

```
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 466 2011-01-12 18:24:42Z attie@attie.co.uk $";
21: char svn_rev[128] = "\0";
22:
23: #include "api.h"
24:
25: void ISREADY(xbee_hnd xbee) {
26:     if (!xbee || !xbee->xbee_ready) {
27:         if (stderr) fprintf(stderr, "libxbee: Run xbee_setup() first!...\n");
28: #ifdef _WIN32
29:         MessageBox(0, "Run xbee_setup() first!...", "libxbee", MB_OK);
30: #endif
31:         exit(1);
32:     }
33: }
34:
35: const char *xbee_svn_version(void) {
36:     if (svn_rev[0] == '\0') {
37:         char *t;
38:         sprintf(svn_rev, "r%s", &SVN_REV[11]);
39:         t = strrchr(svn_rev, ' ');
40:         if (t) {
41:             t[0] = '\0';
42:         }
43:     }
44:     return svn_rev;
45: }
46:
47: const char *xbee_build_info(void) {
48:     return "Built on " __DATE__ " @ " __TIME__ " for " HOST_OS;
49: }
50:
51: /* ##### */
52: /* ### Memory Handling ##### */
53: /* ##### */
54:
55: /* malloc wrapper function */
56: static void *Xmalloc2(xbee_hnd xbee, size_t size) {
57:     void *t;
58:     t = malloc(size);
59:     if (!t) {
60:         /* uhoh... thats pretty bad... */
61:         xbee_perror("libxbee:malloc()");
62:         exit(1);
63:     }
64:     return t;
65: }
66:
67: /* calloc wrapper function */
68: static void *Xcalloc2(xbee_hnd xbee, size_t size) {
69:     void *t;
70:     t = calloc(1, size);
71:     if (!t) {
72:         /* uhoh... thats pretty bad... */
73:         xbee_perror("libxbee:calloc()");
74:         exit(1);
75:     }
76:     return t;
77: }
78:
79: /* realloc wrapper function */
80: static void *Xrealloc2(xbee_hnd xbee, void *ptr, size_t size) {
81:     void *t;
82:     t = realloc(ptr, size);
83:     if (!t) {
84:         /* uhoh... thats pretty bad... */
85:         fprintf(stderr, "libxbee:realloc(): Returned NULL\n");
```

```

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

```

```

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

```

```

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

```

```

341:     xbeet = xbeet->next;
342: }
343: }
344: if (default_xbee) xbee_mutex_unlock(xbee_hnd_mutex);
345:
346: /* if the api mode was not 2 to begin with then put it back */
347: if (xbee->oldAPI == 2) {
348:     xbee_log("XBee was already in API mode 2, no need to reset");
349:     ret = 0;
350: } else {
351:     int to = 5;
352:
353:     con = _xbee_newcon(xbee, 'I', xbee_localAT);
354:     con->callback = NULL;
355:     con->waitForACK = 1;
356:     _xbee_senddata(xbee, con, "AP%c", xbee->oldAPI);
357:
358:     pkt = NULL;
359:
360:     while (!pkt && to-- > 0) {
361:         pkt = _xbee_getpacketwait(xbee, con);
362:     }
363:     if (pkt) {
364:         ret = pkt->status;
365:         Xfree(pkt);
366:     }
367:     _xbee_endcon(xbee, con);
368: }
369:
370: /* xbee_* functions may no longer run... */
371: xbee->xbee_ready = 0;
372:
373: /* nullify everything */
374:
375: /* stop listening for data... either after timeout or next char read which ever is first */
376: xbee->run = 0;
377:
378: xbee_thread_cancel(xbee->listent, 0);
379: xbee_thread_join(xbee->listent);
380:
381: xbee_thread_cancel(xbee->threadt, 0);
382: xbee_thread_join(xbee->threadt);
383:
384: /* free all connections */
385: con = xbee->conlist;
386: xbee->conlist = NULL;
387: while (con) {
388:     ncon = con->next;
389:     Xfree(con);
390:     con = ncon;
391: }
392:
393: /* free all packets */
394: xbee->pktlast = NULL;
395: pkt = xbee->pktlist;
396: xbee->pktlist = NULL;
397: while (pkt) {
398:     npkt = pkt->next;
399:     Xfree(pkt);
400:     pkt = npkt;
401: }
402:
403: /* destroy mutexes */
404: xbee_mutex_destroy(xbee->conmutex);
405: xbee_mutex_destroy(xbee->pktmutex);
406: xbee_mutex_destroy(xbee->sendmutex);
407:
408: /* close the serial port */
409: Xfree(xbee->path);
410: if (xbee->tty) xbee_close(xbee->tty);
411: #ifdef __GNUC__ /* ---- */
412: if (xbee->ttyfd) close(xbee->ttyfd);
413: #endif /* ----- */
414:
415: /* close log and tty */
416: if (xbee->log) {
417:     i = 0;
418:     xbeet = default_xbeet;
419:     while (xbeet) {
420:         if (xbeet->log == xbee->log) i++;
421:         xbeet = xbeet->next;
422:     }
423:     if (i > 0) xbee_log("%d others are using this log file... leaving it open", i);
424:     xbee_log("libxbee instance stopped!");
425:     fflush(xbee->log);

```

```

426:     if (i == 0) xbee_close(xbee->log);
427: }
428: xbee_mutex_destroy(xbee->logmutex);
429:
430: Xfree(xbee);
431:
432: return ret;
433: }
434:
435: /* #####
436: xbee_setup
437: opens xbee serial port & creates xbee listen thread
438: the xbee must be configured for API mode 2
439: THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
440: int xbee_setup(char *path, int baudrate) {
441:     return xbee_setuplogAPI(path,baudrate,0,0,0);
442: }
443: xbee_hnd _xbee_setup(char *path, int baudrate) {
444:     return _xbee_setuplogAPI(path,baudrate,0,0,0);
445: }
446: int xbee_setuplog(char *path, int baudrate, int logfd) {
447:     return xbee_setuplogAPI(path,baudrate,logfd,0,0);
448: }
449: xbee_hnd _xbee_setuplog(char *path, int baudrate, int logfd) {
450:     return _xbee_setuplogAPI(path,baudrate,logfd,0,0);
451: }
452: int xbee_setupAPI(char *path, int baudrate, char cmdSeq, int cmdTime) {
453:     return xbee_setuplogAPI(path,baudrate,0,cmdSeq,cmdTime);
454: }
455: xbee_hnd _xbee_setupAPI(char *path, int baudrate, char cmdSeq, int cmdTime) {
456:     return _xbee_setuplogAPI(path,baudrate,0,cmdSeq,cmdTime);
457: }
458: int xbee_setuplogAPI(char *path, int baudrate, int logfd, char cmdSeq, int cmdTime) {
459:     if (default_xbee) return 0;
460:     default_xbee = _xbee_setuplogAPI(path,baudrate,logfd,cmdSeq,cmdTime);
461:     return (default_xbee?0:-1);
462: }
463: xbee_hnd _xbee_setuplogAPI(char *path, int baudrate, int logfd, char cmdSeq, int cmdTime) {
464:     t_LTinfo info;
465:     int ret;
466:     xbee_hnd xbee = NULL;;
467:
468:     /* create a new instance */
469:     xbee = Xcalloc(sizeof(struct xbee_hnd));
470:
471: #ifdef DEBUG
472:     /* logfd or stderr */
473:     xbee->logfd = ((logfd)?logfd:2);
474: #else
475:     xbee->logfd = logfd;
476: #endif
477:     xbee_mutex_init(xbee->logmutex);
478:     if (xbee->logfd) {
479:         xbee->log = fdopen(xbee->logfd,"w");
480:         if (!xbee->log) {
481:             /* errno == 9 is bad file descriptor (probably not provided) */
482:             if (errno != 9) xbee_perror("xbee_setup(): Failed opening logfile");
483:             xbee->logfd = 0;
484:         } else {
485: #ifdef __GNUC__ /* ---- */
486:             /* set to line buffer - ensure lines are written to file when complete */
487:             setvbuf(xbee->log,NULL,_IOLBF,BUFSIZ);
488: #else /* ----- */
489:             /* Win32 is rubbish... so we have to completely disable buffering... */
490:             setvbuf(xbee->log,NULL,_IONBF,BUFSIZ);
491: #endif /* ----- */
492:         }
493:     }
494:
495:     xbee_log("-----");
496:     xbee_log("libxbee Starting...");
497:     xbee_log("SVN Info: %s",xbee_svn_version());
498:     xbee_log("Build Info: %s",xbee_build_info());
499:     xbee_log("-----");
500:
501:     /* setup the connection stuff */
502:     xbee->conlist = NULL;
503:
504:     /* setup the packet stuff */
505:     xbee->pktlist = NULL;
506:     xbee->pktlast = NULL;
507:     xbee->pktcount = 0;
508:     xbee->run = 1;
509:
510:     /* setup the mutexes */

```

```

511:     if (xbee_mutex_init(xbee->conmutex)) {
512:         xbee_perror("xbee_setup():xbee_mutex_init(conmutex)");
513:         if (xbee->log) xbee_close(xbee->log);
514:         Xfree(xbee);
515:         return NULL;
516:     }
517:     if (xbee_mutex_init(xbee->pktmutex)) {
518:         xbee_perror("xbee_setup():xbee_mutex_init(pktmutex)");
519:         if (xbee->log) xbee_close(xbee->log);
520:         xbee_mutex_destroy(xbee->conmutex);
521:         Xfree(xbee);
522:         return NULL;
523:     }
524:     if (xbee_mutex_init(xbee->sendmutex)) {
525:         xbee_perror("xbee_setup():xbee_mutex_init(sendmutex)");
526:         if (xbee->log) xbee_close(xbee->log);
527:         xbee_mutex_destroy(xbee->conmutex);
528:         xbee_mutex_destroy(xbee->pktmutex);
529:         Xfree(xbee);
530:         return NULL;
531:     }
532:
533:     /* take a copy of the XBee device path */
534:     if ((xbee->path = Xmalloc(sizeof(char) * (strlen(path) + 1))) == NULL) {
535:         xbee_perror("xbee_setup():Xmalloc(path)");
536:         if (xbee->log) xbee_close(xbee->log);
537:         xbee_mutex_destroy(xbee->conmutex);
538:         xbee_mutex_destroy(xbee->pktmutex);
539:         xbee_mutex_destroy(xbee->sendmutex);
540:         Xfree(xbee);
541:         return NULL;
542:     }
543:     strcpy(xbee->path, path);
544:     if (xbee->log) xbee_log("Opening serial port '%s'...", xbee->path);
545:
546:     /* call the relevant init function */
547:     if ((ret = init_serial(xbee, baudrate)) != 0) {
548:         xbee_log("Something failed while opening the serial port...");
549:         if (xbee->log) xbee_close(xbee->log);
550:         xbee_mutex_destroy(xbee->conmutex);
551:         xbee_mutex_destroy(xbee->pktmutex);
552:         xbee_mutex_destroy(xbee->sendmutex);
553:         Xfree(xbee->path);
554:         Xfree(xbee);
555:         return NULL;
556:     }
557:
558:     /* when xbee_end() is called, if this is not 2 then ATAP will be set to this value */
559:     xbee->oldAPI = 2;
560:     xbee->cmdSeq = cmdSeq;
561:     xbee->cmdTime = cmdTime;
562:     if (xbee->cmdSeq && xbee->cmdTime) {
563:         if (xbee_startAPI(xbee)) {
564:             if (xbee->log) {
565:                 xbee_log("Couldn't communicate with XBee...");
566:                 xbee_close(xbee->log);
567:             }
568:             xbee_mutex_destroy(xbee->conmutex);
569:             xbee_mutex_destroy(xbee->pktmutex);
570:             xbee_mutex_destroy(xbee->sendmutex);
571:             Xfree(xbee->path);
572: #ifdef __GNUC__ /* ---- */
573:             close(xbee->ttyfd);
574: #endif /* ----- */
575:             xbee_close(xbee->tty);
576:             Xfree(xbee);
577:             return NULL;
578:         }
579:     }
580:
581:     /* allow the listen thread to start */
582:     xbee->xbee_ready = -1;
583:
584:     /* can start xbee_listen thread now */
585:     info.xbee = xbee;
586:     if (xbee_thread_create(xbee->listent, xbee_listen_wrapper, &info)) {
587:         xbee_perror("xbee_setup():xbee_thread_create(listent)");
588:         if (xbee->log) xbee_close(xbee->log);
589:         xbee_mutex_destroy(xbee->conmutex);
590:         xbee_mutex_destroy(xbee->pktmutex);
591:         xbee_mutex_destroy(xbee->sendmutex);
592:         Xfree(xbee->path);
593: #ifdef __GNUC__ /* ---- */
594:         close(xbee->ttyfd);
595: #endif /* ----- */

```

```

596:     xbee_close(xbee->tty);
597:     Xfree(xbee);
598:     return NULL;
599: }
600:
601: /* can start xbee_thread_watch thread thread now */
602: if (xbee_thread_create(xbee->threadt, xbee_thread_watch, &info)) {
603:     xbee_perror("xbee_setup():xbee_thread_create(threadt)");
604:     if (xbee->log) xbee_close(xbee->log);
605:     xbee_mutex_destroy(xbee->conmutex);
606:     xbee_mutex_destroy(xbee->pktnutex);
607:     xbee_mutex_destroy(xbee->sendmutex);
608:     Xfree(xbee->path);
609: #ifdef __GNUC__ /* ---- */
610:     close(xbee->ttyfd);
611: #endif /* ----- */
612:     xbee_close(xbee->tty);
613:     Xfree(xbee);
614:     return NULL;
615: }
616:
617: usleep(500);
618: while (xbee->xbee_ready != -2) {
619:     usleep(500);
620:     xbee_log("Waiting for xbee_listen() to be ready...");
621: }
622:
623: /* allow other functions to be used! */
624: xbee->xbee_ready = 1;
625:
626: xbee_log("Linking xbee instance...");
627: if (!default_xbee) {
628:     xbee_mutex_init(xbee_hnd_mutex);
629:     xbee_mutex_lock(xbee_hnd_mutex);
630:     default_xbee = xbee;
631:     xbee_mutex_unlock(xbee_hnd_mutex);
632: } else {
633:     xbee_hnd xbeet;
634:     xbee_mutex_lock(xbee_hnd_mutex);
635:     xbeet = default_xbee;
636:     while (xbeet->next) {
637:         xbeet = xbeet->next;
638:     }
639:     xbeet->next = xbee;
640:     xbee_mutex_unlock(xbee_hnd_mutex);
641: }
642:
643: xbee_log("libxbee: Started!");
644:
645: return xbee;
646: }
647:
648: /* #####
649: xbee_con
650: produces a connection to the specified device and frameID
651: if a connection had already been made, then this connection will be returned */
652: xbee_con *xbee_newcon(unsigned char frameID, xbee_types type, ...) {
653:     xbee_con *ret;
654:     va_list ap;
655:
656:     /* xbee_vsenddata() wants a va_list... */
657:     va_start(ap, type);
658:     /* hand it over :) */
659:     ret = _xbee_vnewcon(default_xbee, frameID, type, ap);
660:     va_end(ap);
661:     return ret;
662: }
663: xbee_con *_xbee_newcon(xbee_hnd xbee, unsigned char frameID, xbee_types type, ...) {
664:     xbee_con *ret;
665:     va_list ap;
666:
667:     /* xbee_vsenddata() wants a va_list... */
668:     va_start(ap, type);
669:     /* hand it over :) */
670:     ret = _xbee_vnewcon(xbee, frameID, type, ap);
671:     va_end(ap);
672:     return ret;
673: }
674: xbee_con *_xbee_vnewcon(xbee_hnd xbee, unsigned char frameID, xbee_types type, va_list ap) {
675:     xbee_con *con, *ocon;
676:     unsigned char tAddr[8];
677:     int t;
678:     int i;
679:
680:     ISREADY(xbee);

```



```

681:
682: if (!type || type == xbee_unknown) type = xbee_localAT; /* default to local AT */
683: else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */
684:
685: /* if: 64 bit address expected (2 ints) */
686: if ((type == xbee_64bitRemoteAT) ||
687:     (type == xbee_64bitData) ||
688:     (type == xbee_64bitIO)) {
689:     t = va_arg(ap, int);
690:     tAddr[0] = (t >> 24) & 0xFF;
691:     tAddr[1] = (t >> 16) & 0xFF;
692:     tAddr[2] = (t >> 8) & 0xFF;
693:     tAddr[3] = (t >> 0) & 0xFF;
694:     t = va_arg(ap, int);
695:     tAddr[4] = (t >> 24) & 0xFF;
696:     tAddr[5] = (t >> 16) & 0xFF;
697:     tAddr[6] = (t >> 8) & 0xFF;
698:     tAddr[7] = (t >> 0) & 0xFF;
699:
700:     /* if: 16 bit address expected (1 int) */
701: } else if ((type == xbee_16bitRemoteAT) ||
702:           (type == xbee_16bitData) ||
703:           (type == xbee_16bitIO)) {
704:     t = va_arg(ap, int);
705:     tAddr[0] = (t >> 8) & 0xFF;
706:     tAddr[1] = (t >> 0) & 0xFF;
707:     tAddr[2] = 0;
708:     tAddr[3] = 0;
709:     tAddr[4] = 0;
710:     tAddr[5] = 0;
711:     tAddr[6] = 0;
712:     tAddr[7] = 0;
713:
714:     /* otherwise clear the address */
715: } else {
716:     memset(tAddr, 0, 8);
717: }
718:
719: /* lock the connection mutex */
720: xbee_mutex_lock(xbee->conmutex);
721:
722: /* are there any connections? */
723: if (xbee->conlist) {
724:     con = xbee->conlist;
725:     while (con) {
726:         /* if: looking for a modemStatus, and the types match! */
727:         if ((type == xbee_modemStatus) &&
728:             (con->type == type)) {
729:             xbee_mutex_unlock(xbee->conmutex);
730:             return con;
731:
732:             /* if: looking for a txStatus and frameIDs match! */
733:         } else if ((type == xbee_txStatus) &&
734:                    (con->type == type) &&
735:                    (frameID == con->frameID)) {
736:             xbee_mutex_unlock(xbee->conmutex);
737:             return con;
738:
739:             /* if: looking for a localAT, and the frameIDs match! */
740:         } else if ((type == xbee_localAT) &&
741:                    (con->type == type) &&
742:                    (frameID == con->frameID)) {
743:             xbee_mutex_unlock(xbee->conmutex);
744:             return con;
745:
746:             /* if: connection types match, the frameIDs match, and the addresses match! */
747:         } else if ((type == con->type) &&
748:                    (frameID == con->frameID) &&
749:                    (!memcmp(tAddr, con->tAddr, 8))) {
750:             xbee_mutex_unlock(xbee->conmutex);
751:             return con;
752:         }
753:
754:         /* if there are more, move along, dont want to loose that last item! */
755:         if (con->next == NULL) break;
756:         con = con->next;
757:     }
758:
759:     /* keep hold of the last connection... we will need to link it up later */
760:     ocon = con;
761: }
762:
763: /* create a new connection and set its attributes */
764: con = Xcalloc(sizeof(xbee_con));
765: con->type = type;

```

```

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

```

```

851:     ISREADY(xbee);
852:
853:     /* lock the packet mutex */
854:     xbee_mutex_lock(xbee->pktmutex);
855:
856:     /* if: there are packets */
857:     if ((p = xbee->pktlist) != NULL) {
858:         r = NULL;
859:         /* get all packets for this connection */
860:         do {
861:             /* does the packet match the connection? */
862:             if (xbee_matchpktcon(xbee,p,con)) {
863:                 /* if it was the first packet */
864:                 if (!r) {
865:                     /* move the chain along */
866:                     xbee->pktlist = p->next;
867:                 } else {
868:                     /* otherwise relink the list */
869:                     r->next = p->next;
870:                 }
871:                 xbee->pktpcount--;
872:
873:                 /* free this packet! */
874:                 n = p->next;
875:                 Xfree(p);
876:                 /* move on */
877:                 p = n;
878:             } else {
879:                 /* move on */
880:                 r = p;
881:                 p = p->next;
882:             }
883:         } while (p);
884:         xbee->pktlast = r;
885:     }
886:
887:     /* unlock the packet mutex */
888:     xbee_mutex_unlock(xbee->pktmutex);
889: }
890:
891: /* #####
892: xbee_endcon
893: close the unwanted connection
894: free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
895: void xbee_endcon2(xbee_con **con, int alreadyUnlinked) {
896:     _xbee_endcon2(default_xbee, con, alreadyUnlinked);
897: }
898: void _xbee_endcon2(xbee_hnd xbee, xbee_con **con, int alreadyUnlinked) {
899:     xbee_con *t, *u;
900:
901:     ISREADY(xbee);
902:
903:     /* lock the connection mutex */
904:     xbee_mutex_lock(xbee->conmutex);
905:
906:     u = t = xbee->conlist;
907:     while (t && t != *con) {
908:         u = t;
909:         t = t->next;
910:     }
911:     if (!t) {
912:         /* this could be true if coming from the destroySelf signal... */
913:         if (!alreadyUnlinked) {
914:             /* invalid connection given... */
915:             if (xbee->log) {
916:                 xbee_log("Attempted to close invalid connection...");
917:             }
918:             /* unlock the connection mutex */
919:             xbee_mutex_unlock(xbee->conmutex);
920:             return;
921:         }
922:     } else {
923:         /* extract this connection from the list */
924:         if (t == xbee->conlist) {
925:             xbee->conlist = t->next;
926:         } else {
927:             u->next = t->next;
928:         }
929:     }
930:
931:     /* unlock the connection mutex */
932:     xbee_mutex_unlock(xbee->conmutex);
933:
934:     /* check if a callback thread is running... */
935:     if (t->callback && xbee_mutex_trylock(t->callbackmutex)) {

```

```

936:      /* if it is running... tell it to destroy the connection on completion */
937:      xbee_log("Attempted to close a connection with active callbacks... "
938:              "Connection will be destroyed when callbacks have completed...");
939:      t->destroySelf = 1;
940:      return;
941:  }
942:
943:  /* remove all packets for this connection */
944:  _xbee_flushcon(xbee,t);
945:
946:  /* destroy the callback mutex */
947:  xbee_mutex_destroy(t->callbackmutex);
948:  xbee_mutex_destroy(t->callbackListmutex);
949:  xbee_mutex_destroy(t->Txmutex);
950:  xbee_sem_destroy(t->waitForACKsem);
951:
952:  /* free the connection! */
953:  Xfree(*con);
954: }
955:
956: /* #####
957:  xbee_senddata
958:  send the specified data to the provided connection */
959: int xbee_senddata(xbee_con *con, char *format, ...) {
960:     int ret;
961:     va_list ap;
962:
963:     /* xbee_vsenddata() wants a va_list... */
964:     va_start(ap, format);
965:     /* hand it over :) */
966:     ret = _xbee_vsenddata(default_xbee, con, format, ap);
967:     va_end(ap);
968:     return ret;
969: }
970: int _xbee_senddata(xbee_hnd xbee, xbee_con *con, char *format, ...) {
971:     int ret;
972:     va_list ap;
973:
974:     /* xbee_vsenddata() wants a va_list... */
975:     va_start(ap, format);
976:     /* hand it over :) */
977:     ret = _xbee_vsenddata(xbee, con, format, ap);
978:     va_end(ap);
979:     return ret;
980: }
981:
982: int xbee_vsenddata(xbee_con *con, char *format, va_list ap) {
983:     return _xbee_vsenddata(default_xbee, con, format, ap);
984: }
985: int _xbee_vsenddata(xbee_hnd xbee, xbee_con *con, char *format, va_list ap) {
986:     unsigned char data[128]; /* max payload is 100 bytes... plus a bit of fluff... */
987:     int length;
988:
989:     /* make up the data and keep the length, its possible there are nulls in there */
990:     length = vsprintf((char *)data, 128, format, ap);
991:
992:     /* hand it over :) */
993:     return _xbee_nsenddata(xbee, con, (char *)data, length);
994: }
995:
996: /* returns:
997:     1 - if NAC was recieved
998:     0 - if packet was successfully sent (or just sent if waitForACK is off)
999:    -1 - if there was an error building the packet
1000:    -2 - if the connection type was unknown */
1001: int xbee_nsenddata(xbee_con *con, char *data, int length) {
1002:     return _xbee_nsenddata(default_xbee, con, data, length);
1003: }
1004: int _xbee_nsenddata(xbee_hnd xbee, xbee_con *con, char *data, int length) {
1005:     t_data *pkt;
1006:     int i;
1007:     unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
1008:
1009:     ISREADY(xbee);
1010:
1011:     if (!con) return -1;
1012:     if (con->type == xbee_unknown) return -1;
1013:     if (length > 127) return -1;
1014:
1015:     if (xbee->log) {
1016:         xbee_log("==== TX Packet =====");
1017:         xbee_logc("Connection Type: ");
1018:         switch (con->type) {
1019:             case xbee_unknown:      fprintf(xbee->log,"Unknown"); break;
1020:             case xbee_localAT:      fprintf(xbee->log,"Local AT"); break;

```

```

1021:     case xbee_remoteAT:      fprintf(xbee->log, "Remote AT"); break;
1022:     case xbee_16bitRemoteAT: fprintf(xbee->log, "Remote AT (16-bit)"); break;
1023:     case xbee_64bitRemoteAT: fprintf(xbee->log, "Remote AT (64-bit)"); break;
1024:     case xbee_16bitData:     fprintf(xbee->log, "Data (16-bit)"); break;
1025:     case xbee_64bitData:     fprintf(xbee->log, "Data (64-bit)"); break;
1026:     case xbee_16bitIO:       fprintf(xbee->log, "IO (16-bit)"); break;
1027:     case xbee_64bitIO:       fprintf(xbee->log, "IO (64-bit)"); break;
1028:     case xbee_txStatus:      fprintf(xbee->log, "Tx Status"); break;
1029:     case xbee_modemStatus:   fprintf(xbee->log, "Modem Status"); break;
1030:     }
1031:     xbee_logcf(xbee);
1032:     xbee_logc("Destination: ");
1033:     for (i=0; i<(con->tAddr64?8:2); i++) {
1034:         fprintf(xbee->log, (i?"%02X":"%02X"), con->tAddr[i]);
1035:     }
1036:     xbee_logcf(xbee);
1037:     xbee_log("Length: %d", length);
1038:     for (i=0; i<length; i++) {
1039:         xbee_logc("%3d | 0x%02X ", i, (unsigned char)data[i]);
1040:         if ((data[i] > 32) && (data[i] < 127)) {
1041:             fprintf(xbee->log, "'%c'", data[i]);
1042:         } else {
1043:             fprintf(xbee->log, " _");
1044:         }
1045:         xbee_logcf(xbee);
1046:     }
1047: }
1048:
1049: /* ##### */
1050: /* if: local AT */
1051: if (con->type == xbee_localAT) {
1052:     /* AT commands are 2 chars long (plus optional parameter) */
1053:     if (length < 2) return -1;
1054:
1055:     /* use the command? */
1056:     buf[0] = ((!con->atQueue)?XBEE_LOCAL_ATREQ:XBEE_LOCAL_ATQUE);
1057:     buf[1] = con->frameID;
1058:
1059:     /* copy in the data */
1060:     for (i=0; i<length; i++) {
1061:         buf[i+2] = data[i];
1062:     }
1063:
1064:     /* setup the packet */
1065:     pkt = xbee_make_pkt(xbee, buf, i+2);
1066:     /* send it on */
1067:     return xbee_send_pkt(xbee, pkt, con);
1068:
1069:     /* ##### */
1070:     /* if: remote AT */
1071: } else if ((con->type == xbee_16bitRemoteAT) ||
1072:           (con->type == xbee_64bitRemoteAT)) {
1073:     if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter) */
1074:     buf[0] = XBEE_REMOTE_ATREQ;
1075:     buf[1] = con->frameID;
1076:
1077:     /* copy in the relevant address */
1078:     if (con->tAddr64) {
1079:         memcpy(&buf[2], con->tAddr, 8);
1080:         buf[10] = 0xFF;
1081:         buf[11] = 0xFE;
1082:     } else {
1083:         memset(&buf[2], 0, 8);
1084:         memcpy(&buf[10], con->tAddr, 2);
1085:     }
1086:     /* queue the command? */
1087:     buf[12] = ((!con->atQueue)?0x02:0x00);
1088:
1089:     /* copy in the data */
1090:     for (i=0; i<length; i++) {
1091:         buf[i+13] = data[i];
1092:     }
1093:
1094:     /* setup the packet */
1095:     pkt = xbee_make_pkt(xbee, buf, i+13);
1096:     /* send it on */
1097:     return xbee_send_pkt(xbee, pkt, con);
1098:
1099:     /* ##### */
1100:     /* if: 16 or 64bit Data */
1101: } else if ((con->type == xbee_16bitData) ||
1102:           (con->type == xbee_64bitData)) {
1103:     int offset;
1104:
1105:     /* if: 16bit Data */

```

```

1106:     if (con->type == xbee_16bitData) {
1107:         buf[0] = XBEE_16BIT_DATATX;
1108:         offset = 5;
1109:         /* copy in the address */
1110:         memcpy(&buf[2], con->tAddr, 2);
1111:
1112:         /* if: 64bit Data */
1113:     } else { /* 64bit Data */
1114:         buf[0] = XBEE_64BIT_DATATX;
1115:         offset = 11;
1116:         /* copy in the address */
1117:         memcpy(&buf[2], con->tAddr, 8);
1118:     }
1119:
1120:     /* copy frameID */
1121:     buf[1] = con->frameID;
1122:
1123:     /* disable ack? broadcast? */
1124:     buf[offset-1] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
1125:
1126:     /* copy in the data */
1127:     for (i=0; i<length; i++) {
1128:         buf[i+offset] = data[i];
1129:     }
1130:
1131:     /* setup the packet */
1132:     pkt = xbee_make_pkt(xbee, buf, i+offset);
1133:     /* send it on */
1134:     return xbee_send_pkt(xbee, pkt, con);
1135:
1136:     /* ##### */
1137:     /* if: I/O */
1138: } else if ((con->type == xbee_64bitIO) ||
1139:           (con->type == xbee_16bitIO)) {
1140:     /* not currently implemented... is it even allowed? */
1141:     if (xbee->log) {
1142:         xbee_log("***** TODO *****\n");
1143:     }
1144: }
1145:
1146: return -2;
1147: }
1148:
1149: /* #####
1150: xbee_getpacket
1151: retrieves the next packet destined for the given connection
1152: once the packet has been retrieved, it is removed for the list! */
1153: xbee_pkt *xbee_getpacketwait(xbee_con *con) {
1154:     return _xbee_getpacketwait(default_xbee, con);
1155: }
1156: xbee_pkt *_xbee_getpacketwait(xbee_hnd xbee, xbee_con *con) {
1157:     xbee_pkt *p = NULL;
1158:     int i = 20;
1159:
1160:     /* 50ms * 20 = 1 second */
1161:     for (; i; i--) {
1162:         p = _xbee_getpacket(xbee, con);
1163:         if (p) break;
1164:         usleep(50000); /* 50ms */
1165:     }
1166:
1167:     return p;
1168: }
1169: xbee_pkt *xbee_getpacket(xbee_con *con) {
1170:     return _xbee_getpacket(default_xbee, con);
1171: }
1172: xbee_pkt *_xbee_getpacket(xbee_hnd xbee, xbee_con *con) {
1173:     xbee_pkt *l, *p, *q;
1174:
1175:     ISREADY(xbee);
1176:
1177:     /* lock the packet mutex */
1178:     xbee_mutex_lock(xbee->pktmutex);
1179:
1180:     /* if: there are no packets */
1181:     if ((p = xbee->pktlist) == NULL) {
1182:         xbee_mutex_unlock(xbee->pktmutex);
1183:         /*if (xbee->log) {
1184:             xbee_log("No packets available...");
1185:         }*/
1186:         return NULL;
1187:     }
1188:
1189:     l = NULL;
1190:     q = NULL;

```

```

1191:  /* get the first available packet for this connection */
1192:  do {
1193:      /* does the packet match the connection? */
1194:      if (xbee_matchpktcon(xbee, p, con)) {
1195:          q = p;
1196:          break;
1197:      }
1198:      /* move on */
1199:      l = p;
1200:      p = p->next;
1201:  } while (p);
1202:
1203:  /* if: no packet was found */
1204:  if (!q) {
1205:      xbee_mutex_unlock(xbee->pkmutex);
1206:      if (xbee->log) {
1207:          struct timeval tv;
1208:          xbee_log("---- Get Packet -----");
1209:          gettimeofday(&tv, NULL);
1210:          xbee_log("Didn't get a packet @ %ld.%06ld", tv.tv_sec, tv.tv_usec);
1211:      }
1212:      return NULL;
1213:  }
1214:
1215:  /* if it was the first packet */
1216:  if (l) {
1217:      /* relink the list */
1218:      l->next = p->next;
1219:      if (!l->next) xbee->pktlast = l;
1220:  } else {
1221:      /* move the chain along */
1222:      xbee->pktlist = p->next;
1223:      if (!xbee->pktlist) {
1224:          xbee->pktlast = NULL;
1225:      } else if (!xbee->pktlist->next) {
1226:          xbee->pktlast = xbee->pktlist;
1227:      }
1228:  }
1229:  xbee->pktcount--;
1230:
1231:  /* unlink this packet from the chain! */
1232:  q->next = NULL;
1233:
1234:  if (xbee->log) {
1235:      struct timeval tv;
1236:      xbee_log("---- Get Packet -----");
1237:      gettimeofday(&tv, NULL);
1238:      xbee_log("Got a packet @ %ld.%06ld", tv.tv_sec, tv.tv_usec);
1239:      xbee_log("Packets left: %d", xbee->pktcount);
1240:  }
1241:
1242:  /* unlock the packet mutex */
1243:  xbee_mutex_unlock(xbee->pkmutex);
1244:
1245:  /* and return the packet (must be free'd by caller!) */
1246:  return q;
1247: }
1248:
1249: /* #####
1250:  xbee_matchpktcon - INTERNAL
1251:  checks if the packet matches the connection */
1252: static int xbee_matchpktcon(xbee_hnd xbee, xbee_pkt *pkt, xbee_con *con) {
1253:     /* if: the connection type matches the packet type OR
1254:      the connection is 16/64bit remote AT, and the packet is a remote AT response */
1255:     if ((pkt->type == con->type) || /* -- */
1256:         ((pkt->type == xbee_remoteAT) && /* -- */
1257:          ((con->type == xbee_16bitRemoteAT) ||
1258:           (con->type == xbee_64bitRemoteAT)))) {
1259:
1260:
1261:         /* if: is a modem status (there can only be 1 modem status connection) */
1262:         if (pkt->type == xbee_modemStatus) return 1;
1263:
1264:         /* if: the packet is a txStatus or localAT and the frameIDs match */
1265:         if ((pkt->type == xbee_txStatus) ||
1266:             (pkt->type == xbee_localAT)) {
1267:             if (pkt->frameID == con->frameID) {
1268:                 return 1;
1269:             }
1270:         }
1271:         /* if: the packet was sent as a 16bit remoteAT, and the 16bit addresss match */
1272:     } else if ((pkt->type == xbee_remoteAT) &&
1273:                (con->type == xbee_16bitRemoteAT) &&
1274:                !memcmp(pkt->Addr16, con->tAddr, 2)) {
1275:         return 1;
1276:     }
1277:     /* if: the packet was sent as a 64bit remoteAT, and the 64bit addresss match */

```



```

1276:     } else if ((pkt->type == xbee_remoteAT) &&
1277:                (con->type == xbee_64bitRemoteAT) &&
1278:                !memcmp(pkt->Addr64, con->tAddr, 8)) {
1279:         return 1;
1280:         /* if: the packet is 64bit addressed, and the addresses match */
1281:     } else if (pkt->sAddr64 && !memcmp(pkt->Addr64, con->tAddr, 8)) {
1282:         return 1;
1283:         /* if: the packet is 16bit addressed, and the addresses match */
1284:     } else if (!pkt->sAddr64 && !memcmp(pkt->Addr16, con->tAddr, 2)) {
1285:         return 1;
1286:     }
1287: }
1288: return 0;
1289: }
1290:
1291: /* #####
1292: xbee_parse_io - INTERNAL
1293: parses the data given into the packet io information */
1294: static int xbee_parse_io(xbee_hnd xbee, xbee_pkt *p, unsigned char *d,
1295:                          int maskOffset, int sampleOffset, int sample) {
1296:     xbee_sample *s = &(p->Iodata[sample]);
1297:
1298:     /* copy in the I/O data mask */
1299:     s->Iomask = (((d[maskOffset]<<8) | d[maskOffset + 1]) & 0x7FFF);
1300:
1301:     /* copy in the digital I/O data */
1302:     s->Iodigital = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x01FF);
1303:
1304:     /* advance over the digital data, if its there */
1305:     sampleOffset += ((s->Iomask & 0x01FF)?2:0);
1306:
1307:     /* copy in the analog I/O data */
1308:     if (s->Iomask & 0x0200) {
1309:         s->IOanalog[0] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1310:         sampleOffset+=2;
1311:     }
1312:     if (s->Iomask & 0x0400) {
1313:         s->IOanalog[1] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1314:         sampleOffset+=2;
1315:     }
1316:     if (s->Iomask & 0x0800) {
1317:         s->IOanalog[2] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1318:         sampleOffset+=2;
1319:     }
1320:     if (s->Iomask & 0x1000) {
1321:         s->IOanalog[3] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1322:         sampleOffset+=2;
1323:     }
1324:     if (s->Iomask & 0x2000) {
1325:         s->IOanalog[4] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1326:         sampleOffset+=2;
1327:     }
1328:     if (s->Iomask & 0x4000) {
1329:         s->IOanalog[5] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1330:         sampleOffset+=2;
1331:     }
1332:
1333:     if (xbee->log) {
1334:         if (s->Iomask & 0x0001)
1335:             xbee_log("Digital 0: %c", ((s->Iodigital & 0x0001)?'1':'0'));
1336:         if (s->Iomask & 0x0002)
1337:             xbee_log("Digital 1: %c", ((s->Iodigital & 0x0002)?'1':'0'));
1338:         if (s->Iomask & 0x0004)
1339:             xbee_log("Digital 2: %c", ((s->Iodigital & 0x0004)?'1':'0'));
1340:         if (s->Iomask & 0x0008)
1341:             xbee_log("Digital 3: %c", ((s->Iodigital & 0x0008)?'1':'0'));
1342:         if (s->Iomask & 0x0010)
1343:             xbee_log("Digital 4: %c", ((s->Iodigital & 0x0010)?'1':'0'));
1344:         if (s->Iomask & 0x0020)
1345:             xbee_log("Digital 5: %c", ((s->Iodigital & 0x0020)?'1':'0'));
1346:         if (s->Iomask & 0x0040)
1347:             xbee_log("Digital 6: %c", ((s->Iodigital & 0x0040)?'1':'0'));
1348:         if (s->Iomask & 0x0080)
1349:             xbee_log("Digital 7: %c", ((s->Iodigital & 0x0080)?'1':'0'));
1350:         if (s->Iomask & 0x0100)
1351:             xbee_log("Digital 8: %c", ((s->Iodigital & 0x0100)?'1':'0'));
1352:         if (s->Iomask & 0x0200)
1353:             xbee_log("Analog 0: %d ( %.2fv)", s->IOanalog[0], (3.3/1023)*s->IOanalog[0]);
1354:         if (s->Iomask & 0x0400)
1355:             xbee_log("Analog 1: %d ( %.2fv)", s->IOanalog[1], (3.3/1023)*s->IOanalog[1]);
1356:         if (s->Iomask & 0x0800)
1357:             xbee_log("Analog 2: %d ( %.2fv)", s->IOanalog[2], (3.3/1023)*s->IOanalog[2]);
1358:         if (s->Iomask & 0x1000)
1359:             xbee_log("Analog 3: %d ( %.2fv)", s->IOanalog[3], (3.3/1023)*s->IOanalog[3]);
1360:         if (s->Iomask & 0x2000)

```



```

1361:     xbee_log("Analog 4: %d (~%.2fv)", s->IOanalog[4], (3.3/1023)*s->IOanalog[4]);
1362:     if (s->IOMask & 0x4000)
1363:         xbee_log("Analog 5: %d (~%.2fv)", s->IOanalog[5], (3.3/1023)*s->IOanalog[5]);
1364: }
1365:
1366: return sampleOffset;
1367: }
1368:
1369: /* #####
1370: xbee_listen_stop
1371: stops the listen thread after the current packet has been processed */
1372: void xbee_listen_stop(xbee_hnd xbee) {
1373:     ISREADY(xbee);
1374:     xbee->run = 0;
1375: }
1376:
1377: /* #####
1378: xbee_listen_wrapper - INTERNAL
1379: the xbee_listen wrapper. Prints an error when xbee_listen ends */
1380: static void xbee_listen_wrapper(t_LTinfo *info) {
1381:     xbee_hnd xbee;
1382:     int ret;
1383:     xbee = info->xbee;
1384:     /* just falls out if the proper 'go-ahead' isn't given */
1385:     if (xbee->xbee_ready != -1) return;
1386:     /* now allow the parent to continue */
1387:     xbee->xbee_ready = -2;
1388:
1389: #ifdef _WIN32 /* ---- */
1390:     /* win32 requires this delay... no idea why */
1391:     usleep(1000000);
1392: #endif /* ----- */
1393:
1394:     while (xbee->run) {
1395:         info->i = -1;
1396:         ret = xbee_listen(xbee, info);
1397:         if (!xbee->run) break;
1398:         xbee_log("xbee_listen() returned [%d]... Restarting in 250ms!", ret);
1399:         usleep(25000);
1400:     }
1401: }
1402:
1403: /* xbee_listen - INTERNAL
1404: the xbee_listen thread
1405: reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
1406: static int xbee_listen(xbee_hnd xbee, t_LTinfo *info) {
1407:     unsigned char c, t, d[1024];
1408:     unsigned int l, i, chksum, o;
1409:     int j;
1410:     xbee_pkt *p, *q;
1411:     xbee_con *con;
1412:     int hasCon;
1413:
1414:     /* just falls out if the proper 'go-ahead' isn't given */
1415:     if (info->i != -1) return -1;
1416:     /* do this forever :) */
1417:     while (xbee->run) {
1418:         /* wait for a valid start byte */
1419:         if ((c = xbee_getrawbyte(xbee)) != 0x7E) {
1420:             if (xbee->log) xbee_log("***** Unexpected byte (0x%02X)... *****", c);
1421:             continue;
1422:         }
1423:         if (!xbee->run) return 0;
1424:
1425:         if (xbee->log) {
1426:             struct timeval tv;
1427:             xbee_log("==== RX Packet =====");
1428:             gettimeofday(&tv, NULL);
1429:             xbee_log("Got a packet @ %ld.%06ld", tv.tv_sec, tv.tv_usec);
1430:         }
1431:
1432:         /* get the length */
1433:         l = xbee_getbyte(xbee) << 8;
1434:         l += xbee_getbyte(xbee);
1435:
1436:         /* check it is a valid length... */
1437:         if (!l) {
1438:             if (xbee->log) {
1439:                 xbee_log("Recived zero length packet!");
1440:             }
1441:             continue;
1442:         }
1443:         if (l > 100) {
1444:             if (xbee->log) {
1445:                 xbee_log("Recived oversized packet! Length: %d", l - 1);

```

```

1446:     }
1447: }
1448: if (l > sizeof(d) - 1) {
1449:     if (xbee->log) {
1450:         xbee_log("Recieved packet larger than buffer! Discarding...");
1451:     }
1452:     continue;
1453: }
1454:
1455: if (xbee->log) {
1456:     xbee_log("Length: %d", l - 1);
1457: }
1458:
1459: /* get the packet type */
1460: t = xbee_getbyte(xbee);
1461:
1462: /* start the checksum */
1463: chksum = t;
1464:
1465: /* suck in all the data */
1466: for (i = 0; l > 1 && i < 128; l--, i++) {
1467:     /* get an unescaped byte */
1468:     c = xbee_getbyte(xbee);
1469:     d[i] = c;
1470:     chksum += c;
1471:     if (xbee->log) {
1472:         xbee_logc("%3d | 0x%02X | ", i, c);
1473:         if ((c > 32) && (c < 127)) fprintf(xbee->log, "'%c'", c); else fprintf(xbee->log, " _ ");
1474:
1475:         if ((t == XBEE_LOCAL_AT && i == 4) ||
1476:             (t == XBEE_REMOTE_AT && i == 14) ||
1477:             (t == XBEE_64BIT_DATARX && i == 10) ||
1478:             (t == XBEE_16BIT_DATARX && i == 4) ||
1479:             (t == XBEE_64BIT_IO && i == 13) ||
1480:             (t == XBEE_16BIT_IO && i == 7)) {
1481:             /* mark the beginning of the 'data' bytes */
1482:             fprintf(xbee->log, " <-- data starts");
1483:         } else if (t == XBEE_64BIT_IO) {
1484:             if (i == 10) fprintf(xbee->log, " <-- sample count");
1485:             else if (i == 11) fprintf(xbee->log, " <-- mask (msb)");
1486:             else if (i == 12) fprintf(xbee->log, " <-- mask (lsb)");
1487:         } else if (t == XBEE_16BIT_IO) {
1488:             if (i == 4) fprintf(xbee->log, " <-- sample count");
1489:             else if (i == 5) fprintf(xbee->log, " <-- mask (msb)");
1490:             else if (i == 6) fprintf(xbee->log, " <-- mask (lsb)");
1491:         }
1492:         xbee_logcf(xbee);
1493:     }
1494: }
1495: i--; /* it went up too many times!... */
1496:
1497: /* add the checksum */
1498: chksum += xbee_getbyte(xbee);
1499:
1500: /* check if the whole packet was recieved, or something else occured... unlikely... */
1501: if (l > 1) {
1502:     if (xbee->log) {
1503:         xbee_log("Didn't get whole packet... :(");
1504:     }
1505:     continue;
1506: }
1507:
1508: /* check the checksum */
1509: if ((chksum & 0xFF) != 0xFF) {
1510:     if (xbee->log) {
1511:         chksum &= 0xFF;
1512:         xbee_log("Invalid Checksum: 0x%02X", chksum);
1513:     }
1514:     continue;
1515: }
1516:
1517: /* make a new packet */
1518: p = Xcalloc(sizeof(xbee_pkt));
1519: q = NULL;
1520: p->datalen = 0;
1521:
1522: /* ##### */
1523: /* if: modem status */
1524: if (t == XBEE_MODEM_STATUS) {
1525:     if (xbee->log) {
1526:         xbee_log("Packet type: Modem Status (0x8A)");
1527:         xbee_logc("Event: ");
1528:         switch (d[0]) {
1529:             case 0x00: fprintf(xbee->log, "Hardware reset"); break;
1530:             case 0x01: fprintf(xbee->log, "Watchdog timer reset"); break;

```

```

1531:         case 0x02: fprintf(xbee->log,"Associated"); break;
1532:         case 0x03: fprintf(xbee->log,"Disassociated"); break;
1533:         case 0x04: fprintf(xbee->log,"Synchronization lost"); break;
1534:         case 0x05: fprintf(xbee->log,"Coordinator realignment"); break;
1535:         case 0x06: fprintf(xbee->log,"Coordinator started"); break;
1536:     }
1537:     fprintf(xbee->log,"... (0x%02X)",d[0]);
1538:     xbee_logcf(xbee);
1539: }
1540: p->type = xbee_modemStatus;
1541:
1542: p->sAddr64 = FALSE;
1543: p->dataPkt = FALSE;
1544: p->txStatusPkt = FALSE;
1545: p->modemStatusPkt = TRUE;
1546: p->remoteATPkt = FALSE;
1547: p->IOPkt = FALSE;
1548:
1549: /* modem status can only ever give 1 'data' byte */
1550: p->datalen = 1;
1551: p->data[0] = d[0];
1552:
1553: /* ##### */
1554: /* if: local AT response */
1555: } else if (t == XBEE_LOCAL_AT) {
1556:     if (xbee->log) {
1557:         xbee_log("Packet type: Local AT Response (0x88)");
1558:         xbee_log("FrameID: 0x%02X",d[0]);
1559:         xbee_log("AT Command: %c%c",d[1],d[2]);
1560:         xbee_logc("Status: ");
1561:         if (d[3] == 0x00) fprintf(xbee->log,"OK");
1562:         else if (d[3] == 0x01) fprintf(xbee->log,"Error");
1563:         else if (d[3] == 0x02) fprintf(xbee->log,"Invalid Command");
1564:         else if (d[3] == 0x03) fprintf(xbee->log,"Invalid Parameter");
1565:         fprintf(xbee->log," (0x%02X)",d[3]);
1566:         xbee_logcf(xbee);
1567:     }
1568:     p->type = xbee_localAT;
1569:
1570:     p->sAddr64 = FALSE;
1571:     p->dataPkt = FALSE;
1572:     p->txStatusPkt = FALSE;
1573:     p->modemStatusPkt = FALSE;
1574:     p->remoteATPkt = FALSE;
1575:     p->IOPkt = FALSE;
1576:
1577:     p->frameID = d[0];
1578:     p->atCmd[0] = d[1];
1579:     p->atCmd[1] = d[2];
1580:
1581:     p->status = d[3];
1582:
1583:     /* copy in the data */
1584:     p->datalen = i-3;
1585:     for (;i>3;i--) p->data[i-4] = d[i];
1586:
1587:     /* ##### */
1588:     /* if: remote AT response */
1589: } else if (t == XBEE_REMOTE_AT) {
1590:     if (xbee->log) {
1591:         xbee_log("Packet type: Remote AT Response (0x97)");
1592:         xbee_log("FrameID: 0x%02X",d[0]);
1593:         xbee_logc("64-bit Address: ");
1594:         for (j=0;j<8;j++) {
1595:             fprintf(xbee->log,(j?"%02X":"%02X"),d[1+j]);
1596:         }
1597:         xbee_logcf(xbee);
1598:         xbee_logc("16-bit Address: ");
1599:         for (j=0;j<2;j++) {
1600:             fprintf(xbee->log,(j?"%02X":"%02X"),d[9+j]);
1601:         }
1602:         xbee_logcf(xbee);
1603:         xbee_log("AT Command: %c%c",d[11],d[12]);
1604:         xbee_logc("Status: ");
1605:         if (d[13] == 0x00) fprintf(xbee->log,"OK");
1606:         else if (d[13] == 0x01) fprintf(xbee->log,"Error");
1607:         else if (d[13] == 0x02) fprintf(xbee->log,"Invalid Command");
1608:         else if (d[13] == 0x03) fprintf(xbee->log,"Invalid Parameter");
1609:         else if (d[13] == 0x04) fprintf(xbee->log,"No Response");
1610:         fprintf(xbee->log," (0x%02X)",d[13]);
1611:         xbee_logcf(xbee);
1612:     }
1613:     p->type = xbee_remoteAT;
1614:
1615:     p->sAddr64 = FALSE;

```

```

1616:     p->dataPkt = FALSE;
1617:     p->txStatusPkt = FALSE;
1618:     p->modemStatusPkt = FALSE;
1619:     p->remoteATPkt = TRUE;
1620:     p->IOPkt = FALSE;
1621:
1622:     p->frameID = d[0];
1623:
1624:     p->Addr64[0] = d[1];
1625:     p->Addr64[1] = d[2];
1626:     p->Addr64[2] = d[3];
1627:     p->Addr64[3] = d[4];
1628:     p->Addr64[4] = d[5];
1629:     p->Addr64[5] = d[6];
1630:     p->Addr64[6] = d[7];
1631:     p->Addr64[7] = d[8];
1632:
1633:     p->Addr16[0] = d[9];
1634:     p->Addr16[1] = d[10];
1635:
1636:     p->atCmd[0] = d[11];
1637:     p->atCmd[1] = d[12];
1638:
1639:     p->status = d[13];
1640:
1641:     p->samples = 1;
1642:
1643:     if (p->status == 0x00 && p->atCmd[0] == 'I' && p->atCmd[1] == 'S') {
1644:         /* parse the io data */
1645:         xbee_log("---- Sample -----");
1646:         xbee_parse_io(xbee, p, d, 15, 17, 0);
1647:         xbee_log("-----");
1648:     } else {
1649:         /* copy in the data */
1650:         p->datalen = i-13;
1651:         for (;i>13;i--) p->data[i-14] = d[i];
1652:     }
1653:
1654:     /* ##### */
1655:     /* if: TX status */
1656: } else if (t == XBEE_TX_STATUS) {
1657:     if (xbee->log) {
1658:         xbee_log("Packet type: TX Status Report (0x89)");
1659:         xbee_log("FrameID: 0x%02X",d[0]);
1660:         xbee_logc("Status: ");
1661:         if (d[1] == 0x00) fprintf(xbee->log,"Success");
1662:         else if (d[1] == 0x01) fprintf(xbee->log,"No ACK");
1663:         else if (d[1] == 0x02) fprintf(xbee->log,"CCA Failure");
1664:         else if (d[1] == 0x03) fprintf(xbee->log,"Purged");
1665:         fprintf(xbee->log," (0x%02X)",d[1]);
1666:         xbee_logcf(xbee);
1667:     }
1668:     p->type = xbee_txStatus;
1669:
1670:     p->sAddr64 = FALSE;
1671:     p->dataPkt = FALSE;
1672:     p->txStatusPkt = TRUE;
1673:     p->modemStatusPkt = FALSE;
1674:     p->remoteATPkt = FALSE;
1675:     p->IOPkt = FALSE;
1676:
1677:     p->frameID = d[0];
1678:
1679:     p->status = d[1];
1680:
1681:     /* never returns data */
1682:     p->datalen = 0;
1683:
1684:     /* check for any connections waiting for a status update */
1685:     /* lock the connection mutex */
1686:     xbee_mutex_lock(xbee->conmutex);
1687:     xbee_log("Looking for a connection that wants a status update...");
1688:     con = xbee->conlist;
1689:     while (con) {
1690:         if ((con->frameID == p->frameID) &&
1691:             (con->ACKstatus == 0xFF)) {
1692:             xbee_log("Found @ 0x%08X!",con);
1693:             con->ACKstatus = p->status;
1694:             xbee_sem_post(con->waitForACKsem);
1695:         }
1696:         con = con->next;
1697:     }
1698:
1699:     /* unlock the connection mutex */
1700:     xbee_mutex_unlock(xbee->conmutex);

```

```

1701:
1702:     /* ##### */
1703:     /* if: 16 / 64bit data recieve */
1704: } else if ((t == XBEE_64BIT_DATARX) ||
1705:           (t == XBEE_16BIT_DATARX)) {
1706:     int offset;
1707:     if (t == XBEE_64BIT_DATARX) { /* 64bit */
1708:         offset = 8;
1709:     } else { /* 16bit */
1710:         offset = 2;
1711:     }
1712:     if (xbee->log) {
1713:         xbee_log("Packet type: %d-bit RX Data (0x%02X)", ((t == XBEE_64BIT_DATARX)?64:16), t);
1714:         xbee_logc("%d-bit Address: ", ((t == XBEE_64BIT_DATARX)?64:16));
1715:         for (j=0; j<offset; j++) {
1716:             fprintf(xbee->log, (j?"%02X":"%02X"), d[j]);
1717:         }
1718:         xbee_logcf(xbee);
1719:         xbee_log("RSSI: -%ddb", d[offset]);
1720:         if (d[offset + 1] & 0x02) xbee_log("Options: Address Broadcast");
1721:         if (d[offset + 1] & 0x03) xbee_log("Options: PAN Broadcast");
1722:     }
1723:     p->dataPkt = TRUE;
1724:     p->txStatusPkt = FALSE;
1725:     p->modemStatusPkt = FALSE;
1726:     p->remoteATPkt = FALSE;
1727:     p->IOPkt = FALSE;
1728:
1729:     if (t == XBEE_64BIT_DATARX) { /* 64bit */
1730:         p->type = xbee_64bitData;
1731:
1732:         p->sAddr64 = TRUE;
1733:
1734:         p->Addr64[0] = d[0];
1735:         p->Addr64[1] = d[1];
1736:         p->Addr64[2] = d[2];
1737:         p->Addr64[3] = d[3];
1738:         p->Addr64[4] = d[4];
1739:         p->Addr64[5] = d[5];
1740:         p->Addr64[6] = d[6];
1741:         p->Addr64[7] = d[7];
1742:     } else { /* 16bit */
1743:         p->type = xbee_16bitData;
1744:
1745:         p->sAddr64 = FALSE;
1746:
1747:         p->Addr16[0] = d[0];
1748:         p->Addr16[1] = d[1];
1749:     }
1750:
1751:     /* save the RSSI / signal strength
1752:     this can be used with printf as:
1753:     printf("-%ddb\n", p->RSSI); */
1754:     p->RSSI = d[offset];
1755:
1756:     p->status = d[offset + 1];
1757:
1758:     /* copy in the data */
1759:     p->datalen = i-(offset + 1);
1760:     for (; i>offset + 1; i--) p->data[i-(offset + 2)] = d[i];
1761:
1762:     /* ##### */
1763:     /* if: 16 / 64bit I/O recieve */
1764: } else if ((t == XBEE_64BIT_IO) ||
1765:           (t == XBEE_16BIT_IO)) {
1766:     int offset, i2;
1767:     if (t == XBEE_64BIT_IO) { /* 64bit */
1768:         p->type = xbee_64bitIO;
1769:
1770:         p->sAddr64 = TRUE;
1771:
1772:         p->Addr64[0] = d[0];
1773:         p->Addr64[1] = d[1];
1774:         p->Addr64[2] = d[2];
1775:         p->Addr64[3] = d[3];
1776:         p->Addr64[4] = d[4];
1777:         p->Addr64[5] = d[5];
1778:         p->Addr64[6] = d[6];
1779:         p->Addr64[7] = d[7];
1780:
1781:         offset = 8;
1782:         p->samples = d[10];
1783:     } else { /* 16bit */
1784:         p->type = xbee_16bitIO;
1785:

```

```

1786:         p->sAddr64 = FALSE;
1787:
1788:         p->Addr16[0] = d[0];
1789:         p->Addr16[1] = d[1];
1790:
1791:         offset = 2;
1792:         p->samples = d[4];
1793:     }
1794:     if (p->samples > 1) {
1795:         p = Xrealloc(p, sizeof(xbee_pkt) + (sizeof(xbee_sample) * (p->samples - 1)));
1796:     }
1797:     if (xbee->log) {
1798:         xbee_log("Packet type: %d-bit RX I/O Data (0x%02X)", ((t == XBEE_64BIT_IO)?64:16), t);
1799:         xbee_logc("%d-bit Address: ", ((t == XBEE_64BIT_IO)?64:16));
1800:         for (j = 0; j < offset; j++) {
1801:             fprintf(xbee->log, (j?"%02X":"%02X"), d[j]);
1802:         }
1803:         xbee_logcf(xbee);
1804:         xbee_log("RSSI: -%ddb", d[offset]);
1805:         if (d[9] & 0x02) xbee_log("Options: Address Broadcast");
1806:         if (d[9] & 0x02) xbee_log("Options: PAN Broadcast");
1807:         xbee_log("Samples: %d", d[offset + 2]);
1808:     }
1809:     i2 = offset + 5;
1810:
1811:     /* never returns data */
1812:     p->datalen = 0;
1813:
1814:     p->dataPkt = FALSE;
1815:     p->txStatusPkt = FALSE;
1816:     p->modemStatusPkt = FALSE;
1817:     p->remoteATPkt = FALSE;
1818:     p->IOPkt = TRUE;
1819:
1820:     /* save the RSSI / signal strength
1821:      this can be used with printf as:
1822:      printf("-%ddb\n", p->RSSI); */
1823:     p->RSSI = d[offset];
1824:
1825:     p->status = d[offset + 1];
1826:
1827:     /* each sample is split into its own packet here, for simplicity */
1828:     for (o = 0; o < p->samples; o++) {
1829:         if (i2 >= i) {
1830:             xbee_log("Invalid I/O data! Actually contained %d samples...", o);
1831:             p = Xrealloc(p, sizeof(xbee_pkt) + (sizeof(xbee_sample) * ((o>1)?o:1)));
1832:             p->samples = o;
1833:             break;
1834:         }
1835:         xbee_log("--- Sample %3d -----", o);
1836:
1837:         /* parse the io data */
1838:         i2 = xbee_parse_io(xbee, p, d, offset + 3, i2, o);
1839:     }
1840:     xbee_log("-----");
1841:
1842:     /* ##### */
1843:     /* if: Unknown */
1844: } else {
1845:     xbee_log("Packet type: Unknown (0x%02X)", t);
1846:     p->type = xbee_unknown;
1847: }
1848: p->next = NULL;
1849:
1850: /* lock the connection mutex */
1851: xbee_mutex_lock(xbee->conmutex);
1852:
1853: con = xbee->conlist;
1854: hasCon = 0;
1855: while (con) {
1856:     if (xbee_matchpktcon(xbee, p, con)) {
1857:         hasCon = 1;
1858:         break;
1859:     }
1860:     con = con->next;
1861: }
1862:
1863: /* unlock the connection mutex */
1864: xbee_mutex_unlock(xbee->conmutex);
1865:
1866: /* if the packet doesn't have a connection, don't add it! */
1867: if (!hasCon) {
1868:     Xfree(p);
1869:     xbee_log("Connectionless packet... discarding!");
1870:     continue;

```

```

1871:     }
1872:
1873:     /* if the connection has a callback function then it is passed the packet
1874:        and the packet is not added to the list */
1875:     if (con && con->callback) {
1876:         t_callback_list *l, *q;
1877:
1878:         xbee_mutex_lock(con->callbackListmutex);
1879:         l = con->callbackList;
1880:         q = NULL;
1881:         while (l) {
1882:             q = l;
1883:             l = l->next;
1884:         }
1885:         l = Xcalloc(sizeof(t_callback_list));
1886:         l->pkt = p;
1887:         if (!con->callbackList || q == NULL) {
1888:             con->callbackList = l;
1889:         } else {
1890:             q->next = l;
1891:         }
1892:         xbee_mutex_unlock(con->callbackListmutex);
1893:
1894:         xbee_log("Using callback function!");
1895:         xbee_log("  info block @ 0x%08X", l);
1896:         xbee_log("  function   @ 0x%08X", con->callback);
1897:         xbee_log("  connection @ 0x%08X", con);
1898:         xbee_log("  packet     @ 0x%08X", p);
1899:
1900:         /* if the callback thread not still running, then start a new one! */
1901:         if (!xbee_mutex_trylock(con->callbackmutex)) {
1902:             xbee_thread_t t;
1903:             int ret;
1904:             t_threadList *p, *q;
1905:             t_CBinfo info;
1906:             info.xbee = xbee;
1907:             info.con = con;
1908:             xbee_log("Starting new callback thread!");
1909:             if ((ret = xbee_thread_create(t, xbee_callbackWrapper, &info)) != 0) {
1910:                 xbee_mutex_unlock(con->callbackmutex);
1911:                 /* this MAY help... */
1912:                 xbee_sem_post(xbee->threadsem);
1913:                 xbee_log("An error occured while starting thread (%d)... Out of resources?", ret);
1914:                 xbee_log("This packet has been lost!");
1915:                 continue;
1916:             }
1917:             xbee_log("Started thread 0x%08X!", t);
1918:             xbee_mutex_lock(xbee->threadmutex);
1919:             p = xbee->threadList;
1920:             q = NULL;
1921:             while (p) {
1922:                 q = p;
1923:                 p = p->next;
1924:             }
1925:             p = Xcalloc(sizeof(t_threadList));
1926:             if (q == NULL) {
1927:                 xbee->threadList = p;
1928:             } else {
1929:                 q->next = p;
1930:             }
1931:             p->thread = t;
1932:             xbee_mutex_unlock(xbee->threadmutex);
1933:         } else {
1934:             xbee_log("Using existing callback thread... callback has been scheduled.");
1935:         }
1936:         continue;
1937:     }
1938:
1939:     /* lock the packet mutex, so we can safely add the packet to the list */
1940:     xbee_mutex_lock(xbee->pktmutex);
1941:
1942:     /* if: the list is empty */
1943:     if (!xbee->pktlist) {
1944:         /* start the list! */
1945:         xbee->pktlist = p;
1946:     } else if (xbee->pktlast) {
1947:         /* add the packet to the end */
1948:         xbee->pktlast->next = p;
1949:     } else {
1950:         /* pktlast wasnt set... look for the end and then set it */
1951:         i = 0;
1952:         q = xbee->pktlist;
1953:         while (q->next) {
1954:             q = q->next;
1955:             i++;

```



```

1956:     }
1957:     q->next = p;
1958:     xbee->pktpcount = i;
1959: }
1960: xbee->pktpcount = p;
1961: xbee->pktpcount++;
1962:
1963: /* unlock the packet mutex */
1964: xbee_mutex_unlock(xbee->pktpmutex);
1965:
1966: xbee_log("-----");
1967: xbee_log("Packets: %d",xbee->pktpcount);
1968:
1969: p = q = NULL;
1970: }
1971: return 0;
1972: }
1973:
1974: static void xbee_callbackWrapper(t_CBInfo *info) {
1975:     xbee_hnd xbee;
1976:     xbee_con *con;
1977:     xbee_pkt *pkt;
1978:     t_callback_list *temp;
1979:     xbee = info->xbee;
1980:     con = info->con;
1981:     /* dont forget! the callback mutex is already locked... by the parent thread :) */
1982:     xbee_mutex_lock(con->callbackListmutex);
1983:     while (con->callbackList) {
1984:         /* shift the list along 1 */
1985:         temp = con->callbackList;
1986:         con->callbackList = temp->next;
1987:         xbee_mutex_unlock(con->callbackListmutex);
1988:         /* get the packet */
1989:         pkt = temp->pkt;
1990:
1991:         xbee_log("Starting callback function...");
1992:         xbee_log("  info block @ 0x%08X",temp);
1993:         xbee_log("  function @ 0x%08X",con->callback);
1994:         xbee_log("  connection @ 0x%08X",con);
1995:         xbee_log("  packet @ 0x%08X",pkt);
1996:         Xfree(temp);
1997:         con->callback(con,pkt);
1998:         xbee_log("Callback complete!");
1999:         Xfree(pkt);
2000:
2001:         xbee_mutex_lock(con->callbackListmutex);
2002:     }
2003:
2004:     xbee_log("Callback thread ending...");
2005:     /* releasing the thread mutex is the last thing we do! */
2006:     xbee_mutex_unlock(con->callbackmutex);
2007:     xbee_mutex_unlock(con->callbackListmutex);
2008:
2009:     if (con->destroySelf) {
2010:         _xbee_endcon2(xbee,&con,1);
2011:     }
2012:     xbee_sem_post(xbee->threadsem);
2013: }
2014:
2015: /* #####
2016:   xbee_thread_watch - INTERNAL
2017:   watches for dead threads and tidies up */
2018: static void xbee_thread_watch(t_LTInfo *info) {
2019:     xbee_hnd xbee;
2020:
2021:     xbee = info->xbee;
2022:     xbee_mutex_init(xbee->threadmutex);
2023:     xbee_sem_init(xbee->threadsem);
2024:
2025:     while (xbee->run) {
2026:         t_threadList *p, *q;
2027:         xbee_mutex_lock(xbee->threadmutex);
2028:         p = xbee->threadList;
2029:         q = NULL;
2030:
2031:         while (p) {
2032:             if (!(xbee_thread_tryjoin(p->thread))) {
2033:                 xbee_log("Joined with thread 0x%08X...",p->thread);
2034:                 if (p == xbee->threadList) {
2035:                     xbee->threadList = p->next;
2036:                 } else if (q) {
2037:                     q->next = p->next;
2038:                 }
2039:                 free(p);
2040:             } else {

```



```

2041:         q = p;
2042:     }
2043:     p = p->next;
2044: }
2045:
2046: xbee_mutex_unlock(xbee->threadmutex);
2047: xbee_log("Waiting...");
2048: xbee_sem_wait(xbee->threadsem);
2049: usleep(25000); /* 25ms to allow the thread to end before we try to join */
2050: }
2051:
2052: xbee_mutex_destroy(xbee->threadmutex);
2053: xbee_sem_destroy(xbee->threadsem);
2054: }
2055:
2056:
2057: /* #####
2058:  xbee_getbyte - INTERNAL
2059:  waits for an escaped byte of data */
2060: static unsigned char xbee_getbyte(xbee_hnd xbee) {
2061:     unsigned char c;
2062:
2063:     /* take a byte */
2064:     c = xbee_getrawbyte(xbee);
2065:     /* if its escaped, take another and un-escape */
2066:     if (c == 0x7D) c = xbee_getrawbyte(xbee) ^ 0x20;
2067:
2068:     return (c & 0xFF);
2069: }
2070:
2071: /* #####
2072:  xbee_getrawbyte - INTERNAL
2073:  waits for a raw byte of data */
2074: static unsigned char xbee_getrawbyte(xbee_hnd xbee) {
2075:     int ret;
2076:     unsigned char c = 0x00;
2077:
2078:     /* the loop is just incase there actually isnt a byte there to be read... */
2079:     do {
2080:         /* wait for a read to be possible */
2081:         if ((ret = xbee_select(xbee,NULL)) == -1) {
2082:             xbee_perror("libxbee:xbee_getrawbyte()");
2083:             exit(1);
2084:         }
2085:         if (!xbee->run) break;
2086:         if (ret == 0) continue;
2087:
2088:         /* read 1 character */
2089:         if (xbee_read(xbee,&c,1) == 0) {
2090:             /* for some reason no characters were read... */
2091:             if (xbee_ferror(xbee) || xbee_feof(xbee)) {
2092:                 xbee_log("Error or EOF detected");
2093:                 fprintf(stderr,"libxbee:xbee_read(): Error or EOF detected\n");
2094:                 exit(1);
2095:             }
2096:             /* no error... try again */
2097:             usleep(10);
2098:             continue;
2099:         }
2100:     } while (0);
2101:
2102:     return (c & 0xFF);
2103: }
2104:
2105: /* #####
2106:  xbee_send_pkt - INTERNAL
2107:  sends a complete packet of data */
2108: static int xbee_send_pkt(xbee_hnd xbee, t_data *pkt, xbee_con *con) {
2109:     int retval = 0;
2110:
2111:     /* lock connection mutex */
2112:     xbee_mutex_lock(con->Txmutex);
2113:     /* lock the send mutex */
2114:     xbee_mutex_lock(xbee->sendmutex);
2115:
2116:     /* write and flush the data */
2117:     xbee_write(xbee,pkt->data,pkt->length);
2118:
2119:     /* unlock the mutex */
2120:     xbee_mutex_unlock(xbee->sendmutex);
2121:
2122:     if (xbee->log) {
2123:         int i,x,y;
2124:         /* prints packet in hex byte-by-byte */
2125:         xbee_logc("TX Packet:");

```

```

2126:     for (i=0,x=0,y=0;i<pkt->length;i++,x--) {
2127:         if (x == 0) {
2128:             fprintf(xbee->log,"\n  0x%04X | ",y);
2129:             x = 0x8;
2130:             y += x;
2131:         }
2132:         if (x == 4) {
2133:             fprintf(xbee->log," ");
2134:         }
2135:         fprintf(xbee->log,"0x%02X ",pkt->data[i]);
2136:     }
2137:     xbee_logcf(xbee);
2138: }
2139:
2140: if (con->waitForACK &&
2141:     ((con->type == xbee_16bitData) ||
2142:      (con->type == xbee_64bitData))) {
2143:     con->ACKstatus = 0xFF; /* waiting */
2144:     xbee_log("Waiting for ACK/NAK response...");
2145:     xbee_sem_wait1sec(con->waitForACKsem);
2146:     switch (con->ACKstatus) {
2147:         case 0: xbee_log("ACK recieved!"); break;
2148:         case 1: xbee_log("NAK recieved..."); break;
2149:         case 2: xbee_log("CCA failure..."); break;
2150:         case 3: xbee_log("Purged..."); break;
2151:         case 255: default: xbee_log("Timeout...");
2152:     }
2153:     if (con->ACKstatus) retval = 1; /* error */
2154: }
2155:
2156: /* unlock connection mutex */
2157: xbee_mutex_unlock(con->Txmutex);
2158:
2159: /* free the packet */
2160: Xfree(pkt);
2161:
2162: return retval;
2163: }
2164:
2165: /* #####
2166: xbee_make_pkt - INTERNAL
2167: adds delimiter field
2168: calculates length and checksum
2169: escapes bytes */
2170: static t_data *xbee_make_pkt(xbee_hnd xbee, unsigned char *data, int length) {
2171:     t_data *pkt;
2172:     unsigned int l, i, o, t, x, m;
2173:     char d = 0;
2174:
2175:     /* check the data given isnt too long
2176:        100 bytes maximum payload + 12 bytes header information */
2177:     if (length > 100 + 12) return NULL;
2178:
2179:     /* calculate the length of the whole packet
2180:        start, length (MSB), length (LSB), DATA, checksum */
2181:     l = 3 + length + 1;
2182:
2183:     /* prepare memory */
2184:     pkt = Xcalloc(sizeof(t_data));
2185:
2186:     /* put start byte on */
2187:     pkt->data[0] = 0x7E;
2188:
2189:     /* copy data into packet */
2190:     for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
2191:         /* if: its time for the checksum */
2192:         if (i == length) d = M8((0xFF - M8(t)));
2193:         /* if: its time for the high length byte */
2194:         else if (m == 1) d = M8(length >> 8);
2195:         /* if: its time for the low length byte */
2196:         else if (m == 2) d = M8(length);
2197:         /* if: its time for the normal data */
2198:         else if (m > 2) d = data[i];
2199:
2200:         x = 0;
2201:         /* check for any escapes needed */
2202:         if ((d == 0x11) || /* XON */
2203:             (d == 0x13) || /* XOFF */
2204:             (d == 0x7D) || /* Escape */
2205:             (d == 0x7E)) { /* Frame Delimiter */
2206:             l++;
2207:             pkt->data[o++] = 0x7D;
2208:             x = 1;
2209:         }
2210:     }

```

```
2211:      /* move data in */
2212:      pkt->data[o] = ((!x)?d:d^0x20);
2213:      if (m > 2) {
2214:          i++;
2215:          t += d;
2216:      }
2217:  }
2218:
2219:      /* remember the length */
2220:      pkt->length = l;
2221:
2222:      return pkt;
2223: }
```