(xbee.path);
368:  #ifdef __GNUC__ /* ---- */
369:  if (xbee.tty) xbee_close(xbee.tty);
370:  if (xbee.ttyfd) close(xbee.ttyfd);
371:  #else /* ----- */
372:  if (xbee.tty) CloseHandle(xbee.tty);
373:  #endif /* ----- */
374:
375:  /* close log and tty */
376:  if (xbee.log) {
377:      xbee_log("libxbee: Stopped!");
378:      fflush(xbee.log);
379:      xbee_close(xbee.log);
380:  }
381:  xbee_mutex_destroy(xbee.logmutex);
382:
383:  /* wipe everything else... */
384:  memset(&xbee, 0, sizeof(xbee));
385:
386:  return ret;
387: }
388:
389: /* #####
390:  xbee_setup
391:  opens xbee serial port & creates xbee listen thread
392:  the xbee must be configured for API mode 2
393:  THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
394: int xbee_setup(char *path, int baudrate) {
395:     return xbee_setuplogAPI(path, baudrate, 0, 0, 0);
396: }
397: int xbee_setuplog(char *path, int baudrate, int logfd) {
398:     return xbee_setuplogAPI(path, baudrate, logfd, 0, 0);
399: }
400: int xbee_setupAPI(char *path, int baudrate, char cmdSeq, int cmdTime) {
401:     return xbee_setuplogAPI(path, baudrate, 0, cmdSeq, cmdTime);
402: }
403: int xbee_setuplogAPI(char *path, int baudrate, int logfd, char cmdSeq, int cmdTime) {
404:     t_info info;
405:     int ret;
406:
407:     memset(&xbee, 0, sizeof(xbee));
408:
409:     #ifdef DEBUG
410:     /* logfd or stderr */
411:     xbee.logfd = ((logfd)?logfd:2);
412:     #else
413:     xbee.logfd = logfd;
414:     #endif
415:     xbee_mutex_init(xbee.logmutex);
416:     if (xbee.logfd) {
417:         xbee.log = fdopen(xbee.logfd, "w");
418:         if (!xbee.log) {
419:             /* errno == 9 is bad file descriptor (probably not provided) */
420:             if (errno != 9) perror("xbee_setup(): Failed opening logfile");
421:             xbee.logfd = 0;
422:         } else {
423:             #ifdef __GNUC__ /* ---- */
424:             /* set to line buffer - ensure lines are written to file when complete */
425:             setvbuf(xbee.log, NULL, _IOLBF, BUFSIZ);

```

```

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

```

```

511: }
512:
513: /* allow the listen thread to start */
514: xbee_ready = -1;
515:
516: /* can start xbee_listen thread now */
517: if (xbee_thread_create(xbee.listent,xbee_listen_wrapper,&info)) {
518:     perror("xbee_setup():xbee_thread_create()");
519:     if (xbee.log) fclose(xbee.log);
520:     xbee_mutex_destroy(xbee.conmutex);
521:     xbee_mutex_destroy(xbee.pktmutex);
522:     xbee_mutex_destroy(xbee.sendmutex);
523:     Xfree(xbee.path);
524: #ifdef __GNUC__ /* ---- */
525:     close(xbee.ttyfd);
526: #endif /* ----- */
527:     xbee_close(xbee.tty);
528:     return -1;
529: }
530:
531: usleep(500);
532: while (xbee_ready != -2) {
533:     usleep(500);
534:     if (xbee.log) {
535:         xbee_log("Waiting for xbee_listen() to be ready...");
536:     }
537: }
538:
539: /* allow other functions to be used! */
540: xbee_ready = 1;
541:
542: if (xbee.log) xbee_log("libxbee: Started!");
543:
544: return 0;
545: }
546:
547: /* #####
548: xbee_con
549: produces a connection to the specified device and frameID
550: if a connection had already been made, then this connection will be returned */
551: xbee_con *xbee_newcon(unsigned char frameID, xbee_types type, ...) {
552:     xbee_con *con, *ocon;
553:     unsigned char tAddr[8];
554:     va_list ap;
555:     int t;
556:     int i;
557:
558:     ISREADY;
559:
560:     if (!type || type == xbee_unknown) type = xbee_localAT; /* default to local AT */
561:     else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */
562:
563:     va_start(ap,type);
564:     /* if: 64 bit address expected (2 ints) */
565:     if ((type == xbee_64bitRemoteAT) ||
566:         (type == xbee_64bitData) ||
567:         (type == xbee_64bitIO)) {
568:         t = va_arg(ap, int);
569:         tAddr[0] = (t >> 24) & 0xFF;
570:         tAddr[1] = (t >> 16) & 0xFF;
571:         tAddr[2] = (t >> 8) & 0xFF;
572:         tAddr[3] = (t >> 0) & 0xFF;
573:         t = va_arg(ap, int);
574:         tAddr[4] = (t >> 24) & 0xFF;
575:         tAddr[5] = (t >> 16) & 0xFF;
576:         tAddr[6] = (t >> 8) & 0xFF;
577:         tAddr[7] = (t >> 0) & 0xFF;
578:
579:         /* if: 16 bit address expected (1 int) */
580:     } else if ((type == xbee_16bitRemoteAT) ||
581:               (type == xbee_16bitData) ||
582:               (type == xbee_16bitIO)) {
583:         t = va_arg(ap, int);
584:         tAddr[0] = (t >> 8) & 0xFF;
585:         tAddr[1] = (t >> 0) & 0xFF;
586:         tAddr[2] = 0;
587:         tAddr[3] = 0;
588:         tAddr[4] = 0;
589:         tAddr[5] = 0;
590:         tAddr[6] = 0;
591:         tAddr[7] = 0;
592:
593:         /* otherwise clear the address */
594:     } else {
595:         memset(tAddr,0,8);

```

```

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

```



```

681:         break;
682:     case xbee_16bitIO:
683:     case xbee_64bitIO:
684:         xbee_logc("New %d-bit IO connection! (to: ", (con->tAddr64?64:16));
685:         for (i=0;i<(con->tAddr64?8:2);i++) {
686:             fprintf(xbee.log, (i?"%02X":"%02X"), tAddr[i]);
687:         }
688:         fprintf(xbee.log, ")\n");
689:         xbee_logcf();
690:         break;
691:     case xbee_txStatus:
692:         xbee_log("New Tx status connection!");
693:         break;
694:     case xbee_modemStatus:
695:         xbee_log("New modem status connection!");
696:         break;
697:     case xbee_unknown:
698:     default:
699:         xbee_log("New unknown connection!");
700:     }
701: }
702:
703: /* make it the last in the list */
704: con->next = NULL;
705: /* add it to the list */
706: if (xbee.conlist) {
707:     ocon->next = con;
708: } else {
709:     xbee.conlist = con;
710: }
711:
712: /* unlock the mutex */
713: xbee_mutex_unlock(xbee.conmutex);
714: return con;
715: }
716:
717: /* #####
718: xbee_conflush
719: removes any packets that have been collected for the specified
720: connection */
721: void xbee_flushcon(xbee_con *con) {
722:     xbee_pkt *r, *p, *n;
723:
724:     /* lock the packet mutex */
725:     xbee_mutex_lock(xbee.pktmutex);
726:
727:     /* if: there are packets */
728:     if ((p = xbee.pktlist) != NULL) {
729:         r = NULL;
730:         /* get all packets for this connection */
731:         do {
732:             /* does the packet match the connection? */
733:             if (xbee_matchpktcon(p, con)) {
734:                 /* if it was the first packet */
735:                 if (!r) {
736:                     /* move the chain along */
737:                     xbee.pktlist = p->next;
738:                 } else {
739:                     /* otherwise relink the list */
740:                     r->next = p->next;
741:                 }
742:                 xbee.pktcount--;
743:
744:                 /* free this packet! */
745:                 n = p->next;
746:                 Xfree(p);
747:                 /* move on */
748:                 p = n;
749:             } else {
750:                 /* move on */
751:                 r = p;
752:                 p = p->next;
753:             }
754:         } while (p);
755:         xbee.pktlast = r;
756:     }
757:
758:     /* unlock the packet mutex */
759:     xbee_mutex_unlock(xbee.pktmutex);
760: }
761:
762: /* #####
763: xbee_endcon
764: close the unwanted connection
765: free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */

```

```

766: void xbee_endcon2(xbee_con **con, int skipUnlink) {
767:     xbee_con *t, *u;
768:
769:     if (!skipUnlink) {
770:         /* lock the connection mutex */
771:         xbee_mutex_lock(xbee.conmutex);
772:
773:         u = t = xbee.conlist;
774:         while (t && t != *con) {
775:             u = t;
776:             t = t->next;
777:         }
778:         if (!t) {
779:             /* invalid connection given... */
780:             if (xbee.log) {
781:                 xbee_log("Attempted to close invalid connection...");
782:             }
783:             /* unlock the connection mutex */
784:             xbee_mutex_unlock(xbee.conmutex);
785:             return;
786:         }
787:         /* extract this connection from the list */
788:         u->next = t->next;
789:
790:         /* unlock the connection mutex */
791:         xbee_mutex_unlock(xbee.conmutex);
792:     }
793:
794:     /* check if a callback thread is running... */
795:     if (t->callback && xbee_mutex_trylock(t->callbackmutex)) {
796:         /* if it is running... tell it to destroy the connection on completion */
797:         xbee_log("Attempted to close a connection with active callbacks... "
798:             "Connection will be destroyed when callbacks have completed...");
799:         t->destroySelf = 1;
800:         return;
801:     }
802:
803:     /* remove all packets for this connection */
804:     xbee_flushcon(t);
805:
806:     /* destroy the callback mutex */
807:     xbee_mutex_destroy(t->callbackmutex);
808:
809:     /* free the connection! */
810:     Xfree(*con);
811: }
812:
813: /* #####
814: xbee_senddata
815: send the specified data to the provided connection */
816: int xbee_senddata(xbee_con *con, char *format, ...) {
817:     int ret;
818:     va_list ap;
819:
820:     ISREADY;
821:
822:     /* xbee_vsenddata() wants a va_list... */
823:     va_start(ap, format);
824:     /* hand it over :) */
825:     ret = xbee_vsenddata(con, format, ap);
826:     va_end(ap);
827:     return ret;
828: }
829:
830: int xbee_vsenddata(xbee_con *con, char *format, va_list ap) {
831:     unsigned char data[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
832:     int length;
833:
834:     ISREADY;
835:
836:     /* make up the data and keep the length, its possible there are nulls in there */
837:     length = vsnprintf((char *)data, 128, format, ap);
838:
839:     /* hand it over :) */
840:     return xbee_nsenddata(con, (char *)data, length);
841: }
842:
843: int xbee_nsenddata(xbee_con *con, char *data, int length) {
844:     t_data *pkt;
845:     int i;
846:     unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
847:
848:     ISREADY;
849:
850:     if (!con) return -1;

```

```

851:  if (con->type == xbee_unknown) return -1;
852:  if (length > 127) return -1;
853:
854:
855:  if (xbee.log) {
856:    xbee_log("==== TX Packet =====");
857:    xbee_logc("Connection Type: ");
858:    switch (con->type) {
859:      case xbee_unknown:      fprintf(xbee.log,"Unknown"); break;
860:      case xbee_localAT:      fprintf(xbee.log,"Local AT"); break;
861:      case xbee_remoteAT:     fprintf(xbee.log,"Remote AT"); break;
862:      case xbee_16bitRemoteAT: fprintf(xbee.log,"Remote AT (16-bit)"); break;
863:      case xbee_64bitRemoteAT: fprintf(xbee.log,"Remote AT (64-bit)"); break;
864:      case xbee_16bitData:     fprintf(xbee.log,"Data (16-bit)"); break;
865:      case xbee_64bitData:     fprintf(xbee.log,"Data (64-bit)"); break;
866:      case xbee_16bitIO:       fprintf(xbee.log,"IO (16-bit)"); break;
867:      case xbee_64bitIO:       fprintf(xbee.log,"IO (64-bit)"); break;
868:      case xbee_txStatus:      fprintf(xbee.log,"Tx Status"); break;
869:      case xbee_modemStatus:   fprintf(xbee.log,"Modem Status"); break;
870:    }
871:    xbee_logcf();
872:    xbee_logc("Destination: ");
873:    for (i=0;i<(con->tAddr64?8:2);i++) {
874:      fprintf(xbee.log,(i?"%02X":"%02X"),con->tAddr[i]);
875:    }
876:    xbee_logcf();
877:    xbee_log("Length: %d",length);
878:    for (i=0;i<length;i++) {
879:      xbee_logc("%3d | 0x%02X ",i,(unsigned char)data[i]);
880:      if ((data[i] > 32) && (data[i] < 127)) {
881:        fprintf(xbee.log,"%c",data[i]);
882:      } else {
883:        fprintf(xbee.log,"_");
884:      }
885:    }
886:    xbee_logcf();
887:  }
888:
889:  /* ##### */
890:  /* if: local AT */
891:  if (con->type == xbee_localAT) {
892:    /* AT commands are 2 chars long (plus optional parameter) */
893:    if (length < 2) return -1;
894:
895:    /* use the command? */
896:    buf[0] = ((!con->atQueue)?XBEE_LOCAL_ATREQ:XBEE_LOCAL_ATQUE);
897:    buf[1] = con->frameID;
898:
899:    /* copy in the data */
900:    for (i=0;i<length;i++) {
901:      buf[i+2] = data[i];
902:    }
903:
904:    /* setup the packet */
905:    pkt = xbee_make_pkt(buf,i+2);
906:    /* send it on */
907:    xbee_send_pkt(pkt);
908:
909:    return 0;
910:
911:    /* ##### */
912:    /* if: remote AT */
913:  } else if ((con->type == xbee_16bitRemoteAT) ||
914:            (con->type == xbee_64bitRemoteAT)) {
915:    if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter) */
916:    buf[0] = XBEE_REMOTE_ATREQ;
917:    buf[1] = con->frameID;
918:
919:    /* copy in the relevant address */
920:    if (con->tAddr64) {
921:      memcpy(&buf[2],con->tAddr,8);
922:      buf[10] = 0xFF;
923:      buf[11] = 0xFE;
924:    } else {
925:      memset(&buf[2],0,8);
926:      memcpy(&buf[10],con->tAddr,2);
927:    }
928:    /* queue the command? */
929:    buf[12] = ((!con->atQueue)?0x02:0x00);
930:
931:    /* copy in the data */
932:    for (i=0;i<length;i++) {
933:      buf[i+13] = data[i];
934:    }
935:

```

```

936:      /* setup the packet */
937:      pkt = xbee_make_pkt(buf,i+13);
938:      /* send it on */
939:      xbee_send_pkt(pkt);
940:
941:      return 0;
942:
943:      /* ##### */
944:      /* if: 16 or 64bit Data */
945:  } else if ((con->type == xbee_16bitData) ||
946:             (con->type == xbee_64bitData)) {
947:      int offset;
948:
949:      /* if: 16bit Data */
950:      if (con->type == xbee_16bitData) {
951:          buf[0] = XBEE_16BIT_DATATX;
952:          offset = 5;
953:          /* copy in the address */
954:          memcpy(&buf[2],con->tAddr,2);
955:
956:          /* if: 64bit Data */
957:      } else { /* 64bit Data */
958:          buf[0] = XBEE_64BIT_DATATX;
959:          offset = 11;
960:          /* copy in the address */
961:          memcpy(&buf[2],con->tAddr,8);
962:      }
963:
964:      /* copy frameID */
965:      buf[1] = con->frameID;
966:
967:      /* disable ack? broadcast? */
968:      buf[offset-1] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
969:
970:      /* copy in the data */
971:      for (i=0;i<length;i++) {
972:          buf[i+offset] = data[i];
973:      }
974:
975:      /* setup the packet */
976:      pkt = xbee_make_pkt(buf,i+offset);
977:      /* send it on */
978:      xbee_send_pkt(pkt);
979:
980:      return 0;
981:
982:      /* ##### */
983:      /* if: I/O */
984:  } else if ((con->type == xbee_64bitIO) ||
985:             (con->type == xbee_16bitIO)) {
986:      /* not currently implemented... is it even allowed? */
987:      if (xbee.log) {
988:          xbee_log("***** TODO *****\n");
989:      }
990:  }
991:
992:  return -2;
993: }
994:
995: /* #####
996: xbee_getpacket
997: retrieves the next packet destined for the given connection
998: once the packet has been retrieved, it is removed for the list! */
999: xbee_pkt *xbee_getpacketwait(xbee_con *con) {
1000:     xbee_pkt *p;
1001:     int i;
1002:
1003:     /* 50ms * 20 = 1 second */
1004:     for (i = 0; i < 20; i++) {
1005:         p = xbee_getpacket(con);
1006:         if (p) break;
1007:         usleep(50000); /* 50ms */
1008:     }
1009:
1010:     return p;
1011: }
1012: xbee_pkt *xbee_getpacket(xbee_con *con) {
1013:     xbee_pkt *l, *p, *q;
1014:
1015:     /* lock the packet mutex */
1016:     xbee_mutex_lock(xbee.pktmutex);
1017:
1018:     /* if: there are no packets */
1019:     if ((p = xbee.pktlist) == NULL) {
1020:         xbee_mutex_unlock(xbee.pktmutex);

```

```

1021:     /*if (xbee.log) {
1022:         xbee_log("No packets available...");
1023:     }*/
1024:     return NULL;
1025: }
1026:
1027: l = NULL;
1028: q = NULL;
1029: /* get the first available packet for this connection */
1030: do {
1031:     /* does the packet match the connection? */
1032:     if (xbee_matchpktcon(p,con)) {
1033:         q = p;
1034:         break;
1035:     }
1036:     /* move on */
1037:     l = p;
1038:     p = p->next;
1039: } while (p);
1040:
1041: /* if: no packet was found */
1042: if (!q) {
1043:     xbee_mutex_unlock(xbee.pktmutex);
1044:     return NULL;
1045: }
1046:
1047: /* if it was the first packet */
1048: if (l) {
1049:     /* relink the list */
1050:     l->next = p->next;
1051:     if (!l->next) xbee.pktlast = l;
1052: } else {
1053:     /* move the chain along */
1054:     xbee.pktlist = p->next;
1055:     if (!xbee.pktlist) {
1056:         xbee.pktlast = NULL;
1057:     } else if (!xbee.pktlist->next) {
1058:         xbee.pktlast = xbee.pktlist;
1059:     }
1060: }
1061: xbee.pktcount--;
1062:
1063: /* unlink this packet from the chain! */
1064: q->next = NULL;
1065:
1066: if (xbee.log) {
1067:     xbee_log("==== Get Packet =====");
1068:     xbee_log("Got a packet");
1069:     xbee_log("Packets left: %d",xbee.pktcount);
1070: }
1071:
1072: /* unlock the packet mutex */
1073: xbee_mutex_unlock(xbee.pktmutex);
1074:
1075: /* and return the packet (must be free'd by caller!) */
1076: return q;
1077: }
1078:
1079: /* #####
1080: xbee_matchpktcon - INTERNAL
1081: checks if the packet matches the connection */
1082: static int xbee_matchpktcon(xbee_pkt *pkt, xbee_con *con) {
1083:     /* if: the connection type matches the packet type OR
1084:     the connection is 16/64bit remote AT, and the packet is a remote AT response */
1085:     if ((pkt->type == con->type) || /* -- */
1086:         ((pkt->type == xbee_remoteAT) && /* -- */
1087:          ((con->type == xbee_16bitRemoteAT) ||
1088:           (con->type == xbee_64bitRemoteAT)))) {
1089:
1090:         /* if: the packet is modem status OR
1091:         the packet is tx status or AT data and the frame IDs match OR
1092:         the addresses match */
1093:         if (pkt->type == xbee_modemStatus) return 1;
1094:
1095:         if ((pkt->type == xbee_txStatus) ||
1096:             (pkt->type == xbee_localAT) ||
1097:             (pkt->type == xbee_remoteAT)) {
1098:             if (pkt->frameID == con->frameID) {
1099:                 return 1;
1100:             }
1101:         } else if (pkt->sAddr64 && !memcmp(pkt->Addr64,con->tAddr,8)) {
1102:             return 1;
1103:         } else if (!pkt->sAddr64 && !memcmp(pkt->Addr16,con->tAddr,2)) {
1104:             return 1;
1105:         }

```

```

1106:     }
1107:     return 0;
1108: }
1109:
1110: /* #####
1111: xbee_parse_io - INTERNAL
1112: parses the data given into the packet io information */
1113: static int xbee_parse_io(xbee_pkt *p, unsigned char *d, int maskOffset, int sampleOffset, int sample) {
1114:     xbee_sample *s = &(p->IOdata[sample]);
1115:
1116:     /* copy in the I/O data mask */
1117:     s->Iomask = (((d[maskOffset]<<8) | d[maskOffset + 1]) & 0x7FFF);
1118:
1119:     /* copy in the digital I/O data */
1120:     s->IOdigital = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x01FF);
1121:
1122:     /* advance over the digital data, if its there */
1123:     sampleOffset += ((s->Iomask & 0x01FF)?2:0);
1124:
1125:     /* copy in the analog I/O data */
1126:     if (s->Iomask & 0x0200) {
1127:         s->IOanalog[0] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1128:         sampleOffset+=2;
1129:     }
1130:     if (s->Iomask & 0x0400) {
1131:         s->IOanalog[1] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1132:         sampleOffset+=2;
1133:     }
1134:     if (s->Iomask & 0x0800) {
1135:         s->IOanalog[2] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1136:         sampleOffset+=2;
1137:     }
1138:     if (s->Iomask & 0x1000) {
1139:         s->IOanalog[3] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1140:         sampleOffset+=2;
1141:     }
1142:     if (s->Iomask & 0x2000) {
1143:         s->IOanalog[4] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1144:         sampleOffset+=2;
1145:     }
1146:     if (s->Iomask & 0x4000) {
1147:         s->IOanalog[5] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1148:         sampleOffset+=2;
1149:     }
1150:
1151:     if (xbee.log) {
1152:         if (s->Iomask & 0x0001)
1153:             xbee_log("Digital 0: %c",((s->IOdigital & 0x0001)?'1':'0'));
1154:         if (s->Iomask & 0x0002)
1155:             xbee_log("Digital 1: %c",((s->IOdigital & 0x0002)?'1':'0'));
1156:         if (s->Iomask & 0x0004)
1157:             xbee_log("Digital 2: %c",((s->IOdigital & 0x0004)?'1':'0'));
1158:         if (s->Iomask & 0x0008)
1159:             xbee_log("Digital 3: %c",((s->IOdigital & 0x0008)?'1':'0'));
1160:         if (s->Iomask & 0x0010)
1161:             xbee_log("Digital 4: %c",((s->IOdigital & 0x0010)?'1':'0'));
1162:         if (s->Iomask & 0x0020)
1163:             xbee_log("Digital 5: %c",((s->IOdigital & 0x0020)?'1':'0'));
1164:         if (s->Iomask & 0x0040)
1165:             xbee_log("Digital 6: %c",((s->IOdigital & 0x0040)?'1':'0'));
1166:         if (s->Iomask & 0x0080)
1167:             xbee_log("Digital 7: %c",((s->IOdigital & 0x0080)?'1':'0'));
1168:         if (s->Iomask & 0x0100)
1169:             xbee_log("Digital 8: %c",((s->IOdigital & 0x0100)?'1':'0'));
1170:         if (s->Iomask & 0x0200)
1171:             xbee_log("Analog 0: %d ( %.2fv)\n",s->IOanalog[0],(3.3/1023)*s->IOanalog[0]);
1172:         if (s->Iomask & 0x0400)
1173:             xbee_log("Analog 1: %d ( %.2fv)\n",s->IOanalog[1],(3.3/1023)*s->IOanalog[1]);
1174:         if (s->Iomask & 0x0800)
1175:             xbee_log("Analog 2: %d ( %.2fv)\n",s->IOanalog[2],(3.3/1023)*s->IOanalog[2]);
1176:         if (s->Iomask & 0x1000)
1177:             xbee_log("Analog 3: %d ( %.2fv)\n",s->IOanalog[3],(3.3/1023)*s->IOanalog[3]);
1178:         if (s->Iomask & 0x2000)
1179:             xbee_log("Analog 4: %d ( %.2fv)\n",s->IOanalog[4],(3.3/1023)*s->IOanalog[4]);
1180:         if (s->Iomask & 0x4000)
1181:             xbee_log("Analog 5: %d ( %.2fv)\n",s->IOanalog[5],(3.3/1023)*s->IOanalog[5]);
1182:     }
1183:
1184:     return sampleOffset;
1185: }
1186:
1187: /* #####
1188: xbee_listen_stop
1189: stops the listen thread after the current packet has been processed */
1190: void xbee_listen_stop(void) {

```

```

1191:     xbee.listenrun = 0;
1192: }
1193:
1194: /* #####
1195:  xbee_listen_wrapper - INTERNAL
1196:  the xbee_listen wrapper. Prints an error when xbee_listen ends */
1197: static void xbee_listen_wrapper(t_info *info) {
1198:     int ret;
1199:     /* just falls out if the proper 'go-ahead' isn't given */
1200:     if (xbee_ready != -1) return;
1201:     /* now allow the parent to continue */
1202:     xbee_ready = -2;
1203:
1204: #ifdef _WIN32 /* ---- */
1205:     /* win32 requires this delay... no idea why */
1206:     usleep(1000000);
1207: #endif /* ----- */
1208:
1209:     while (xbee.listenrun) {
1210:         info->i = -1;
1211:         ret = xbee_listen(info);
1212:         if (!xbee.listenrun) break;
1213:         if (xbee.log) {
1214:             xbee_log("xbee_listen() returned [%d]... Restarting in 250ms!",ret);
1215:         }
1216:         usleep(25000);
1217:     }
1218: }
1219:
1220: /* xbee_listen - INTERNAL
1221:  the xbee xbee_listen thread
1222:  reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
1223: static int xbee_listen(t_info *info) {
1224:     unsigned char c, t, d[1024];
1225:     unsigned int l, i, chksum, o;
1226:     struct timeval tv;
1227:     int j;
1228:     xbee_pkt *p, *q;
1229:     xbee_con *con;
1230:     int hasCon;
1231:
1232:     /* just falls out if the proper 'go-ahead' isn't given */
1233:     if (info->i != -1) return -1;
1234:     /* do this forever :) */
1235:     while (xbee.listenrun) {
1236:         /* wait for a valid start byte */
1237:         if (xbee_getrawbyte() != 0x7E) continue;
1238:         if (!xbee.listenrun) return 0;
1239:
1240:         if (xbee.log) {
1241:             xbee_log("==== RX Packet =====");
1242:             gettimeofday(&tv,NULL);
1243:             xbee_log("Got a packet @ %ld.%06ld",tv.tv_sec,tv.tv_usec);
1244:         }
1245:
1246:         /* get the length */
1247:         l = xbee_getbyte() << 8;
1248:         l += xbee_getbyte();
1249:
1250:         /* check it is a valid length... */
1251:         if (!l) {
1252:             if (xbee.log) {
1253:                 xbee_log("Recived zero length packet!");
1254:             }
1255:             continue;
1256:         }
1257:         if (l > 100) {
1258:             if (xbee.log) {
1259:                 xbee_log("Recived oversized packet! Length: %d",l - 1);
1260:             }
1261:         }
1262:         if (l > sizeof(d) - 1) {
1263:             if (xbee.log) {
1264:                 xbee_log("Recived packet larger than buffer! Discarding...");
1265:             }
1266:             continue;
1267:         }
1268:
1269:         if (xbee.log) {
1270:             xbee_log("Length: %d",l - 1);
1271:         }
1272:
1273:         /* get the packet type */
1274:         t = xbee_getbyte();
1275:

```



```

1276:      /* start the checksum */
1277:      chksum = t;
1278:
1279:      /* suck in all the data */
1280:      for (i = 0; l > 1 && i < 128; l--, i++) {
1281:          /* get an unescaped byte */
1282:          c = xbee_getbyte();
1283:          d[i] = c;
1284:          chksum += c;
1285:          if (xbee.log) {
1286:              xbee_logc("%3d | 0x%02X | ", i, c);
1287:              if ((c > 32) && (c < 127)) fprintf(xbee.log, "'%c'", c); else fprintf(xbee.log, " _ ");
1288:
1289:              if ((t == XBEE_64BIT_DATA && i == 10) ||
1290:                  (t == XBEE_16BIT_DATA && i == 4) ||
1291:                  (t == XBEE_LOCAL_AT && i == 4) ||
1292:                  (t == XBEE_REMOTE_AT && i == 14)) {
1293:                  /* mark the beginning of the 'data' bytes */
1294:                  fprintf(xbee.log, " <-- data starts");
1295:              }
1296:              xbee_logcf();
1297:          }
1298:      }
1299:      i--; /* it went up too many times!... */
1300:
1301:      /* add the checksum */
1302:      chksum += xbee_getbyte();
1303:
1304:      /* check if the whole packet was recieved, or something else occurred... unlikely... */
1305:      if (l > 1) {
1306:          if (xbee.log) {
1307:              xbee_log("Didn't get whole packet... :(");
1308:          }
1309:          continue;
1310:      }
1311:
1312:      /* check the checksum */
1313:      if ((chksum & 0xFF) != 0xFF) {
1314:          if (xbee.log) {
1315:              xbee_log("Invalid Checksum: 0x%02X", chksum);
1316:          }
1317:          continue;
1318:      }
1319:
1320:      /* make a new packet */
1321:      p = Xcalloc(sizeof(xbee_pkt));
1322:      q = NULL;
1323:      p->datalen = 0;
1324:
1325:      /* ##### */
1326:      /* if: modem status */
1327:      if (t == XBEE_MODEM_STATUS) {
1328:          if (xbee.log) {
1329:              xbee_log("Packet type: Modem Status (0x8A)");
1330:              xbee_logc("Event: ");
1331:              switch (d[0]) {
1332:                  case 0x00: fprintf(xbee.log, "Hardware reset"); break;
1333:                  case 0x01: fprintf(xbee.log, "Watchdog timer reset"); break;
1334:                  case 0x02: fprintf(xbee.log, "Associated"); break;
1335:                  case 0x03: fprintf(xbee.log, "Disassociated"); break;
1336:                  case 0x04: fprintf(xbee.log, "Synchronization lost"); break;
1337:                  case 0x05: fprintf(xbee.log, "Coordinator realignment"); break;
1338:                  case 0x06: fprintf(xbee.log, "Coordinator started"); break;
1339:              }
1340:              fprintf(xbee.log, "... (0x%02X)", d[0]);
1341:              xbee_logcf();
1342:          }
1343:          p->type = xbee_modemStatus;
1344:
1345:          p->sAddr64 = FALSE;
1346:          p->dataPkt = FALSE;
1347:          p->txStatusPkt = FALSE;
1348:          p->modemStatusPkt = TRUE;
1349:          p->remoteATPkt = FALSE;
1350:          p->IOPkt = FALSE;
1351:
1352:          /* modem status can only ever give 1 'data' byte */
1353:          p->datalen = 1;
1354:          p->data[0] = d[0];
1355:
1356:          /* ##### */
1357:          /* if: local AT response */
1358:      } else if (t == XBEE_LOCAL_AT) {
1359:          if (xbee.log) {
1360:              xbee_log("Packet type: Local AT Response (0x88)");

```



```

1361:         xbee_log("FrameID: 0x%02X",d[0]);
1362:         xbee_log("AT Command: %c%c",d[1],d[2]);
1363:         xbee_logc("Status: ");
1364:         if (d[3] == 0) fprintf(xbee.log,"OK");
1365:         else if (d[3] == 1) fprintf(xbee.log,"Error");
1366:         else if (d[3] == 2) fprintf(xbee.log,"Invalid Command");
1367:         else if (d[3] == 3) fprintf(xbee.log,"Invalid Parameter");
1368:         fprintf(xbee.log," (0x%02X)",d[3]);
1369:         xbee_logcf();
1370:     }
1371:     p->type = xbee_localAT;
1372:
1373:     p->sAddr64 = FALSE;
1374:     p->dataPkt = FALSE;
1375:     p->txStatusPkt = FALSE;
1376:     p->modemStatusPkt = FALSE;
1377:     p->remoteATPkt = FALSE;
1378:     p->IOPkt = FALSE;
1379:
1380:     p->frameID = d[0];
1381:     p->atCmd[0] = d[1];
1382:     p->atCmd[1] = d[2];
1383:
1384:     p->status = d[3];
1385:
1386:     /* copy in the data */
1387:     p->datalen = i-3;
1388:     for (;i>3;i--) p->data[i-4] = d[i];
1389:
1390:     /* ##### */
1391:     /* if: remote AT response */
1392: } else if (t == XBEE_REMOTE_AT) {
1393:     if (xbee.log) {
1394:         xbee_log("Packet type: Remote AT Response (0x97)");
1395:         xbee_log("FrameID: 0x%02X",d[0]);
1396:         xbee_logc("64-bit Address: ");
1397:         for (j=0;j<8;j++) {
1398:             fprintf(xbee.log,(j?"%02X":"%02X"),d[1+j]);
1399:         }
1400:         xbee_logcf();
1401:         xbee_logc("16-bit Address: ");
1402:         for (j=0;j<2;j++) {
1403:             fprintf(xbee.log,(j?"%02X":"%02X"),d[9+j]);
1404:         }
1405:         xbee_logcf();
1406:         xbee_log("AT Command: %c%c",d[11],d[12]);
1407:         xbee_logc("Status: ");
1408:         if (d[13] == 0) fprintf(xbee.log,"OK");
1409:         else if (d[13] == 1) fprintf(xbee.log,"Error");
1410:         else if (d[13] == 2) fprintf(xbee.log,"Invalid Command");
1411:         else if (d[13] == 3) fprintf(xbee.log,"Invalid Parameter");
1412:         else if (d[13] == 4) fprintf(xbee.log,"No Response");
1413:         fprintf(xbee.log," (0x%02X)",d[13]);
1414:         xbee_logcf();
1415:     }
1416:     p->type = xbee_remoteAT;
1417:
1418:     p->sAddr64 = FALSE;
1419:     p->dataPkt = FALSE;
1420:     p->txStatusPkt = FALSE;
1421:     p->modemStatusPkt = FALSE;
1422:     p->remoteATPkt = TRUE;
1423:     p->IOPkt = FALSE;
1424:
1425:     p->frameID = d[0];
1426:
1427:     p->Addr64[0] = d[1];
1428:     p->Addr64[1] = d[2];
1429:     p->Addr64[2] = d[3];
1430:     p->Addr64[3] = d[4];
1431:     p->Addr64[4] = d[5];
1432:     p->Addr64[5] = d[6];
1433:     p->Addr64[6] = d[7];
1434:     p->Addr64[7] = d[8];
1435:
1436:     p->Addr16[0] = d[9];
1437:     p->Addr16[1] = d[10];
1438:
1439:     p->atCmd[0] = d[11];
1440:     p->atCmd[1] = d[12];
1441:
1442:     p->status = d[13];
1443:
1444:     p->samples = 1;
1445:

```

```

1446:     if (p->status == 0x00 && p->atCmd[0] == 'I' && p->atCmd[1] == 'S') {
1447:         /* parse the io data */
1448:         if (xbee.log) xbee_log("--- Sample -----");
1449:         xbee_parse_io(p, d, 15, 17, 0);
1450:         if (xbee.log) xbee_log("-----");
1451:     } else {
1452:         /* copy in the data */
1453:         p->datalen = i-13;
1454:         for (;i>13;i--) p->data[i-14] = d[i];
1455:     }
1456:
1457:     /* ##### */
1458:     /* if: TX status */
1459: } else if (t == XBEE_TX_STATUS) {
1460:     if (xbee.log) {
1461:         xbee_log("Packet type: TX Status Report (0x89)");
1462:         xbee_log("FrameID: 0x%02X",d[0]);
1463:         xbee_logc("Status: ");
1464:         if (d[1] == 0) fprintf(xbee.log,"Success");
1465:         else if (d[1] == 1) fprintf(xbee.log,"No ACK");
1466:         else if (d[1] == 2) fprintf(xbee.log,"CCA Failure");
1467:         else if (d[1] == 3) fprintf(xbee.log,"Purged");
1468:         fprintf(xbee.log," (0x%02X)",d[1]);
1469:         xbee_logcf();
1470:     }
1471:     p->type = xbee_txStatus;
1472:
1473:     p->sAddr64 = FALSE;
1474:     p->dataPkt = FALSE;
1475:     p->txStatusPkt = TRUE;
1476:     p->modemStatusPkt = FALSE;
1477:     p->remoteATPkt = FALSE;
1478:     p->IOPkt = FALSE;
1479:
1480:     p->frameID = d[0];
1481:
1482:     p->status = d[1];
1483:
1484:     /* never returns data */
1485:     p->datalen = 0;
1486:
1487:     /* ##### */
1488:     /* if: 16 / 64bit data recieve */
1489: } else if ((t == XBEE_64BIT_DATA) ||
1490:           (t == XBEE_16BIT_DATA)) {
1491:     int offset;
1492:     if (t == XBEE_64BIT_DATA) { /* 64bit */
1493:         offset = 8;
1494:     } else { /* 16bit */
1495:         offset = 2;
1496:     }
1497:     if (xbee.log) {
1498:         xbee_log("Packet type: %d-bit RX Data (0x%02X)",((t == XBEE_64BIT_DATA)?64:16),t);
1499:         xbee_logc("%d-bit Address: ",((t == XBEE_64BIT_DATA)?64:16));
1500:         for (j=0;j<offset;j++) {
1501:             fprintf(xbee.log,(j?"%02X":"%02X"),d[j]);
1502:         }
1503:         xbee_logcf();
1504:         xbee_log("RSSI: -%ddb",d[offset]);
1505:         if (d[offset + 1] & 0x02) xbee_log("Options: Address Broadcast");
1506:         if (d[offset + 1] & 0x03) xbee_log("Options: PAN Broadcast");
1507:     }
1508:     p->dataPkt = TRUE;
1509:     p->txStatusPkt = FALSE;
1510:     p->modemStatusPkt = FALSE;
1511:     p->remoteATPkt = FALSE;
1512:     p->IOPkt = FALSE;
1513:
1514:     if (t == XBEE_64BIT_DATA) { /* 64bit */
1515:         p->type = xbee_64bitData;
1516:
1517:         p->sAddr64 = TRUE;
1518:
1519:         p->Addr64[0] = d[0];
1520:         p->Addr64[1] = d[1];
1521:         p->Addr64[2] = d[2];
1522:         p->Addr64[3] = d[3];
1523:         p->Addr64[4] = d[4];
1524:         p->Addr64[5] = d[5];
1525:         p->Addr64[6] = d[6];
1526:         p->Addr64[7] = d[7];
1527:     } else { /* 16bit */
1528:         p->type = xbee_16bitData;
1529:
1530:         p->sAddr64 = FALSE;

```

```

1531:
1532:     p->Addr16[0] = d[0];
1533:     p->Addr16[1] = d[1];
1534: }
1535:
1536: /* save the RSSI / signal strength
1537:    this can be used with printf as:
1538:    printf("-%ddB\n",p->RSSI); */
1539: p->RSSI = d[offset];
1540:
1541: p->status = d[offset + 1];
1542:
1543: /* copy in the data */
1544: p->datalen = i-(offset + 1);
1545: for (;i>offset + 1;i--) p->data[i-(offset + 2)] = d[i];
1546:
1547: /* ##### */
1548: /* if: 16 / 64bit I/O recieve */
1549: } else if ((t == XBEE_64BIT_IO) ||
1550:           (t == XBEE_16BIT_IO)) {
1551:     int offset;
1552:     if (t == XBEE_64BIT_IO) { /* 64bit */
1553:         p->type = xbee_64bitIO;
1554:
1555:         p->sAddr64 = TRUE;
1556:
1557:         p->Addr64[0] = d[0];
1558:         p->Addr64[1] = d[1];
1559:         p->Addr64[2] = d[2];
1560:         p->Addr64[3] = d[3];
1561:         p->Addr64[4] = d[4];
1562:         p->Addr64[5] = d[5];
1563:         p->Addr64[6] = d[6];
1564:         p->Addr64[7] = d[7];
1565:
1566:         offset = 8;
1567:         p->samples = d[10];
1568:     } else { /* 16bit */
1569:         p->type = xbee_16bitIO;
1570:
1571:         p->sAddr64 = FALSE;
1572:
1573:         p->Addr16[0] = d[0];
1574:         p->Addr16[1] = d[1];
1575:
1576:         offset = 2;
1577:         p->samples = d[4];
1578:     }
1579:     if (p->samples > 1) {
1580:         p = Xrealloc(p, sizeof(xbee_pkt) + (sizeof(xbee_sample) * (p->samples - 1)));
1581:     }
1582:     if (xbee.log) {
1583:         xbee_log("Packet type: %d-bit RX I/O Data (0x%02X)\n",((t == XBEE_64BIT_IO)?64:16),t);
1584:         xbee_logc("%d-bit Address: ",((t == XBEE_64BIT_IO)?64:16));
1585:         for (j = 0; j < offset; j++) {
1586:             fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
1587:         }
1588:         xbee_logcf();
1589:         xbee_log("RSSI: -%ddB",d[offset]);
1590:         if (d[9] & 0x02) xbee_log("Options: Address Broadcast");
1591:         if (d[9] & 0x02) xbee_log("Options: PAN Broadcast");
1592:         xbee_log("Samples: %d",d[offset + 2]);
1593:     }
1594:     i = offset + 5;
1595:
1596: /* never returns data */
1597: p->datalen = 0;
1598:
1599: p->dataPkt = FALSE;
1600: p->txStatusPkt = FALSE;
1601: p->modemStatusPkt = FALSE;
1602: p->remoteATPkt = FALSE;
1603: p->IOPkt = TRUE;
1604:
1605: /* save the RSSI / signal strength
1606:    this can be used with printf as:
1607:    printf("-%ddB\n",p->RSSI); */
1608: p->RSSI = d[offset];
1609:
1610: p->status = d[offset + 1];
1611:
1612: /* each sample is split into its own packet here, for simplicity */
1613: for (o = 0; o < p->samples; o++) {
1614:     if (xbee.log) {
1615:         xbee_log("--- Sample %3d -----", o);

```

```

1616:     }
1617:
1618:     /* parse the io data */
1619:     i = xbee_parse_io(p, d, offset + 3, i, o);
1620: }
1621: if (xbee.log) {
1622:     xbee_log("-----");
1623: }
1624:
1625: /* ##### */
1626: /* if: Unknown */
1627: } else {
1628:     if (xbee.log) {
1629:         xbee_log("Packet type: Unknown (0x%02X)", t);
1630:     }
1631:     p->type = xbee_unknown;
1632: }
1633: p->next = NULL;
1634:
1635: /* lock the connection mutex */
1636: xbee_mutex_lock(xbee.conmutex);
1637:
1638: con = xbee.conlist;
1639: hasCon = 0;
1640: while (con) {
1641:     if (xbee_matchpktcon(p, con)) {
1642:         hasCon = 1;
1643:         break;
1644:     }
1645:     con = con->next;
1646: }
1647:
1648: /* unlock the connection mutex */
1649: xbee_mutex_unlock(xbee.conmutex);
1650:
1651: /* if the packet doesn't have a connection, don't add it! */
1652: if (!hasCon) {
1653:     Xfree(p);
1654:     if (xbee.log) {
1655:         xbee_log("Connectionless packet... discarding!");
1656:     }
1657:     continue;
1658: }
1659:
1660: /* if the connection has a callback function then it is passed the packet
1661:    and the packet is not added to the list */
1662: if (con && con->callback) {
1663: #ifdef __GNUC__
1664:     pthread_t t;
1665: #else
1666:     HANDLE t;
1667: #endif
1668:     t_callback_list *l, *q;
1669:
1670:     xbee_mutex_lock(con->callbackListmutex);
1671:     l = con->callbackList;
1672:     q = NULL;
1673:     while (l) {
1674:         q = l;
1675:         l = l->next;
1676:     }
1677:     l = Xcalloc(sizeof(t_callback_list));
1678:     l->pkt = p;
1679:     if (!con->callbackList) {
1680:         con->callbackList = l;
1681:     } else {
1682:         q->next = l;
1683:     }
1684:     xbee_mutex_unlock(con->callbackListmutex);
1685:
1686:     xbee_log("Using callback function!");
1687:     xbee_log("  info block @ 0x%08X", l);
1688:     xbee_log("  function   @ 0x%08X", con->callback);
1689:     xbee_log("  connection @ 0x%08X", con);
1690:     xbee_log("  packet     @ 0x%08X", p);
1691:
1692:     /* if the callback thread not still running, then start a new one! */
1693:     if (!xbee_mutex_trylock(con->callbackmutex)) {
1694:         xbee_log("Starting new callback thread!");
1695:         xbee_thread_create(t, xbee_callbackWrapper, con);
1696:     } else {
1697:         xbee_log("Using existing new callback thread");
1698:     }
1699:     continue;
1700: }

```

```

1701:
1702:     /* lock the packet mutex, so we can safely add the packet to the list */
1703:     xbee_mutex_lock(xbee.pktmutex);
1704:
1705:     /* if: the list is empty */
1706:     if (!xbee.pktlist) {
1707:         /* start the list! */
1708:         xbee.pktlist = p;
1709:     } else if (xbee.pktlast) {
1710:         /* add the packet to the end */
1711:         xbee.pktlast->next = p;
1712:     } else {
1713:         /* pktlast wasnt set... look for the end and then set it */
1714:         i = 0;
1715:         q = xbee.pktlist;
1716:         while (q->next) {
1717:             q = q->next;
1718:             i++;
1719:         }
1720:         q->next = p;
1721:         xbee.pktcount = i;
1722:     }
1723:     xbee.pktlast = p;
1724:     xbee.pktcount++;
1725:
1726:     /* unlock the packet mutex */
1727:     xbee_mutex_unlock(xbee.pktmutex);
1728:
1729:     if (xbee.log) {
1730:         xbee_log("-----");
1731:         xbee_log("Packets: %d", xbee.pktcount);
1732:     }
1733:
1734:     p = q = NULL;
1735: }
1736: return 0;
1737: }
1738: static void xbee_callbackWrapper(xbee_con *con) {
1739:     xbee_pkt *pkt;
1740:     t_callback_list *temp;
1741:     /* dont forget! the callback mutex is already locked... by the parent thread :) */
1742:
1743:     xbee_mutex_lock(con->callbackListmutex);
1744:     while (con->callbackList) {
1745:         temp = con->callbackList;
1746:         /* get the packet */
1747:         pkt = temp->pkt;
1748:         /* shift the list along 1 */
1749:         con->callbackList = temp->next;
1750:         Xfree(temp);
1751:         xbee_mutex_unlock(con->callbackListmutex);
1752:
1753:         xbee_log("Starting callback function...");
1754:         con->callback(con, pkt);
1755:         xbee_log("Callback complete!");
1756:         Xfree(pkt);
1757:
1758:         xbee_mutex_lock(con->callbackListmutex);
1759:     }
1760:     xbee_mutex_unlock(con->callbackListmutex);
1761:
1762:     xbee_log("Callback thread ending...");
1763:     /* releasing the thread mutex is the last thing we do! */
1764:     xbee_mutex_unlock(con->callbackmutex);
1765:
1766:     if (con->destroySelf) {
1767:         xbee_endcon2(&con, 1);
1768:     }
1769: }
1770:
1771: /* #####
1772:     xbee_getbyte - INTERNAL
1773:     waits for an escaped byte of data */
1774: static unsigned char xbee_getbyte(void) {
1775:     unsigned char c;
1776:
1777:     ISREADY;
1778:
1779:     /* take a byte */
1780:     c = xbee_getrawbyte();
1781:     /* if its escaped, take another and un-escape */
1782:     if (c == 0x7D) c = xbee_getrawbyte() ^ 0x20;
1783:
1784:     return (c & 0xFF);
1785: }

```

```
1786:
1787: /* #####
1788: xbee_getrawbyte - INTERNAL
1789: waits for a raw byte of data */
1790: static unsigned char xbee_getrawbyte(void) {
1791:     int ret;
1792:     unsigned char c = 0x00;
1793:
1794:     ISREADY;
1795:
1796:     /* the loop is just incase there actually isnt a byte there to be read... */
1797:     do {
1798:         /* wait for a read to be possible */
1799:         if ((ret = xbee_select(NULL)) == -1) {
1800:             perror("libxbee:xbee_getrawbyte()");
1801:             exit(1);
1802:         }
1803:         if (!xbee.listenrun) break;
1804:         if (ret == 0) continue;
1805:
1806:         /* read 1 character */
1807:         xbee_read(&c,1);
1808: #ifdef _WIN32 /* ---- */
1809:         ret = xbee.ttyr;
1810:         if (ret == 0) {
1811:             usleep(10);
1812:             continue;
1813:         }
1814: #endif /* ----- */
1815:     } while (0);
1816:
1817:     return (c & 0xFF);
1818: }
1819:
1820: /* #####
1821: xbee_send_pkt - INTERNAL
1822: sends a complete packet of data */
1823: static void xbee_send_pkt(t_data *pkt) {
1824:     ISREADY;
1825:
1826:     /* lock the send mutex */
1827:     xbee_mutex_lock(xbee.sendmutex);
1828:
1829:     /* write and flush the data */
1830:     xbee_write(pkt->data,pkt->length);
1831:
1832:     /* unlock the mutex */
1833:     xbee_mutex_unlock(xbee.sendmutex);
1834:
1835:     if (xbee.log) {
1836:         int i,x,y;
1837:         /* prints packet in hex byte-by-byte */
1838:         xbee_logc("TX Packet:");
1839:         for (i=0,x=0,y=0;i<pkt->length;i++,x--) {
1840:             if (x == 0) {
1841:                 fprintf(xbee.log,"\n 0x%04X | ",y);
1842:                 x = 0x8;
1843:                 y += x;
1844:             }
1845:             if (x == 4) {
1846:                 fprintf(xbee.log," ");
1847:             }
1848:             fprintf(xbee.log,"0x%02X ",pkt->data[i]);
1849:         }
1850:         xbee_logcf();
1851:     }
1852:
1853:     /* free the packet */
1854:     Xfree(pkt);
1855: }
1856:
1857: /* #####
1858: xbee_make_pkt - INTERNAL
1859: adds delimiter field
1860: calculates length and checksum
1861: escapes bytes */
1862: static t_data *xbee_make_pkt(unsigned char *data, int length) {
1863:     t_data *pkt;
1864:     unsigned int l, i, o, t, x, m;
1865:     char d = 0;
1866:
1867:     ISREADY;
1868:
1869:     /* check the data given isnt too long
1870:     100 bytes maximum payload + 12 bytes header information */
```

```
1871:  if (length > 100 + 12) return NULL;
1872:
1873:  /* calculate the length of the whole packet
1874:   start, length (MSB), length (LSB), DATA, checksum */
1875:  l = 3 + length + 1;
1876:
1877:  /* prepare memory */
1878:  pkt = Xcalloc(sizeof(t_data));
1879:
1880:  /* put start byte on */
1881:  pkt->data[0] = 0x7E;
1882:
1883:  /* copy data into packet */
1884:  for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
1885:      /* if: its time for the checksum */
1886:      if (i == length) d = M8((0xFF - M8(t)));
1887:      /* if: its time for the high length byte */
1888:      else if (m == 1) d = M8(length >> 8);
1889:      /* if: its time for the low length byte */
1890:      else if (m == 2) d = M8(length);
1891:      /* if: its time for the normal data */
1892:      else if (m > 2) d = data[i];
1893:
1894:      x = 0;
1895:      /* check for any escapes needed */
1896:      if ((d == 0x11) || /* XON */
1897:          (d == 0x13) || /* XOFF */
1898:          (d == 0x7D) || /* Escape */
1899:          (d == 0x7E)) { /* Frame Delimiter */
1900:          l++;
1901:          pkt->data[o++] = 0x7D;
1902:          x = 1;
1903:      }
1904:
1905:      /* move data in */
1906:      pkt->data[o] = ((!x)?d:d^0x20);
1907:      if (m > 2) {
1908:          i++;
1909:          t += d;
1910:      }
1911:  }
1912:
1913:  /* remember the length */
1914:  pkt->length = l;
1915:
1916:  return pkt;
1917: }
```