```c
/*
  libxbee - a C library to aid the use of Digi's Series 1 XBee modules
            running in API mode (AP=2).

  Copyright (C) 2009  Attie Grande (attie@attie.co.uk)

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
const char *SVN_REV = "$Id: api.c 513 2011-06-24 15:45:32Z attie@attie.co.uk $";
char svn_rev[128] = "\0";

#include "api.h"

const char *xbee_svn_version(void) {
  if (svn_rev[0] == '\0') {
    char *t;
    sprintf(svn_rev,"r%s",&SVN_REV[11]);
    t = strrchr(svn_rev,' ');
    if (t) {
      t[0] = '\0';
    }
  }
  return svn_rev;
}

const char *xbee_build_info(void) {
  return "Built on " __DATE__ " @ " __TIME__ " for " HOST_OS;
}

/* ######################################################################## */
/* ### Memory Handling #################################################### */
/* ######################################################################## */

/* malloc wrapper function */
static void *Xmalloc2(xbee_hnd xbee, size_t size) {
  void *t;
  t = malloc(size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    xbee_perror("libxbee:malloc()");
    exit(1);
  }
  return t;
}

/* calloc wrapper function */
static void *Xcalloc2(xbee_hnd xbee, size_t size) {
  void *t;
  t = calloc(1, size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    xbee_perror("libxbee:calloc()");
    exit(1);
  }
  return t;
}

/* realloc wrapper function */
static void *Xrealloc2(xbee_hnd xbee, void *ptr, size_t size) {
  void *t;
  t = realloc(ptr,size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    fprintf(stderr,"libxbee:realloc(): Returned NULL\n");
    exit(1);
  }
  return t;
}

/* free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
static void Xfree2(void **ptr) {
  if (!*ptr) return;
  free(*ptr);
  *ptr = NULL;
```

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

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

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

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

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

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

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

```
681:   else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */
682:
683:   /* if: 64 bit address expected (2 ints) */
684:   if ((type == xbee_64bitRemoteAT) ||
685:       (type == xbee_64bitData) ||
686:       (type == xbee_64bitIO) ||
687:       (type == xbee2_data)) {
688:     t = va_arg(ap, int);
689:     tAddr[0] = (t >> 24) & 0xFF;
690:     tAddr[1] = (t >> 16) & 0xFF;
691:     tAddr[2] = (t >>  8) & 0xFF;
692:     tAddr[3] = (t      ) & 0xFF;
693:     t = va_arg(ap, int);
694:     tAddr[4] = (t >> 24) & 0xFF;
695:     tAddr[5] = (t >> 16) & 0xFF;
696:     tAddr[6] = (t >>  8) & 0xFF;
697:     tAddr[7] = (t      ) & 0xFF;
698:
699:     /* if: 16 bit address expected (1 int) */
700:   } else if ((type == xbee_16bitRemoteAT) ||
701:              (type == xbee_16bitData) ||
702:              (type == xbee_16bitIO)) {
703:     t = va_arg(ap, int);
704:     tAddr[0] = (t >>  8) & 0xFF;
705:     tAddr[1] = (t      ) & 0xFF;
706:     tAddr[2] = 0;
707:     tAddr[3] = 0;
708:     tAddr[4] = 0;
709:     tAddr[5] = 0;
710:     tAddr[6] = 0;
711:     tAddr[7] = 0;
712:
713:     /* otherwise clear the address */
714:   } else {
715:     memset(tAddr,0,8);
716:   }
717:
718:   /* lock the connection mutex */
719:   xbee_mutex_lock(xbee->conmutex);
720:
721:   /* are there any connections? */
722:   if (xbee->conlist) {
723:     con = xbee->conlist;
724:     while (con) {
725:       /* if: looking for a modemStatus, and the types match! */
726:       if ((type == xbee_modemStatus) &&
727:           (con->type == type)) {
728:         xbee_mutex_unlock(xbee->conmutex);
729:         return con;
730:
731:         /* if: looking for a txStatus and frameIDs match! */
732:       } else if ((type == xbee_txStatus) &&
733:                  (con->type == type) &&
734:                  (frameID == con->frameID)) {
735:         xbee_mutex_unlock(xbee->conmutex);
736:         return con;
737:
738:         /* if: looking for a localAT, and the frameIDs match! */
739:       } else if ((type == xbee_localAT) &&
740:                  (con->type == type) &&
741:                  (frameID == con->frameID)) {
742:         xbee_mutex_unlock(xbee->conmutex);
743:         return con;
744:
745:         /* if: connection types match, the frameIDs match, and the addresses match! */
746:       } else if ((type == con->type) &&
747:                  (frameID == con->frameID) &&
748:                  (!memcmp(tAddr,con->tAddr,8))) {
749:         xbee_mutex_unlock(xbee->conmutex);
750:         return con;
751:       }
752:
753:       /* if there are more, move along, dont want to loose that last item! */
754:       if (con->next == NULL) break;
755:       con = con->next;
756:     }
757:
758:     /* keep hold of the last connection... we will need to link it up later */
759:     ocon = con;
760:   }
761:
762:   /* unlock the connection mutex */
763:   xbee_mutex_unlock(xbee->conmutex);
764:
765:   /* create a new connection and set its attributes */
```

```
766:   con = Xcalloc(sizeof(xbee_con));
767:   con->type = type;
768:   /* is it a 64bit connection? */
769:   if ((type == xbee_64bitRemoteAT) ||
770:       (type == xbee_64bitData) ||
771:       (type == xbee_64bitIO) ||
772:       (type == xbee2_data)) {
773:     con->tAddr64 = TRUE;
774:   }
775:   con->atQueue = 0; /* queue AT commands? */
776:   con->txDisableACK = 0; /* disable ACKs? */
777:   con->txBroadcastPAN = 0; /* broadcast? */
778:   con->frameID = frameID;
779:   con->waitforACK = 0;
780:   memcpy(con->tAddr,tAddr,8); /* copy in the remote address */
781:   xbee_mutex_init(con->callbackmutex);
782:   xbee_mutex_init(con->callbackListmutex);
783:   xbee_mutex_init(con->Txmutex);
784:   xbee_sem_init(con->waitforACKsem);
785:
786:   if (xbee->log) {
787:     switch(type) {
788:     case xbee_localAT:
789:       xbee_log("New local AT connection!");
790:       break;
791:     case xbee_16bitRemoteAT:
792:     case xbee_64bitRemoteAT:
793:       xbee_logc("New %d-bit remote AT connection! (to: ",(con->tAddr64?64:16));
794:       for (i=0;i<(con->tAddr64?8:2);i++) {
795:         fprintf(xbee->log,(i?":%02X":"%02X"),tAddr[i]);
796:       }
797:       fprintf(xbee->log,")");
798:       xbee_logcf();
799:       break;
800:     case xbee_16bitData:
801:     case xbee_64bitData:
802:       xbee_logc("New %d-bit data connection! (to: ",(con->tAddr64?64:16));
803:       for (i=0;i<(con->tAddr64?8:2);i++) {
804:         fprintf(xbee->log,(i?":%02X":"%02X"),tAddr[i]);
805:       }
806:       fprintf(xbee->log,")");
807:       xbee_logcf();
808:       break;
809:     case xbee_16bitIO:
810:     case xbee_64bitIO:
811:       xbee_logc("New %d-bit IO connection! (to: ",(con->tAddr64?64:16));
812:       for (i=0;i<(con->tAddr64?8:2);i++) {
813:         fprintf(xbee->log,(i?":%02X":"%02X"),tAddr[i]);
814:       }
815:       fprintf(xbee->log,")");
816:       xbee_logcf();
817:       break;
818:     case xbee2_data:
819:       xbee_logc("New Series 2 data connection! (to: ");
820:       for (i=0;i<8;i++) {
821:         fprintf(xbee->log,(i?":%02X":"%02X"),tAddr[i]);
822:       }
823:       fprintf(xbee->log,")");
824:       xbee_logcf();
825:       break;
826:     case xbee_txStatus:
827:       xbee_log("New Tx status connection!");
828:       break;
829:     case xbee_modemStatus:
830:       xbee_log("New modem status connection!");
831:       break;
832:     case xbee_unknown:
833:     default:
834:       xbee_log("New unknown connection!");
835:     }
836:   }
837:
838:   /* lock the connection mutex */
839:   xbee_mutex_lock(xbee->conmutex);
840:
841:   /* make it the last in the list */
842:   con->next = NULL;
843:   /* add it to the list */
844:   if (xbee->conlist) {
845:     ocon->next = con;
846:   } else {
847:     xbee->conlist = con;
848:   }
849:
850:   /* unlock the mutex */
```

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

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

```
1021:   unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
1022:
1023:   ISREADYR(-1);
1024:
1025:   if (!con) return -1;
1026:   if (con->type == xbee_unknown) return -1;
1027:   if (length > 127) return -1;
1028:
1029:   if (xbee->log) {
1030:     xbee_logS("--== TX Packet ============--");
1031:     xbee_logIc("Connection Type: ");
1032:     switch (con->type) {
1033:     case xbee_unknown:       fprintf(xbee->log,"Unknown"); break;
1034:     case xbee_localAT:       fprintf(xbee->log,"Local AT"); break;
1035:     case xbee_remoteAT:      fprintf(xbee->log,"Remote AT"); break;
1036:     case xbee_16bitRemoteAT: fprintf(xbee->log,"Remote AT (16-bit)"); break;
1037:     case xbee_64bitRemoteAT: fprintf(xbee->log,"Remote AT (64-bit)"); break;
1038:     case xbee_16bitData:     fprintf(xbee->log,"Data (16-bit)"); break;
1039:     case xbee_64bitData:     fprintf(xbee->log,"Data (64-bit)"); break;
1040:     case xbee_16bitIO:       fprintf(xbee->log,"IO (16-bit)"); break;
1041:     case xbee_64bitIO:       fprintf(xbee->log,"IO (64-bit)"); break;
1042:     case xbee2_data:         fprintf(xbee->log,"Series 2 Data"); break;
1043:     case xbee2_txStatus:     fprintf(xbee->log,"Series 2 Tx Status"); break;
1044:     case xbee_txStatus:      fprintf(xbee->log,"Tx Status"); break;
1045:     case xbee_modemStatus:   fprintf(xbee->log,"Modem Status"); break;
1046:     }
1047:     xbee_logIcf();
1048:     switch (con->type) {
1049:     case xbee_localAT: case xbee_remoteAT: case xbee_txStatus: case xbee_modemStatus:
1050:       break;
1051:     default:
1052:       xbee_logIc("Destination: ");
1053:       for (i=0;i<(con->tAddr64?8:2);i++) {
1054:         fprintf(xbee->log,(i?":%02X":"%02X"),con->tAddr[i]);
1055:       }
1056:       xbee_logIcf();
1057:     }
1058:     xbee_logI("Length: %d",length);
1059:     for (i=0;i<length;i++) {
1060:       xbee_logIc("%3d | 0x%02X ",i,(unsigned char)data[i]);
1061:       if ((data[i] > 32) && (data[i] < 127)) {
1062:         fprintf(xbee->log,"'%c'",data[i]);
1063:       } else{
1064:         fprintf(xbee->log," _");
1065:       }
1066:       xbee_logIcf();
1067:     }
1068:     xbee_logEf();
1069:   }
1070:
1071:   /* ########################################## */
1072:   /* if: local AT */
1073:   if (con->type == xbee_localAT) {
1074:     /* AT commands are 2 chars long (plus optional parameter) */
1075:     if (length < 2) return -1;
1076:     if (length > 32) return -1;
1077:
1078:     /* use the command? */
1079:     buf[0] = ((!con->atQueue)?XBEE_LOCAL_ATREQ:XBEE_LOCAL_ATQUE);
1080:     buf[1] = con->frameID;
1081:
1082:     /* copy in the data */
1083:     for (i=0;i<length;i++) {
1084:       buf[i+2] = data[i];
1085:     }
1086:
1087:     /* setup the packet */
1088:     pkt = xbee_make_pkt(xbee, buf, i+2);
1089:     /* send it on */
1090:     return _xbee_send_pkt(xbee, pkt, con);
1091:
1092:     /* ########################################## */
1093:     /* if: remote AT */
1094:   } else if ((con->type == xbee_16bitRemoteAT) ||
1095:              (con->type == xbee_64bitRemoteAT)) {
1096:     if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter) */
1097:     if (length > 32) return -1;
1098:     buf[0] = XBEE_REMOTE_ATREQ;
1099:     buf[1] = con->frameID;
1100:
1101:     /* copy in the relevant address */
1102:     if (con->tAddr64) {
1103:       memcpy(&buf[2],con->tAddr,8);
1104:       buf[10] = 0xFF;
1105:       buf[11] = 0xFE;
```

```
1106:     } else {
1107:       memset(&buf[2],0,8);
1108:       memcpy(&buf[10],con->tAddr,2);
1109:     }
1110:     /* queue the command? */
1111:     buf[12] = ((!con->atQueue)?0x02:0x00);
1112:
1113:     /* copy in the data */
1114:     for (i=0;i<length;i++) {
1115:       buf[i+13] = data[i];
1116:     }
1117:
1118:     /* setup the packet */
1119:     pkt = xbee_make_pkt(xbee, buf, i+13);
1120:     /* send it on */
1121:     return _xbee_send_pkt(xbee, pkt, con);
1122:
1123:     /* ########################################## */
1124:     /* if: 16 or 64bit Data */
1125:   } else if ((con->type == xbee_16bitData) ||
1126:              (con->type == xbee_64bitData)) {
1127:     int offset;
1128:     if (length > 100) return -1;
1129:
1130:     /* if: 16bit Data */
1131:     if (con->type == xbee_16bitData) {
1132:       buf[0] = XBEE_16BIT_DATATX;
1133:       offset = 5;
1134:       /* copy in the address */
1135:       memcpy(&buf[2],con->tAddr,2);
1136:
1137:       /* if: 64bit Data */
1138:     } else { /* 64bit Data */
1139:       buf[0] = XBEE_64BIT_DATATX;
1140:       offset = 11;
1141:       /* copy in the address */
1142:       memcpy(&buf[2],con->tAddr,8);
1143:     }
1144:
1145:     /* copy frameID */
1146:     buf[1] = con->frameID;
1147:
1148:     /* disable ack? broadcast? */
1149:     buf[offset-1] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcastPAN)?0x04:0x00);
1150:
1151:     /* copy in the data */
1152:     for (i=0;i<length;i++) {
1153:       buf[i+offset] = data[i];
1154:     }
1155:
1156:     /* setup the packet */
1157:     pkt = xbee_make_pkt(xbee, buf, i+offset);
1158:     /* send it on */
1159:     return _xbee_send_pkt(xbee, pkt, con);
1160:
1161:     /* ########################################## */
1162:     /* if: I/O */
1163:   } else if ((con->type == xbee_64bitIO) ||
1164:              (con->type == xbee_16bitIO)) {
1165:     /* not currently implemented... is it even allowed? */
1166:     if (xbee->log) {
1167:       xbee_log("******* TODO ********\n");
1168:     }
1169:
1170:     /* ########################################## */
1171:     /* if: Series 2 Data */
1172:   } else if (con->type == xbee2_data) {
1173:     if (length > 72) return -1;
1174:
1175:     buf[0] = XBEE2_DATATX;
1176:     buf[1] = con->frameID;
1177:
1178:     /* copy in the relevant address */
1179:     memcpy(&buf[2],con->tAddr,8);
1180:     buf[10] = 0xFF;
1181:     buf[11] = 0xFE;
1182:
1183:     /* Maximum Radius/hops */
1184:     buf[12] = 0x00;
1185:
1186:     /* Options */
1187:     buf[13] = 0x00;
1188:
1189:     /* copy in the data */
1190:     for (i=0;i<length;i++) {
```

```
1191:       buf[i+14] = data[i];
1192:     }
1193:
1194:     /* setup the packet */
1195:     pkt = xbee_make_pkt(xbee, buf, i+14);
1196:     /* send it on */
1197:     return _xbee_send_pkt(xbee, pkt, con);
1198:   }
1199:
1200:   return -2;
1201: }
1202:
1203: /* ###################################################################
1204:    xbee_getpacket
1205:    retrieves the next packet destined for the given connection
1206:    once the packet has been retrieved, it is removed for the list! */
1207: xbee_pkt *xbee_getpacketwait(xbee_con *con) {
1208:   return _xbee_getpacketwait(default_xbee, con);
1209: }
1210: xbee_pkt *_xbee_getpacketwait(xbee_hnd xbee, xbee_con *con) {
1211:   xbee_pkt *p = NULL;
1212:   int i = 20;
1213:
1214:   /* 50ms * 20 = 1 second */
1215:   for (; i; i--) {
1216:     p = _xbee_getpacket(xbee, con);
1217:     if (p) break;
1218:     usleep(50000); /* 50ms */
1219:   }
1220:
1221:   return p;
1222: }
1223: xbee_pkt *xbee_getpacket(xbee_con *con) {
1224:   return _xbee_getpacket(default_xbee, con);
1225: }
1226: xbee_pkt *_xbee_getpacket(xbee_hnd xbee, xbee_con *con) {
1227:   xbee_pkt *l, *p, *q;
1228:
1229:   ISREADYR(NULL);
1230:
1231:   /* lock the packet mutex */
1232:   xbee_mutex_lock(xbee->pktmutex);
1233:
1234:   /* if: there are no packets */
1235:   if ((p = xbee->pktlist) == NULL) {
1236:     xbee_mutex_unlock(xbee->pktmutex);
1237:     /*if (xbee->log) {
1238:       xbee_log("No packets avaliable...");
1239:       }*/
1240:     return NULL;
1241:   }
1242:
1243:   l = NULL;
1244:   q = NULL;
1245:   /* get the first avaliable packet for this connection */
1246:   do {
1247:     /* does the packet match the connection? */
1248:     if (xbee_matchpktcon(xbee, p, con)) {
1249:       q = p;
1250:       break;
1251:     }
1252:     /* move on */
1253:     l = p;
1254:     p = p->next;
1255:   } while (p);
1256:
1257:   /* if: no packet was found */
1258:   if (!q) {
1259:     xbee_mutex_unlock(xbee->pktmutex);
1260:     if (xbee->log) {
1261:       struct timeval tv;
1262:       xbee_logS("--== Get Packet ==========--");
1263:       gettimeofday(&tv,NULL);
1264:       xbee_logE("Didn't get a packet @ %ld.%06ld",tv.tv_sec,tv.tv_usec);
1265:     }
1266:     return NULL;
1267:   }
1268:
1269:   /* if it was the first packet */
1270:   if (l) {
1271:     /* relink the list */
1272:     l->next = p->next;
1273:     if (!l->next) xbee->pktlast = l;
1274:   } else {
1275:     /* move the chain along */
```

```
1276:     xbee->pktlist = p->next;
1277:     if (!xbee->pktlist) {
1278:       xbee->pktlast = NULL;
1279:     } else if (!xbee->pktlist->next) {
1280:       xbee->pktlast = xbee->pktlist;
1281:     }
1282:   }
1283:   xbee->pktcount--;
1284:
1285:   /* unlink this packet from the chain! */
1286:   q->next = NULL;
1287:
1288:   if (xbee->log) {
1289:     struct timeval tv;
1290:     xbee_logS("--== Get Packet ==========--");
1291:     gettimeofday(&tv,NULL);
1292:     xbee_logI("Got a packet @ %ld.%06ld",tv.tv_sec,tv.tv_usec);
1293:     xbee_logE("Packets left: %d",xbee->pktcount);
1294:   }
1295:
1296:   /* unlock the packet mutex */
1297:   xbee_mutex_unlock(xbee->pktmutex);
1298:
1299:   /* and return the packet (must be free'd by caller!) */
1300:   return q;
1301: }
1302:
1303: /* ########################################################################
1304:    xbee_matchpktcon - INTERNAL
1305:    checks if the packet matches the connection */
1306: static int xbee_matchpktcon(xbee_hnd xbee, xbee_pkt *pkt, xbee_con *con) {
1307:   /* if: the connection type matches the packet type OR
1308:      the connection is 16/64bit remote AT, and the packet is a remote AT response */
1309:   if ((pkt->type == con->type) || /* -- */
1310:       ((pkt->type == xbee_remoteAT) && /* -- */
1311:        ((con->type == xbee_16bitRemoteAT) ||
1312:         (con->type == xbee_64bitRemoteAT)))) {
1313:
1314:
1315:     /* if: is a modem status (there can only be 1 modem status connection) */
1316:     if (pkt->type == xbee_modemStatus) return 1;
1317:
1318:     /* if: the packet is a txStatus or localAT and the frameIDs match */
1319:     if ((pkt->type == xbee_txStatus) ||
1320:         (pkt->type == xbee_localAT)) {
1321:       if (pkt->frameID == con->frameID) {
1322:         return 1;
1323:       }
1324:     /* if: the packet was sent as a 16bit remoteAT, and the 16bit addresss match */
1325:     } else if ((pkt->type == xbee_remoteAT) &&
1326:                (con->type == xbee_16bitRemoteAT) &&
1327:                !memcmp(pkt->Addr16,con->tAddr,2)) {
1328:       return 1;
1329:     /* if: the packet was sent as a 64bit remoteAT, and the 64bit addresss match */
1330:     } else if ((pkt->type == xbee_remoteAT) &&
1331:                (con->type == xbee_64bitRemoteAT) &&
1332:                !memcmp(pkt->Addr64,con->tAddr,8)) {
1333:       return 1;
1334:     /* if: the packet is 64bit addressed, and the addresses match */
1335:     } else if (pkt->sAddr64 && !memcmp(pkt->Addr64,con->tAddr,8)) {
1336:       return 1;
1337:     /* if: the packet is 16bit addressed, and the addresses match */
1338:     } else if (!pkt->sAddr64 && !memcmp(pkt->Addr16,con->tAddr,2)) {
1339:       return 1;
1340:     } else if (con->type == pkt->type &&
1341:                (con->type == xbee_16bitData || con->type == xbee_64bitData) &&
1342:                (pkt->isBroadcastADR || pkt->isBroadcastPAN)) {
1343:       unsigned char t[8] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
1344:       if ((con->tAddr64 && !memcmp(con->tAddr,t,8)) ||
1345:           (!con->tAddr64 && !memcmp(con->tAddr,t,2))) {
1346:         return 1;
1347:       }
1348:     }
1349:   }
1350:   return 0;
1351: }
1352:
1353: /* ########################################################################
1354:    xbee_parse_io - INTERNAL
1355:    parses the data given into the packet io information */
1356: static int xbee_parse_io(xbee_hnd xbee, xbee_pkt *p, unsigned char *d,
1357:                          int maskOffset, int sampleOffset, int sample) {
1358:   xbee_sample *s = &(p->IOdata[sample]);
1359:
1360:   /* copy in the I/O data mask */
```

```
1361:   s->IOmask = (((d[maskOffset]<<8) | d[maskOffset + 1]) & 0x7FFF);
1362:
1363:   /* copy in the digital I/O data */
1364:   s->IOdigital = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x01FF);
1365:
1366:   /* advance over the digital data, if its there */
1367:   sampleOffset += ((s->IOmask & 0x01FF)?2:0);
1368:
1369:   /* copy in the analog I/O data */
1370:   if (s->IOmask & 0x0200) {
1371:     s->IOanalog[0] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1372:     sampleOffset+=2;
1373:   }
1374:   if (s->IOmask & 0x0400) {
1375:     s->IOanalog[1] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1376:     sampleOffset+=2;
1377:   }
1378:   if (s->IOmask & 0x0800) {
1379:     s->IOanalog[2] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1380:     sampleOffset+=2;
1381:   }
1382:   if (s->IOmask & 0x1000) {
1383:     s->IOanalog[3] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1384:     sampleOffset+=2;
1385:   }
1386:   if (s->IOmask & 0x2000) {
1387:     s->IOanalog[4] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1388:     sampleOffset+=2;
1389:   }
1390:   if (s->IOmask & 0x4000) {
1391:     s->IOanalog[5] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1392:     sampleOffset+=2;
1393:   }
1394:
1395:   if (xbee->log) {
1396:     if (s->IOmask & 0x0001)
1397:       xbee_logI("Digital 0: %c",((s->IOdigital & 0x0001)?'1':'0'));
1398:     if (s->IOmask & 0x0002)
1399:       xbee_logI("Digital 1: %c",((s->IOdigital & 0x0002)?'1':'0'));
1400:     if (s->IOmask & 0x0004)
1401:       xbee_logI("Digital 2: %c",((s->IOdigital & 0x0004)?'1':'0'));
1402:     if (s->IOmask & 0x0008)
1403:       xbee_logI("Digital 3: %c",((s->IOdigital & 0x0008)?'1':'0'));
1404:     if (s->IOmask & 0x0010)
1405:       xbee_logI("Digital 4: %c",((s->IOdigital & 0x0010)?'1':'0'));
1406:     if (s->IOmask & 0x0020)
1407:       xbee_logI("Digital 5: %c",((s->IOdigital & 0x0020)?'1':'0'));
1408:     if (s->IOmask & 0x0040)
1409:       xbee_logI("Digital 6: %c",((s->IOdigital & 0x0040)?'1':'0'));
1410:     if (s->IOmask & 0x0080)
1411:       xbee_logI("Digital 7: %c",((s->IOdigital & 0x0080)?'1':'0'));
1412:     if (s->IOmask & 0x0100)
1413:       xbee_logI("Digital 8: %c",((s->IOdigital & 0x0100)?'1':'0'));
1414:     if (s->IOmask & 0x0200)
1415:       xbee_logI("Analog  0: %d (~%.2fv)",s->IOanalog[0],(3.3/1023)*s->IOanalog[0]);
1416:     if (s->IOmask & 0x0400)
1417:       xbee_logI("Analog  1: %d (~%.2fv)",s->IOanalog[1],(3.3/1023)*s->IOanalog[1]);
1418:     if (s->IOmask & 0x0800)
1419:       xbee_logI("Analog  2: %d (~%.2fv)",s->IOanalog[2],(3.3/1023)*s->IOanalog[2]);
1420:     if (s->IOmask & 0x1000)
1421:       xbee_logI("Analog  3: %d (~%.2fv)",s->IOanalog[3],(3.3/1023)*s->IOanalog[3]);
1422:     if (s->IOmask & 0x2000)
1423:       xbee_logI("Analog  4: %d (~%.2fv)",s->IOanalog[4],(3.3/1023)*s->IOanalog[4]);
1424:     if (s->IOmask & 0x4000)
1425:       xbee_logI("Analog  5: %d (~%.2fv)",s->IOanalog[5],(3.3/1023)*s->IOanalog[5]);
1426:   }
1427:
1428:   return sampleOffset;
1429: }
1430:
1431: /* ########################################################################
1432:    xbee_listen_stop
1433:    stops the listen thread after the current packet has been processed */
1434: void xbee_listen_stop(xbee_hnd xbee) {
1435:   ISREADYP();
1436:   xbee->run = 0;
1437: }
1438:
1439: /* ########################################################################
1440:    xbee_listen_wrapper - INTERNAL
1441:    the xbee_listen wrapper. Prints an error when xbee_listen ends */
1442: static void xbee_listen_wrapper(xbee_hnd xbee) {
1443:   int ret;
1444:
1445:   /* just falls out if the proper 'go-ahead' isn't given */
```

```
1446:   if (xbee->xbee_ready != -1) return;
1447:   /* now allow the parent to continue */
1448:   xbee->xbee_ready = -2;
1449:
1450: #ifdef _WIN32 /* ---- */
1451:   /* win32 requires this delay... no idea why */
1452:   usleep(1000000);
1453: #endif /* ---------- */
1454:
1455:   while (xbee->run) {
1456:     ret = xbee_listen(xbee);
1457:     if (!xbee->run) break;
1458:     xbee_log("xbee_listen() returned [%d]... Restarting in 25ms!",ret);
1459:     usleep(25000);
1460:   }
1461: }
1462:
1463: /* xbee_listen - INTERNAL
1464:    the xbee xbee_listen thread
1465:    reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
1466: static int xbee_listen(xbee_hnd xbee) {
1467: #define LISTEN_BUFLEN 1024
1468:   unsigned char c, t, d[LISTEN_BUFLEN];
1469:   unsigned int l, i, chksum, o;
1470:   int j;
1471:   xbee_pkt *p = NULL, *q;
1472:   xbee_con *con;
1473:   int hasCon;
1474:
1475:   /* do this forever :) */
1476:   while (xbee->run) {
1477:     /* clean up any undesired storage */
1478:     if (p) Xfree(p);
1479:
1480:     /* wait for a valid start byte */
1481:     if ((c = xbee_getrawbyte(xbee)) != 0x7E) {
1482:       if (xbee->log) xbee_log("****** Unexpected byte (0x%02X)... ******",c);
1483:       continue;
1484:     }
1485:     if (!xbee->run) return 0;
1486:
1487:     xbee_logSf();
1488:     if (xbee->log) {
1489:       struct timeval tv;
1490:       xbee_logI("--== RX Packet ============--");
1491:       gettimeofday(&tv,NULL);
1492:       xbee_logI("Got a packet @ %ld.%06ld",tv.tv_sec,tv.tv_usec);
1493:     }
1494:
1495:     /* get the length */
1496:     l = xbee_getbyte(xbee) << 8;
1497:     l += xbee_getbyte(xbee);
1498:
1499:     /* check it is a valid length... */
1500:     if (!l) {
1501:       if (xbee->log) {
1502:         xbee_logI("Recived zero length packet!");
1503:       }
1504:       continue;
1505:     }
1506:     if (l > 100) {
1507:       if (xbee->log) {
1508:         xbee_logI("Recived oversized packet! Length: %d",l - 1);
1509:       }
1510:     }
1511:     if (l > LISTEN_BUFLEN) {
1512:       if (xbee->log) {
1513:         xbee_logI("Recived packet larger than buffer! Discarding...");
1514:       }
1515:       continue;
1516:     }
1517:
1518:     if (xbee->log) {
1519:       xbee_logI("Length: %d",l - 1);
1520:     }
1521:
1522:     /* get the packet type */
1523:     t = xbee_getbyte(xbee);
1524:
1525:     /* start the checksum */
1526:     chksum = t;
1527:
1528:     /* suck in all the data */
1529:     for (i = 0; l > 1 && i < LISTEN_BUFLEN; l--, i++) {
1530:       /* get an unescaped byte */
```

```
1531:       c = xbee_getbyte(xbee);
1532:       d[i] = c;
1533:       chksum += c;
1534:       if (xbee->log) {
1535:         xbee_logIc("%3d | 0x%02X | ",i,c);
1536:         if ((c > 32) && (c < 127)) fprintf(xbee->log,"'%c'",c); else fprintf(xbee->log," _ ");
1537:
1538:         if ((t == XBEE_LOCAL_AT     && i == 4) ||
1539:             (t == XBEE_REMOTE_AT    && i == 14) ||
1540:             (t == XBEE_64BIT_DATARX && i == 10) ||
1541:             (t == XBEE_16BIT_DATARX && i == 4) ||
1542:             (t == XBEE_64BIT_IO     && i == 13) ||
1543:             (t == XBEE_16BIT_IO     && i == 7)) {
1544:           /* mark the beginning of the 'data' bytes */
1545:           fprintf(xbee->log,"   <-- data starts");
1546:         } else if (t == XBEE_64BIT_IO) {
1547:           if (i == 10)      fprintf(xbee->log,"   <-- sample count");
1548:           else if (i == 11) fprintf(xbee->log,"   <-- mask (msb)");
1549:           else if (i == 12) fprintf(xbee->log,"   <-- mask (lsb)");
1550:         } else if (t == XBEE_16BIT_IO) {
1551:           if (i == 4)       fprintf(xbee->log,"   <-- sample count");
1552:           else if (i == 5)  fprintf(xbee->log,"   <-- mask (msb)");
1553:           else if (i == 6)  fprintf(xbee->log,"   <-- mask (lsb)");
1554:         }
1555:         xbee_logIcf();
1556:       }
1557:     }
1558:     i--; /* it went up too many times!... */
1559:
1560:     /* add the checksum */
1561:     chksum += xbee_getbyte(xbee);
1562:
1563:     /* check if the whole packet was recieved, or something else occured... unlikely... */
1564:     if (l>1) {
1565:       if (xbee->log) {
1566:         xbee_logE("Didn't get whole packet... :(");
1567:       }
1568:       continue;
1569:     }
1570:
1571:     /* check the checksum */
1572:     if ((chksum & 0xFF) != 0xFF) {
1573:       if (xbee->log) {
1574:         chksum &= 0xFF;
1575:         xbee_logE("Invalid Checksum: 0x%02X",chksum);
1576:       }
1577:       continue;
1578:     }
1579:
1580:     /* make a new packet */
1581:     p = Xcalloc(sizeof(xbee_pkt));
1582:     q = NULL;
1583:     p->datalen = 0;
1584:
1585:     /* ######################################### */
1586:     /* if: modem status */
1587:     if (t == XBEE_MODEM_STATUS) {
1588:       if (xbee->log) {
1589:         xbee_logI("Packet type: Modem Status (0x8A)");
1590:         xbee_logIc("Event: ");
1591:         switch (d[0]) {
1592:         case 0x00: fprintf(xbee->log,"Hardware reset"); break;
1593:         case 0x01: fprintf(xbee->log,"Watchdog timer reset"); break;
1594:         case 0x02: fprintf(xbee->log,"Associated"); break;
1595:         case 0x03: fprintf(xbee->log,"Disassociated"); break;
1596:         case 0x04: fprintf(xbee->log,"Synchronization lost"); break;
1597:         case 0x05: fprintf(xbee->log,"Coordinator realignment"); break;
1598:         case 0x06: fprintf(xbee->log,"Coordinator started"); break;
1599:         }
1600:         fprintf(xbee->log,"... (0x%02X)",d[0]);
1601:         xbee_logIcf();
1602:       }
1603:       p->type = xbee_modemStatus;
1604:
1605:       p->sAddr64 = FALSE;
1606:       p->dataPkt = FALSE;
1607:       p->txStatusPkt = FALSE;
1608:       p->modemStatusPkt = TRUE;
1609:       p->remoteATPkt = FALSE;
1610:       p->IOPkt = FALSE;
1611:
1612:       /* modem status can only ever give 1 'data' byte */
1613:       p->datalen = 1;
1614:       p->data[0] = d[0];
1615:
```

```
1616:       /* ######################################### */
1617:       /* if: local AT response */
1618:     } else if (t == XBEE_LOCAL_AT) {
1619:       if (xbee->log) {
1620:         xbee_logI("Packet type: Local AT Response (0x88)");
1621:         xbee_logI("FrameID: 0x%02X",d[0]);
1622:         xbee_logI("AT Command: %c%c",d[1],d[2]);
1623:         xbee_logIc("Status: ");
1624:         if      (d[3] == 0x00) fprintf(xbee->log,"OK");
1625:         else if (d[3] == 0x01) fprintf(xbee->log,"Error");
1626:         else if (d[3] == 0x02) fprintf(xbee->log,"Invalid Command");
1627:         else if (d[3] == 0x03) fprintf(xbee->log,"Invalid Parameter");
1628:         fprintf(xbee->log," (0x%02X)",d[3]);
1629:         xbee_logIcf();
1630:       }
1631:       p->type = xbee_localAT;
1632:
1633:       p->sAddr64 = FALSE;
1634:       p->dataPkt = FALSE;
1635:       p->txStatusPkt = FALSE;
1636:       p->modemStatusPkt = FALSE;
1637:       p->remoteATPkt = FALSE;
1638:       p->IOPkt = FALSE;
1639:
1640:       p->frameID = d[0];
1641:       p->atCmd[0] = d[1];
1642:       p->atCmd[1] = d[2];
1643:
1644:       p->status = d[3];
1645:
1646:       /* copy in the data */
1647:       p->datalen = i-3;
1648:       for (;i>3;i--) p->data[i-4] = d[i];
1649:
1650:       /* ######################################### */
1651:       /* if: remote AT response */
1652:     } else if (t == XBEE_REMOTE_AT) {
1653:       if (xbee->log) {
1654:         xbee_logI("Packet type: Remote AT Response (0x97)");
1655:         xbee_logI("FrameID: 0x%02X",d[0]);
1656:         xbee_logIc("64-bit Address: ");
1657:         for (j=0;j<8;j++) {
1658:           fprintf(xbee->log,(j?":%02X":"%02X"),d[1+j]);
1659:         }
1660:         xbee_logIcf();
1661:         xbee_logIc("16-bit Address: ");
1662:         for (j=0;j<2;j++) {
1663:           fprintf(xbee->log,(j?":%02X":"%02X"),d[9+j]);
1664:         }
1665:         xbee_logIcf();
1666:         xbee_logI("AT Command: %c%c",d[11],d[12]);
1667:         xbee_logIc("Status: ");
1668:         if      (d[13] == 0x00) fprintf(xbee->log,"OK");
1669:         else if (d[13] == 0x01) fprintf(xbee->log,"Error");
1670:         else if (d[13] == 0x02) fprintf(xbee->log,"Invalid Command");
1671:         else if (d[13] == 0x03) fprintf(xbee->log,"Invalid Parameter");
1672:         else if (d[13] == 0x04) fprintf(xbee->log,"No Response");
1673:         fprintf(xbee->log," (0x%02X)",d[13]);
1674:         xbee_logIcf();
1675:       }
1676:       p->type = xbee_remoteAT;
1677:
1678:       p->sAddr64 = FALSE;
1679:       p->dataPkt = FALSE;
1680:       p->txStatusPkt = FALSE;
1681:       p->modemStatusPkt = FALSE;
1682:       p->remoteATPkt = TRUE;
1683:       p->IOPkt = FALSE;
1684:
1685:       p->frameID = d[0];
1686:
1687:       p->Addr64[0] = d[1];
1688:       p->Addr64[1] = d[2];
1689:       p->Addr64[2] = d[3];
1690:       p->Addr64[3] = d[4];
1691:       p->Addr64[4] = d[5];
1692:       p->Addr64[5] = d[6];
1693:       p->Addr64[6] = d[7];
1694:       p->Addr64[7] = d[8];
1695:
1696:       p->Addr16[0] = d[9];
1697:       p->Addr16[1] = d[10];
1698:
1699:       p->atCmd[0] = d[11];
1700:       p->atCmd[1] = d[12];
```

```
1701:
1702:       p->status = d[13];
1703:
1704:       p->samples = 1;
1705:
1706:       if (p->status == 0x00 && p->atCmd[0] == 'I' && p->atCmd[1] == 'S') {
1707:         /* parse the io data */
1708:         xbee_logI("--- Sample -----------------");
1709:         xbee_parse_io(xbee, p, d, 15, 17, 0);
1710:         xbee_logI("----------------------------");
1711:       } else {
1712:         /* copy in the data */
1713:         p->datalen = i-13;
1714:         for (;i>13;i--) p->data[i-14] = d[i];
1715:       }
1716:
1717:       /* ######################################### */
1718:       /* if: TX status */
1719:     } else if (t == XBEE_TX_STATUS) {
1720:       if (xbee->log) {
1721:         xbee_logI("Packet type: TX Status Report (0x89)");
1722:         xbee_logI("FrameID: 0x%02X",d[0]);
1723:         xbee_logIc("Status: ");
1724:         if      (d[1] == 0x00) fprintf(xbee->log,"Success");
1725:         else if (d[1] == 0x01) fprintf(xbee->log,"No ACK");
1726:         else if (d[1] == 0x02) fprintf(xbee->log,"CCA Failure");
1727:         else if (d[1] == 0x03) fprintf(xbee->log,"Purged");
1728:         fprintf(xbee->log," (0x%02X)",d[1]);
1729:         xbee_logIcf();
1730:       }
1731:       p->type = xbee_txStatus;
1732:
1733:       p->sAddr64 = FALSE;
1734:       p->dataPkt = FALSE;
1735:       p->txStatusPkt = TRUE;
1736:       p->modemStatusPkt = FALSE;
1737:       p->remoteATPkt = FALSE;
1738:       p->IOPkt = FALSE;
1739:
1740:       p->frameID = d[0];
1741:
1742:       p->status = d[1];
1743:
1744:       /* never returns data */
1745:       p->datalen = 0;
1746:
1747:       /* check for any connections waiting for a status update */
1748:       /* lock the connection mutex */
1749:       xbee_mutex_lock(xbee->conmutex);
1750:       xbee_logI("Looking for a connection that wants a status update...");
1751:       con = xbee->conlist;
1752:       while (con) {
1753:         if ((con->frameID == p->frameID) &&
1754:             (con->ACKstatus == 0xFF)) {
1755:           xbee_logI("Found @ 0x%08X!",con);
1756:           con->ACKstatus = p->status;
1757:           xbee_sem_post(con->waitforACKsem);
1758:         }
1759:         con = con->next;
1760:       }
1761:
1762:       /* unlock the connection mutex */
1763:       xbee_mutex_unlock(xbee->conmutex);
1764:
1765:       /* ######################################### */
1766:       /* if: 16 / 64bit data recieve */
1767:     } else if ((t == XBEE_64BIT_DATARX) ||
1768:                (t == XBEE_16BIT_DATARX)) {
1769:       int offset;
1770:       if (t == XBEE_64BIT_DATARX) { /* 64bit */
1771:         offset = 8;
1772:       } else { /* 16bit */
1773:         offset = 2;
1774:       }
1775:       if (xbee->log) {
1776:         xbee_logI("Packet type: %d-bit RX Data (0x%02X)",((t == XBEE_64BIT_DATARX)?64:16),t);
1777:         xbee_logIc("%d-bit Address: ",((t == XBEE_64BIT_DATARX)?64:16));
1778:         for (j=0;j<offset;j++) {
1779:           fprintf(xbee->log,(j?":%02X":"%02X"),d[j]);
1780:         }
1781:         xbee_logIcf();
1782:         xbee_logI("RSSI: -%ddB",d[offset]);
1783:         if (d[offset + 1] & 0x02) xbee_logI("Options: Address Broadcast");
1784:         if (d[offset + 1] & 0x04) xbee_logI("Options: PAN Broadcast");
1785:       }
```

```
1786:       p->isBroadcastADR = !!(d[offset+1] & 0x02);
1787:       p->isBroadcastPAN = !!(d[offset+1] & 0x04);
1788:       p->dataPkt = TRUE;
1789:       p->txStatusPkt = FALSE;
1790:       p->modemStatusPkt = FALSE;
1791:       p->remoteATPkt = FALSE;
1792:       p->IOPkt = FALSE;
1793:
1794:       if (t == XBEE_64BIT_DATARX) { /* 64bit */
1795:         p->type = xbee_64bitData;
1796:
1797:         p->sAddr64 = TRUE;
1798:
1799:         p->Addr64[0] = d[0];
1800:         p->Addr64[1] = d[1];
1801:         p->Addr64[2] = d[2];
1802:         p->Addr64[3] = d[3];
1803:         p->Addr64[4] = d[4];
1804:         p->Addr64[5] = d[5];
1805:         p->Addr64[6] = d[6];
1806:         p->Addr64[7] = d[7];
1807:       } else { /* 16bit */
1808:         p->type = xbee_16bitData;
1809:
1810:         p->sAddr64 = FALSE;
1811:
1812:         p->Addr16[0] = d[0];
1813:         p->Addr16[1] = d[1];
1814:       }
1815:
1816:       /* save the RSSI / signal strength
1817:          this can be used with printf as:
1818:          printf("-%ddB\n",p->RSSI); */
1819:       p->RSSI = d[offset];
1820:
1821:       p->status = d[offset + 1];
1822:
1823:       /* copy in the data */
1824:       p->datalen = i-(offset + 1);
1825:       for (;i>offset + 1;i--) p->data[i-(offset + 2)] = d[i];
1826:
1827:       /* ######################################### */
1828:       /* if: 16 / 64bit I/O recieve */
1829:     } else if ((t == XBEE_64BIT_IO) ||
1830:                (t == XBEE_16BIT_IO)) {
1831:       int offset,i2;
1832:       if (t == XBEE_64BIT_IO) { /* 64bit */
1833:         p->type = xbee_64bitIO;
1834:
1835:         p->sAddr64 = TRUE;
1836:
1837:         p->Addr64[0] = d[0];
1838:         p->Addr64[1] = d[1];
1839:         p->Addr64[2] = d[2];
1840:         p->Addr64[3] = d[3];
1841:         p->Addr64[4] = d[4];
1842:         p->Addr64[5] = d[5];
1843:         p->Addr64[6] = d[6];
1844:         p->Addr64[7] = d[7];
1845:
1846:         offset = 8;
1847:         p->samples = d[10];
1848:       } else { /* 16bit */
1849:         p->type = xbee_16bitIO;
1850:
1851:         p->sAddr64 = FALSE;
1852:
1853:         p->Addr16[0] = d[0];
1854:         p->Addr16[1] = d[1];
1855:
1856:         offset = 2;
1857:         p->samples = d[4];
1858:       }
1859:       if (p->samples > 1) {
1860:         p = Xrealloc(p, sizeof(xbee_pkt) + (sizeof(xbee_sample) * (p->samples - 1)));
1861:       }
1862:       if (xbee->log) {
1863:         xbee_logI("Packet type: %d-bit RX I/O Data (0x%02X)",((t == XBEE_64BIT_IO)?64:16),t);
1864:         xbee_logIc("%d-bit Address: ",((t == XBEE_64BIT_IO)?64:16));
1865:         for (j = 0; j < offset; j++) {
1866:           fprintf(xbee->log,(j?":%02X":"%02X"),d[j]);
1867:         }
1868:         xbee_logIcf();
1869:         xbee_logI("RSSI: -%ddB",d[offset]);
1870:         xbee_logI("Samples: %d",d[offset + 2]);
```

```
1871:       }
1872:       i2 = offset + 5;
1873:
1874:       /* never returns data */
1875:       p->datalen = 0;
1876:
1877:       p->dataPkt = FALSE;
1878:       p->txStatusPkt = FALSE;
1879:       p->modemStatusPkt = FALSE;
1880:       p->remoteATPkt = FALSE;
1881:       p->IOPkt = TRUE;
1882:
1883:       /* save the RSSI / signal strength
1884:          this can be used with printf as:
1885:          printf("-%ddB\n",p->RSSI); */
1886:       p->RSSI = d[offset];
1887:
1888:       p->status = d[offset + 1];
1889:
1890:       /* each sample is split into its own packet here, for simplicity */
1891:       for (o = 0; o < p->samples; o++) {
1892:         if (i2 >= i) {
1893:           xbee_logI("Invalid I/O data! Actually contained %d samples...",o);
1894:           p = Xrealloc(p, sizeof(xbee_pkt) + (sizeof(xbee_sample) * ((o>1)?o:1)));
1895:           p->samples = o;
1896:           break;
1897:         }
1898:         xbee_logI("--- Sample %3d -------------", o);
1899:
1900:         /* parse the io data */
1901:         i2 = xbee_parse_io(xbee, p, d, offset + 3, i2, o);
1902:       }
1903:       xbee_logI("----------------------------");
1904:
1905:       /* ######################################### */
1906:       /* if: Series 2 Transmit status */
1907:     } else if (t == XBEE2_TX_STATUS) {
1908:       if (xbee->log) {
1909:         xbee_logI("Packet type: Series 2 Transmit Status (0x%02X)", t);
1910:         xbee_logI("FrameID: 0x%02X",d[0]);
1911:         xbee_logI("16-bit Delivery Address: %02X:%02X",d[1],d[2]);
1912:         xbee_logI("Transmit Retry Count: %02X",d[3]);
1913:         xbee_logIc("Delivery Status: ");
1914:         if      (d[4] == 0x00) fprintf(xbee->log,"Success");
1915:         else if (d[4] == 0x02) fprintf(xbee->log,"CCA Failure");
1916:         else if (d[4] == 0x15) fprintf(xbee->log,"Invalid Destination");
1917:         else if (d[4] == 0x21) fprintf(xbee->log,"Network ACK Failure");
1918:         else if (d[4] == 0x22) fprintf(xbee->log,"Not Joined to Network");
1919:         else if (d[4] == 0x23) fprintf(xbee->log,"Self-Addressed");
1920:         else if (d[4] == 0x24) fprintf(xbee->log,"Address Not Found");
1921:         else if (d[4] == 0x25) fprintf(xbee->log,"Route Not Found");
1922:         else if (d[4] == 0x74) fprintf(xbee->log,"Data Payload Too Large"); /* ??? */
1923:         fprintf(xbee->log," (0x%02X)",d[4]);
1924:         xbee_logIcf();
1925:
1926:         xbee_logIc("Discovery Status: ");
1927:         if      (d[5] == 0x00) fprintf(xbee->log,"No Discovery Overhead");
1928:         else if (d[5] == 0x01) fprintf(xbee->log,"Address Discovery");
1929:         else if (d[5] == 0x02) fprintf(xbee->log,"Route Discovery");
1930:         else if (d[5] == 0x03) fprintf(xbee->log,"Address & Route Discovery");
1931:         fprintf(xbee->log," (0x%02X)",d[5]);
1932:         xbee_logIcf();
1933:       }
1934:
1935:       p->type = xbee2_txStatus;
1936:
1937:       p->sAddr64 = FALSE;
1938:       p->dataPkt = FALSE;
1939:       p->txStatusPkt = TRUE;
1940:       p->modemStatusPkt = FALSE;
1941:       p->remoteATPkt = FALSE;
1942:       p->IOPkt = FALSE;
1943:
1944:       p->frameID = d[0];
1945:
1946:       p->status = d[4];
1947:
1948:       /* never returns data */
1949:       p->datalen = 0;
1950:
1951:       /* ######################################### */
1952:       /* if: Series 2 data recieve */
1953:     } else if (t == XBEE2_DATARX) {
1954:       int offset;
1955:       offset = 10;
```

```
1956:       if (xbee->log) {
1957:         xbee_logI("Packet type: Series 2 Data Rx (0x%02X)", t);
1958:
1959:         xbee_logIc("64-bit Address: ");
1960:         for (j=0;j<8;j++) {
1961:           fprintf(xbee->log,(j?":%02X":"%02X"),d[j]);
1962:         }
1963:         xbee_logIcf();
1964:
1965:         xbee_logIc("16-bit Address: ");
1966:         for (j=0;j<2;j++) {
1967:           fprintf(xbee->log,(j?":%02X":"%02X"),d[j+8]);
1968:         }
1969:         xbee_logIcf();
1970:
1971:         if (d[offset] & 0x01) xbee_logI("Options: Packet Acknowledged");
1972:         if (d[offset] & 0x02) xbee_logI("Options: Packet was a broadcast packet");
1973:         if (d[offset] & 0x20) xbee_logI("Options: Packet Encrypted");                /* ??? */
1974:         if (d[offset] & 0x40) xbee_logI("Options: Packet from end device");          /* ??? */
1975:       }
1976:       p->dataPkt = TRUE;
1977:       p->txStatusPkt = FALSE;
1978:       p->modemStatusPkt = FALSE;
1979:       p->remoteATPkt = FALSE;
1980:       p->IOPkt = FALSE;
1981:       p->type = xbee2_data;
1982:       p->sAddr64 = TRUE;
1983:
1984:       p->Addr64[0] = d[0];
1985:       p->Addr64[1] = d[1];
1986:       p->Addr64[2] = d[2];
1987:       p->Addr64[3] = d[3];
1988:       p->Addr64[4] = d[4];
1989:       p->Addr64[5] = d[5];
1990:       p->Addr64[6] = d[6];
1991:       p->Addr64[7] = d[7];
1992:
1993:       p->Addr16[0] = d[8];
1994:       p->Addr16[1] = d[9];
1995:
1996:       p->status = d[offset];
1997:
1998:       /* copy in the data */
1999:       p->datalen = i - (offset + 1);
2000:       for (;i>offset;i--) {
2001:         p->data[i-(offset + 1)] = d[i];
2002:       }
2003:
2004:       /* ########################################## */
2005:       /* if: Unknown */
2006:     } else {
2007:       xbee_logE("Packet type: Unknown (0x%02X)",t);
2008:       continue;
2009:     }
2010:     p->next = NULL;
2011:
2012:     /* lock the connection mutex */
2013:     xbee_mutex_lock(xbee->conmutex);
2014:
2015:     hasCon = 0;
2016:     if (p->isBroadcastADR || p->isBroadcastPAN) {
2017:       unsigned char t[8] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
2018:       /* if the packet was broadcast, search for a broadcast accepting connection */
2019:       con = xbee->conlist;
2020:       while (con) {
2021:         if (con->type == p->type &&
2022:             (con->type == xbee_16bitData || con->type == xbee_64bitData) &&
2023:             ((con->tAddr64 && !memcmp(con->tAddr,t,8)) ||
2024:              (!con->tAddr64 && !memcmp(con->tAddr,t,2)))) {
2025:           hasCon = 1;
2026:           xbee_logI("Found broadcasting connection @ 0x%08X",con);
2027:           break;
2028:         }
2029:         con = con->next;
2030:       }
2031:     }
2032:     if (!hasCon || !con) {
2033:       con = xbee->conlist;
2034:       while (con) {
2035:         if (xbee_matchpktcon(xbee, p, con)) {
2036:           hasCon = 1;
2037:           break;
2038:         }
2039:         con = con->next;
2040:       }
```

```
2041:     }
2042:
2043:     /* unlock the connection mutex */
2044:     xbee_mutex_unlock(xbee->conmutex);
2045:
2046:     /* if the packet doesn't have a connection, don't add it! */
2047:     if (!hasCon) {
2048:       xbee_logE("Connectionless packet... discarding!");
2049:       continue;
2050:     }
2051:
2052:     /* if the connection has a callback function then it is passed the packet
2053:        and the packet is not added to the list */
2054:     if (con && con->callback) {
2055:       t_callback_list *l, *q;
2056:
2057:       xbee_mutex_lock(con->callbackListmutex);
2058:       l = con->callbackList;
2059:       q = NULL;
2060:       while (l) {
2061:         q = l;
2062:         l = l->next;
2063:       }
2064:       l = Xcalloc(sizeof(t_callback_list));
2065:       l->pkt = p;
2066:       if (!con->callbackList || q == NULL) {
2067:         con->callbackList = l;
2068:       } else {
2069:         q->next = l;
2070:       }
2071:       xbee_mutex_unlock(con->callbackListmutex);
2072:
2073:       xbee_logI("Using callback function!");
2074:       xbee_logI("  info block @ 0x%08X",l);
2075:       xbee_logI("  function   @ 0x%08X",con->callback);
2076:       xbee_logI("  connection @ 0x%08X",con);
2077:       xbee_logE("  packet     @ 0x%08X",p);
2078:
2079:       /* if the callback thread not still running, then start a new one! */
2080:       if (!xbee_mutex_trylock(con->callbackmutex)) {
2081:         xbee_thread_t t;
2082:         int ret;
2083:         t_threadList *p, *q;
2084:         t_CBinfo info;
2085:         info.xbee = xbee;
2086:         info.con = con;
2087:         xbee_log("Starting new callback thread!");
2088:         if ((ret = xbee_thread_create(t,xbee_callbackWrapper,&info)) != 0) {
2089:           xbee_mutex_unlock(con->callbackmutex);
2090:           /* this MAY help with future attempts... */
2091:           xbee_sem_post(xbee->threadsem);
2092:           xbee_logS("An error occured while starting thread (%d)... Out of resources?", ret);
2093:           xbee_logE("This packet has been lost!");
2094:           continue;
2095:         }
2096:         xbee_log("Started thread 0x%08X!", t);
2097:         xbee_mutex_lock(xbee->threadmutex);
2098:         p = xbee->threadList;
2099:         q = NULL;
2100:         while (p) {
2101:           q = p;
2102:           p = p->next;
2103:         }
2104:         p = Xcalloc(sizeof(t_threadList));
2105:         if (q == NULL) {
2106:           xbee->threadList = p;
2107:         } else {
2108:           q->next = p;
2109:         }
2110:         p->thread = t;
2111:         p->next = NULL;
2112:         xbee_mutex_unlock(xbee->threadmutex);
2113:       } else {
2114:         xbee_logE("Using existing callback thread... callback has been scheduled.");
2115:       }
2116:       /* prevent the packet from being free'd */
2117:       p = NULL;
2118:       continue;
2119:     }
2120:
2121:     /* lock the packet mutex, so we can safely add the packet to the list */
2122:     xbee_mutex_lock(xbee->pktmutex);
2123:
2124:     /* if: the list is empty */
2125:     if (!xbee->pktlist) {
```

```
2126:       /* start the list! */
2127:       xbee->pktlist = p;
2128:     } else if (xbee->pktlast) {
2129:       /* add the packet to the end */
2130:       xbee->pktlast->next = p;
2131:     } else {
2132:       /* pktlast wasnt set... look for the end and then set it */
2133:       i = 0;
2134:       q = xbee->pktlist;
2135:       while (q->next) {
2136:         q = q->next;
2137:         i++;
2138:       }
2139:       q->next = p;
2140:       xbee->pktcount = i;
2141:     }
2142:     xbee->pktlast = p;
2143:     xbee->pktcount++;
2144:
2145:     /* unlock the packet mutex */
2146:     xbee_mutex_unlock(xbee->pktmutex);
2147:
2148:     xbee_logI("--=========================--");
2149:     xbee_logE("Packets: %d",xbee->pktcount);
2150:
2151:     p = q = NULL;
2152:   }
2153:   return 0;
2154: }
2155:
2156: static void xbee_callbackWrapper(t_CBinfo *info) {
2157:   xbee_hnd xbee;
2158:   xbee_con *con;
2159:   xbee_pkt *pkt;
2160:   t_callback_list *temp;
2161:   xbee = info->xbee;
2162:   con = info->con;
2163:   /* dont forget! the callback mutex is already locked... by the parent thread :) */
2164:   xbee_mutex_lock(con->callbackListmutex);
2165:   while (con->callbackList) {
2166:     /* shift the list along 1 */
2167:     temp = con->callbackList;
2168:     con->callbackList = temp->next;
2169:     xbee_mutex_unlock(con->callbackListmutex);
2170:     /* get the packet */
2171:     pkt = temp->pkt;
2172:
2173:     xbee_logS("Starting callback function...");
2174:     xbee_logI("  info block @ 0x%08X",temp);
2175:     xbee_logI("  function   @ 0x%08X",con->callback);
2176:     xbee_logI("  connection @ 0x%08X",con);
2177:     xbee_logE("  packet     @ 0x%08X",pkt);
2178:     Xfree(temp);
2179:     if (con->callback) {
2180:       con->callback(con,pkt);
2181:       xbee_log("Callback complete!");
2182:       if (!con->noFreeAfterCB) Xfree(pkt);
2183:     } else {
2184:       xbee_pkt *q;
2185:       int i;
2186:       xbee_log("Callback function was removed! Appending packet to main list...");
2187:       /* lock the packet mutex, so we can safely add the packet to the list */
2188:       xbee_mutex_lock(xbee->pktmutex);
2189:
2190:       /* if: the list is empty */
2191:       if (!xbee->pktlist) {
2192:         /* start the list! */
2193:         xbee->pktlist = pkt;
2194:       } else if (xbee->pktlast) {
2195:         /* add the packet to the end */
2196:         xbee->pktlast->next = pkt;
2197:       } else {
2198:         /* pktlast wasnt set... look for the end and then set it */
2199:         i = 0;
2200:         q = xbee->pktlist;
2201:         while (q->next) {
2202:           q = q->next;
2203:           i++;
2204:         }
2205:         q->next = pkt;
2206:         xbee->pktcount = i;
2207:       }
2208:       xbee->pktlast = pkt;
2209:       xbee->pktcount++;
2210:
```

```
2211:       /* unlock the packet mutex */
2212:       xbee_mutex_unlock(xbee->pktmutex);
2213:     }
2214:
2215:     xbee_mutex_lock(con->callbackListmutex);
2216:   }
2217:   xbee_mutex_unlock(con->callbackListmutex);
2218:
2219:   xbee_log("Callback thread ending...");
2220:   /* releasing the thread mutex is the last thing we do! */
2221:   xbee_mutex_unlock(con->callbackmutex);
2222:
2223:   if (con->destroySelf) {
2224:     _xbee_endcon2(xbee,&con,1);
2225:   }
2226:   xbee_sem_post(xbee->threadsem);
2227: }
2228:
2229: /* ########################################################################
2230:    xbee_thread_watch - INTERNAL
2231:    watches for dead threads and tidies up */
2232: static void xbee_thread_watch(xbee_hnd xbee) {
2233:
2234: #ifdef _WIN32 /* ---- */
2235:   /* win32 requires this delay... no idea why */
2236:   usleep(1000000);
2237: #endif /* ----------- */
2238:
2239:   xbee_mutex_init(xbee->threadmutex);
2240:   xbee_sem_init(xbee->threadsem);
2241:
2242:   while (xbee->run) {
2243:     t_threadList *p, *q, *t;
2244:     xbee_mutex_lock(xbee->threadmutex);
2245:     p = xbee->threadList;
2246:     q = NULL;
2247:
2248:     while (p) {
2249:       t = p;
2250:       p = p->next;
2251:       if (!(xbee_thread_tryjoin(t->thread))) {
2252:         xbee_log("Joined with thread 0x%08X...",t->thread);
2253:         if (t == xbee->threadList) {
2254:           xbee->threadList = t->next;
2255:         } else if (q) {
2256:           q->next = t->next;
2257:         }
2258:         free(t);
2259:       } else {
2260:         q = t;
2261:       }
2262:     }
2263:
2264:     xbee_mutex_unlock(xbee->threadmutex);
2265:     xbee_sem_wait(xbee->threadsem);
2266:     usleep(100000); /* 100ms to allow the thread to end before we try to join */
2267:   }
2268:
2269:   xbee_mutex_destroy(xbee->threadmutex);
2270:   xbee_sem_destroy(xbee->threadsem);
2271: }
2272:
2273:
2274: /* ########################################################################
2275:    xbee_getbyte - INTERNAL
2276:    waits for an escaped byte of data */
2277: static unsigned char xbee_getbyte(xbee_hnd xbee) {
2278:   unsigned char c;
2279:
2280:   /* take a byte */
2281:   c = xbee_getrawbyte(xbee);
2282:   /* if its escaped, take another and un-escape */
2283:   if (c == 0x7D) c = xbee_getrawbyte(xbee) ^ 0x20;
2284:
2285:   return (c & 0xFF);
2286: }
2287:
2288: /* ########################################################################
2289:    xbee_getrawbyte - INTERNAL
2290:    waits for a raw byte of data */
2291: static unsigned char xbee_getrawbyte(xbee_hnd xbee) {
2292:   int ret;
2293:   unsigned char c = 0x00;
2294:
2295:   /* the loop is just incase there actually isnt a byte there to be read... */
```

```
2296:   do {
2297:     /* wait for a read to be possible */
2298:     if ((ret = xbee_select(xbee,NULL)) == -1) {
2299:       xbee_perror("libxbee:xbee_getrawbyte()");
2300:       exit(1);
2301:     }
2302:     if (!xbee->run) break;
2303:     if (ret == 0) continue;
2304:
2305:     /* read 1 character */
2306:     if (xbee_read(xbee,&c,1) == 0) {
2307:       /* for some reason no characters were read... */
2308:       if (xbee_ferror(xbee) || xbee_feof(xbee)) {
2309:         xbee_log("Error or EOF detected");
2310:         fprintf(stderr,"libxbee:xbee_read(): Error or EOF detected\n");
2311:         exit(1); /* this should have something nicer... */
2312:       }
2313:       /* no error... try again */
2314:       usleep(10);
2315:       continue;
2316:     }
2317:   } while (0);
2318:
2319:   return (c & 0xFF);
2320: }
2321:
2322: /* ####################################################################
2323:    _xbee_send_pkt - INTERNAL
2324:    sends a complete packet of data */
2325: static int _xbee_send_pkt(xbee_hnd xbee, t_data *pkt, xbee_con *con) {
2326:   int retval = 0;
2327:
2328:   /* lock connection mutex */
2329:   xbee_mutex_lock(con->Txmutex);
2330:   /* lock the send mutex */
2331:   xbee_mutex_lock(xbee->sendmutex);
2332:
2333:   /* write and flush the data */
2334:   xbee_write(xbee,pkt->data,pkt->length);
2335:
2336:   /* unlock the mutex */
2337:   xbee_mutex_unlock(xbee->sendmutex);
2338:
2339:   xbee_logSf();
2340:   if (xbee->log) {
2341:     int i,x,y;
2342:     /* prints packet in hex byte-by-byte */
2343:     xbee_logIc("TX Packet:");
2344:     for (i=0,x=0,y=0;i<pkt->length;i++,x--) {
2345:       if (x == 0) {
2346:         fprintf(xbee->log,"\n  0x%04X | ",y);
2347:         x = 0x8;
2348:         y += x;
2349:       }
2350:       if (x == 4) {
2351:         fprintf(xbee->log,"  ");
2352:       }
2353:       fprintf(xbee->log,"0x%02X ",pkt->data[i]);
2354:     }
2355:     xbee_logIcf();
2356:   }
2357:   xbee_logEf();
2358:
2359:   if (con->waitforACK &&
2360:       ((con->type == xbee_16bitData) ||
2361:        (con->type == xbee_64bitData))) {
2362:     con->ACKstatus = 0xFF; /* waiting */
2363:     xbee_log("Waiting for ACK/NAK response...");
2364:     xbee_sem_wait1sec(con->waitforACKsem);
2365:     switch (con->ACKstatus) {
2366:       case 0: xbee_log("ACK recieved!"); break;
2367:       case 1: xbee_log("NAK recieved..."); break;
2368:       case 2: xbee_log("CCA failure..."); break;
2369:       case 3: xbee_log("Purged..."); break;
2370:       case 255: default: xbee_log("Timeout...");
2371:     }
2372:     if (con->ACKstatus) retval = 1; /* error */
2373:   }
2374:
2375:   /* unlock connection mutex */
2376:   xbee_mutex_unlock(con->Txmutex);
2377:
2378:   /* free the packet */
2379:   Xfree(pkt);
2380:
```

```
2381:   return retval;
2382: }
2383:
2384: /* ###################################################################
2385:    xbee_make_pkt - INTERNAL
2386:    adds delimiter field
2387:    calculates length and checksum
2388:    escapes bytes */
2389: static t_data *xbee_make_pkt(xbee_hnd xbee, unsigned char *data, int length) {
2390:   t_data *pkt;
2391:   unsigned int l, i, o, t, x, m;
2392:   char d = 0;
2393:
2394:   /* check the data given isnt too long
2395:      100 bytes maximum payload + 12 bytes header information */
2396:   if (length > 100 + 12) return NULL;
2397:
2398:   /* calculate the length of the whole packet
2399:      start, length (MSB), length (LSB), DATA, checksum */
2400:   l = 3 + length + 1;
2401:
2402:   /* prepare memory */
2403:   pkt = Xcalloc(sizeof(t_data));
2404:
2405:   /* put start byte on */
2406:   pkt->data[0] = 0x7E;
2407:
2408:   /* copy data into packet */
2409:   for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
2410:     /* if: its time for the checksum */
2411:     if (i == length) d = M8((0xFF - M8(t)));
2412:     /* if: its time for the high length byte */
2413:     else if (m == 1) d = M8(length >> 8);
2414:     /* if: its time for the low length byte */
2415:     else if (m == 2) d = M8(length);
2416:     /* if: its time for the normal data */
2417:     else if (m > 2) d = data[i];
2418:
2419:     x = 0;
2420:     /* check for any escapes needed */
2421:     if ((d == 0x11) || /* XON */
2422:         (d == 0x13) || /* XOFF */
2423:         (d == 0x7D) || /* Escape */
2424:         (d == 0x7E)) { /* Frame Delimiter */
2425:       l++;
2426:       pkt->data[o++] = 0x7D;
2427:       x = 1;
2428:     }
2429:
2430:     /* move data in */
2431:     pkt->data[o] = ((!x)?d:d^0x20);
2432:     if (m > 2) {
2433:       i++;
2434:       t += d;
2435:     }
2436:   }
2437:
2438:   /* remember the length */
2439:   pkt->length = l;
2440:
2441:   return pkt;
2442: }
```