```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/>.
*/

#define SVN_REV "$Id: api.c 398 2010-07-26 20:54:15Z attie.co.uk $"

#include "api.h"

#ifdef __GNUC__ /* ---- */
#include "xsys/linux.c"
#else /* -------------- */
#include "xsys\win32.c"
#endif /* ------------- */

const char *xbee_svn_version(void) {
  return HOST_OS " - " SVN_REV;
}

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

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

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

/* realloc wrapper function */
static void *Xrealloc(void *ptr, size_t size) {
  void *t;
  t = realloc(ptr,size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    perror("libxbee:realloc()");
    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;
}

/* ######################################################################## */
/* ### Helper Functions ################################################### */
/* ######################################################################## */

```

```
 86: /* ################################################################### */
 87:    returns 1 if the packet has data for the digital input else 0 */
 88: int xbee_hasdigital(xbee_pkt *pkt, int sample, int input) {
 89:   int mask = 0x0001;
 90:   if (input < 0 || input > 7) return 0;
 91:   if (sample >= pkt->samples) return 0;
 92:
 93:   mask <<= input;
 94:   return !!(pkt->IOdata[sample].IOmask & mask);
 95: }
 96:
 97: /* ################################################################### */
 98:    returns 1 if the digital input is high else 0 (or 0 if no digital data present) */
 99: int xbee_getdigital(xbee_pkt *pkt, int sample, int input) {
100:   int mask = 0x0001;
101:   if (!xbee_hasdigital(pkt,sample,input)) return 0;
102:
103:   mask <<= input;
104:   return !!(pkt->IOdata[sample].IOdigital & mask);
105: }
106:
107: /* ################################################################### */
108:    returns 1 if the packet has data for the analog input else 0 */
109: int xbee_hasanalog(xbee_pkt *pkt, int sample, int input) {
110:   int mask = 0x0200;
111:   if (input < 0 || input > 5) return 0;
112:   if (sample >= pkt->samples) return 0;
113:
114:   mask <<= input;
115:   return !!(pkt->IOdata[sample].IOmask & mask);
116: }
117:
118: /* ################################################################### */
119:    returns analog input as a voltage if vRef is non-zero, else raw value (or 0 if no analog data present) */
120: double xbee_getanalog(xbee_pkt *pkt, int sample, int input, double Vref) {
121:   if (!xbee_hasanalog(pkt,sample,input)) return 0;
122:
123:   if (Vref) return (Vref / 1023) * pkt->IOdata[sample].IOanalog[input];
124:   return pkt->IOdata[sample].IOanalog[input];
125: }
126:
127: /* ################################################################### */
128: /* ### XBee Functions ############################################### */
129: /* ################################################################### */
130:
131: static void xbee_logf(const char *logformat, int unlock, const char *file,
132:                       const int line, const char *function, char *format, ...) {
133:   char buf[128];
134:   va_list ap;
135:   FILE *log;
136:   va_start(ap,format);
137:   vsnprintf(buf,127,format,ap);
138:   va_end(ap);
139:   if (xbee.log) {
140:     log = xbee.log;
141:   } else {
142:     log = stderr;
143:   }
144:   xbee_mutex_lock(xbee.logmutex);
145:   fprintf(log,logformat,file,line,function,buf);
146:   if (unlock) xbee_mutex_unlock(xbee.logmutex);
147: }
148:
149: /* ################################################################### */
150:    xbee_sendAT - INTERNAL
151:    allows for an at command to be send, and the reply to be captured */
152: static int xbee_sendAT(char *command, char *retBuf, int retBuflen) {
153:   return xbee_sendATdelay(0,command,retBuf, retBuflen);
154: }
155: static int xbee_sendATdelay(int guartTime, char *command, char *retBuf, int retBuflen) {
156:   struct timeval to;
157:
158:   int ret;
159:   int bufi = 0;
160:
161:   /* if there is a guartTime given, then use it and a bit more */
162:   if (guartTime) usleep(guartTime * 1200);
163:
164:   /* get rid of any pre-command sludge... */
165:   memset(&to, 0, sizeof(to));
166:   ret = xbee_select(&to);
167:   if (ret > 0) {
168:     char t[128];
169:     while (xbee_read(t,127));
170:   }
```

```
171:
172:   /* send the requested command */
173:   if (xbee.log) xbee_log("sendATdelay: Sending '%s'", command);
174:   xbee_write(command, strlen(command));
175:
176:   /* if there is a guartTime, then use it */
177:   if (guartTime) {
178:     usleep(guartTime * 900);
179:
180:     /* get rid of any post-command sludge... */
181:     memset(&to, 0, sizeof(to));
182:     ret = xbee_select(&to);
183:     if (ret > 0) {
184:       char t[128];
185:       while (xbee_read(t,127));
186:     }
187:   }
188:
189:   /* retrieve the data */
190:   memset(retBuf, 0, retBuflen);
191:   memset(&to, 0, sizeof(to));
192:   if (guartTime) {
193:     /* select on the xbee fd... wait at most 0.2 the guartTime for the response */
194:     to.tv_usec = guartTime * 200;
195:   } else {
196:     /* or 250ms */
197:     to.tv_usec = 250000;
198:   }
199:   if ((ret = xbee_select(&to)) == -1) {
200:     perror("libxbee:xbee_sendATdelay()");
201:     exit(1);
202:   }
203:
204:   if (!ret) {
205:     /* timed out, and there is nothing to be read */
206:     if (xbee.log) xbee_log("sendATdelay: No Data to read - Timeout...");
207:     return 1;
208:   }
209:
210:   /* check for any dribble... */
211:   do {
212:     /* if there is actually no space in the retBuf then break out */
213:     if (bufi >= retBuflen - 1) {
214:       break;
215:     }
216:
217:     /* read as much data as is possible into retBuf */
218:     if ((ret = xbee_read(&retBuf[bufi], retBuflen - bufi - 1)) == 0) {
219:       break;
220:     }
221:
222:     /* advance the 'end of string' pointer */
223:     bufi += ret;
224:
225:     /* wait at most 150ms for any more data */
226:     memset(&to, 0, sizeof(to));
227:     to.tv_usec = 150000;
228:     if ((ret = xbee_select(&to)) == -1) {
229:       perror("libxbee:xbee_sendATdelay()");
230:       exit(1);
231:     }
232:
233:     /* loop while data was read */
234:   } while (ret);
235:
236:   if (!bufi) {
237:     if (xbee.log) xbee_log("sendATdelay: No response...");
238:     return 1;
239:   }
240:
241:   /* terminate the string */
242:   retBuf[bufi] = '\0';
243:
244:   if (xbee.log) xbee_log("sendATdelay: Recieved '%s'",retBuf);
245:   return 0;
246: }
247:
248:
249: /* ################################################################
250:    xbee_start
251:    sets up the correct API mode for the xbee
252:    cmdSeq  = CC
253:    cmdTime = GT */
254: static int xbee_startAPI(void) {
255:   char buf[256];
```

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

```
341:   pkt = xbee.pktlist;
342:   xbee.pktlist = NULL;
343:   while (pkt) {
344:     npkt = pkt->next;
345:     Xfree(pkt);
346:     pkt = npkt;
347:   }
348:
349:   /* destroy mutexes */
350:   xbee_mutex_destroy(xbee.conmutex);
351:   xbee_mutex_destroy(xbee.pktmutex);
352:   xbee_mutex_destroy(xbee.sendmutex);
353:
354:   /* close the serial port */
355:   Xfree(xbee.path);
356: #ifdef __GNUC__ /* ---- */
357:   if (xbee.tty) xbee_close(xbee.tty);
358:   if (xbee.ttyfd) close(xbee.ttyfd);
359: #else /* -------------- */
360:   if (xbee.tty) CloseHandle(xbee.tty);
361: #endif /* ------------- */
362:
363:   /* close log and tty */
364:   if (xbee.log) {
365:     xbee_log("libxbee: Stopped! (%s)",xbee_svn_version());
366:     fflush(xbee.log);
367:     xbee_close(xbee.log);
368:   }
369:   xbee_mutex_destroy(xbee.logmutex);
370:
371:   /* wipe everything else... */
372:   memset(&xbee,0,sizeof(xbee));
373:
374:   return ret;
375: }
376:
377: /* ####################################################################
378:    xbee_setup
379:    opens xbee serial port & creates xbee listen thread
380:    the xbee must be configured for API mode 2
381:    THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
382: int xbee_setup(char *path, int baudrate) {
383:   return xbee_setuplogAPI(path,baudrate,0,0,0);
384: }
385: int xbee_setuplog(char *path, int baudrate, int logfd) {
386:   return xbee_setuplogAPI(path,baudrate,logfd,0,0);
387: }
388: int xbee_setupAPI(char *path, int baudrate, char cmdSeq, int cmdTime) {
389:   return xbee_setuplogAPI(path,baudrate,0,cmdSeq,cmdTime);
390: }
391: int xbee_setuplogAPI(char *path, int baudrate, int logfd, char cmdSeq, int cmdTime) {
392:   t_info info;
393:   int ret;
394:
395:   memset(&xbee,0,sizeof(xbee));
396:
397: #ifdef DEBUG
398:   /* logfd or stderr */
399:   xbee.logfd = ((logfd)?logfd:2);
400: #else
401:   xbee.logfd = logfd;
402: #endif
403:   xbee_mutex_init(xbee.logmutex);
404:   if (xbee.logfd) {
405:     xbee.log = fdopen(xbee.logfd,"w");
406:     if (!xbee.log) {
407:       /* errno == 9 is bad file descriptor (probrably not provided) */
408:       if (errno != 9) perror("xbee_setup(): Failed opening logfile");
409:       xbee.logfd = 0;
410:     } else {
411: #ifdef __GNUC__ /* ---- */
412:       /* set to line buffer - ensure lines are written to file when complete */
413:       setvbuf(xbee.log,NULL,_IOLBF,BUFSIZ);
414: #else /* -------------- */
415:       /* Win32 is rubbish... so we have to completely disable buffering... */
416:       setvbuf(xbee.log,NULL,_IONBF,BUFSIZ);
417: #endif /* ------------- */
418:     }
419:   }
420:
421:   if (xbee.log) xbee_log("libxbee: Starting (%s)...",xbee_svn_version());
422:
423:   /* setup the connection stuff */
424:   xbee.conlist = NULL;
425:
```

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

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

```
596:         /* if: after a txStatus and frameIDs match! */
597:       } else if ((type == xbee_txStatus) &&
598:                  (con->type == type) &&
599:                  (frameID == con->frameID)) {
600:         xbee_mutex_unlock(xbee.conmutex);
601:         return con;
602:
603:         /* if: after a localAT, and the frameIDs match! */
604:       } else if ((type == xbee_localAT) &&
605:                  (con->type == type) &&
606:                  (frameID == con->frameID)) {
607:         xbee_mutex_unlock(xbee.conmutex);
608:         return con;
609:
610:         /* if: connection types match, the frameIDs match, and the addresses match! */
611:       } else if ((type == con->type) &&
612:                  (frameID == con->frameID) &&
613:                  (!memcmp(tAddr,con->tAddr,8))) {
614:         xbee_mutex_unlock(xbee.conmutex);
615:         return con;
616:       }
617:
618:       /* if there are more, move along, dont want to loose that last item! */
619:       if (con->next == NULL) break;
620:       con = con->next;
621:     }
622:
623:     /* keep hold of the last connection... we will need to link it up later */
624:     ocon = con;
625:   }
626:
627:   /* create a new connection and set its attributes */
628:   con = Xcalloc(sizeof(xbee_con));
629:   con->type = type;
630:   /* is it a 64bit connection? */
631:   if ((type == xbee_64bitRemoteAT) ||
632:       (type == xbee_64bitData) ||
633:       (type == xbee_64bitIO)) {
634:     con->tAddr64 = TRUE;
635:   }
636:   con->atQueue = 0; /* queue AT commands? */
637:   con->txDisableACK = 0; /* disable ACKs? */
638:   con->txBroadcast = 0; /* broadcast? */
639:   con->frameID = frameID;
640:   memcpy(con->tAddr,tAddr,8); /* copy in the remote address */
641:   xbee_mutex_init(con->callbackmutex);
642:
643:   if (xbee.log) {
644:     switch(type) {
645:     case xbee_localAT:
646:       xbee_log("New local AT connection!");
647:       break;
648:     case xbee_16bitRemoteAT:
649:     case xbee_64bitRemoteAT:
650:       xbee_logc("New %d-bit remote AT connection! (to: ",(con->tAddr64?64:16));
651:       for (i=0;i<(con->tAddr64?8:2);i++) {
652:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
653:       }
654:       fprintf(xbee.log,")");
655:       xbee_logcf();
656:       break;
657:     case xbee_16bitData:
658:     case xbee_64bitData:
659:       xbee_logc("New %d-bit data connection! (to: ",(con->tAddr64?64:16));
660:       for (i=0;i<(con->tAddr64?8:2);i++) {
661:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
662:       }
663:       fprintf(xbee.log,")");
664:       xbee_logcf();
665:       break;
666:     case xbee_16bitIO:
667:     case xbee_64bitIO:
668:       xbee_logc("New %d-bit IO connection! (to: ",(con->tAddr64?64:16));
669:       for (i=0;i<(con->tAddr64?8:2);i++) {
670:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
671:       }
672:       fprintf(xbee.log,")");
673:       xbee_logcf();
674:       break;
675:     case xbee_txStatus:
676:       xbee_log("New Tx status connection!");
677:       break;
678:     case xbee_modemStatus:
679:       xbee_log("New modem status connection!");
680:       break;
```

```
681:     case xbee_unknown:
682:     default:
683:       xbee_log("New unknown connection!");
684:     }
685:   }
686:
687:   /* make it the last in the list */
688:   con->next = NULL;
689:   /* add it to the list */
690:   if (xbee.conlist) {
691:     ocon->next = con;
692:   } else {
693:     xbee.conlist = con;
694:   }
695:
696:   /* unlock the mutex */
697:   xbee_mutex_unlock(xbee.conmutex);
698:   return con;
699: }
700:
701: /* ################################################################### */
702:    xbee_conflush
703:    removes any packets that have been collected for the specified
704:    connection */
705: void xbee_flushcon(xbee_con *con) {
706:   xbee_pkt *r, *p, *n;
707:
708:   /* lock the packet mutex */
709:   xbee_mutex_lock(xbee.pktmutex);
710:
711:   /* if: there are packets */
712:   if ((p = xbee.pktlist) != NULL) {
713:     r = NULL;
714:     /* get all packets for this connection */
715:     do {
716:       /* does the packet match the connection? */
717:       if (xbee_matchpktcon(p,con)) {
718:         /* if it was the first packet */
719:         if (!r) {
720:           /* move the chain along */
721:           xbee.pktlist = p->next;
722:         } else {
723:           /* otherwise relink the list */
724:           r->next = p->next;
725:         }
726:         xbee.pktcount--;
727:
728:         /* free this packet! */
729:         n = p->next;
730:         Xfree(p);
731:         /* move on */
732:         p = n;
733:       } else {
734:         /* move on */
735:         r = p;
736:         p = p->next;
737:       }
738:     } while (p);
739:     xbee.pktlast = r;
740:   }
741:
742:   /* unlock the packet mutex */
743:   xbee_mutex_unlock(xbee.pktmutex);
744: }
745:
746: /* ################################################################### */
747:    xbee_endcon
748:    close the unwanted connection
749:    free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
750: void xbee_endcon2(xbee_con **con, int skipUnlink) {
751:   xbee_con *t, *u;
752:
753:   if (!skipUnlink) {
754:     /* lock the connection mutex */
755:     xbee_mutex_lock(xbee.conmutex);
756:
757:     u = t = xbee.conlist;
758:     while (t && t != *con) {
759:       u = t;
760:       t = t->next;
761:     }
762:     if (!t) {
763:       /* invalid connection given... */
764:       if (xbee.log) {
765:         xbee_log("Attempted to close invalid connection...");
```

```
766:       }
767:       /* unlock the connection mutex */
768:       xbee_mutex_unlock(xbee.conmutex);
769:       return;
770:     }
771:     /* extract this connection from the list */
772:     u->next = t->next;
773:
774:     /* unlock the connection mutex */
775:     xbee_mutex_unlock(xbee.conmutex);
776:   }
777:
778:   /* check if a callback thread is running... */
779:   if (t->callback && xbee_mutex_trylock(t->callbackmutex)) {
780:     /* if it is running... tell it to destroy the connection on completion */
781:     xbee_log("Attempted to close a connection with active callbacks... "
782:              "Connection will be destroied when callbacks have completeted...");
783:     t->destroySelf = 1;
784:     return;
785:   }
786:
787:   /* remove all packets for this connection */
788:   xbee_flushcon(t);
789:
790:   /* destroy the callback mutex */
791:   xbee_mutex_destroy(t->callbackmutex);
792:
793:   /* free the connection! */
794:   Xfree(*con);
795: }
796:
797: /* ####################################################################
798:    xbee_senddata
799:    send the specified data to the provided connection */
800: int xbee_senddata(xbee_con *con, char *format, ...) {
801:   int ret;
802:   va_list ap;
803:
804:   ISREADY;
805:
806:   /* xbee_vsenddata() wants a va_list... */
807:   va_start(ap, format);
808:   /* hand it over :) */
809:   ret = xbee_vsenddata(con,format,ap);
810:   va_end(ap);
811:   return ret;
812: }
813:
814: int xbee_vsenddata(xbee_con *con, char *format, va_list ap) {
815:   unsigned char data[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
816:   int length;
817:
818:   ISREADY;
819:
820:   /* make up the data and keep the length, its possible there are nulls in there */
821:   length = vsnprintf((char *)data,128,format,ap);
822:
823:   /* hand it over :) */
824:   return xbee_nsenddata(con,(char *)data,length);
825: }
826:
827: int xbee_nsenddata(xbee_con *con, char *data, int length) {
828:   t_data *pkt;
829:   int i;
830:   unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
831:
832:   ISREADY;
833:
834:   if (!con) return -1;
835:   if (con->type == xbee_unknown) return -1;
836:   if (length > 127) return -1;
837:
838:
839:   if (xbee.log) {
840:     xbee_log("--== TX Packet ============--");
841:     xbee_logc("Connection Type: ");
842:     switch (con->type) {
843:     case xbee_unknown:       fprintf(xbee.log,"Unknown"); break;
844:     case xbee_localAT:       fprintf(xbee.log,"Local AT"); break;
845:     case xbee_remoteAT:      fprintf(xbee.log,"Remote AT"); break;
846:     case xbee_16bitRemoteAT: fprintf(xbee.log,"Remote AT (16-bit)"); break;
847:     case xbee_64bitRemoteAT: fprintf(xbee.log,"Remote AT (64-bit)"); break;
848:     case xbee_16bitData:     fprintf(xbee.log,"Data (16-bit)"); break;
849:     case xbee_64bitData:     fprintf(xbee.log,"Data (64-bit)"); break;
850:     case xbee_16bitIO:       fprintf(xbee.log,"IO (16-bit)"); break;
```

```
851:     case xbee_64bitIO:       fprintf(xbee.log,"IO (64-bit)"); break;
852:     case xbee_txStatus:      fprintf(xbee.log,"Tx Status"); break;
853:     case xbee_modemStatus:   fprintf(xbee.log,"Modem Status"); break;
854:     }
855:     xbee_logcf();
856:     xbee_logc("Destination: ");
857:     for (i=0;i<(con->tAddr64?8:2);i++) {
858:       fprintf(xbee.log,(i?":%02X":"%02X"),con->tAddr[i]);
859:     }
860:     xbee_logcf();
861:     xbee_log("Length: %d",length);
862:     for (i=0;i<length;i++) {
863:       xbee_logc("%3d | 0x%02X ",i,(unsigned char)data[i]);
864:       if ((data[i] > 32) && (data[i] < 127)) {
865:         fprintf(xbee.log,"'%c'",data[i]);
866:       } else{
867:         fprintf(xbee.log," _");
868:       }
869:       xbee_logcf();
870:     }
871:   }
872:
873:   /* ######################################### */
874:   /* if: local AT */
875:   if (con->type == xbee_localAT) {
876:     /* AT commands are 2 chars long (plus optional parameter) */
877:     if (length < 2) return -1;
878:
879:     /* use the command? */
880:     buf[0] = ((!con->atQueue)?XBEE_LOCAL_ATREQ:XBEE_LOCAL_ATQUE);
881:     buf[1] = con->frameID;
882:
883:     /* copy in the data */
884:     for (i=0;i<length;i++) {
885:       buf[i+2] = data[i];
886:     }
887:
888:     /* setup the packet */
889:     pkt = xbee_make_pkt(buf,i+2);
890:     /* send it on */
891:     xbee_send_pkt(pkt);
892:
893:     return 0;
894:
895:     /* ######################################### */
896:     /* if: remote AT */
897:   } else if ((con->type == xbee_16bitRemoteAT) ||
898:              (con->type == xbee_64bitRemoteAT)) {
899:     if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter) */
900:     buf[0] = XBEE_REMOTE_ATREQ;
901:     buf[1] = con->frameID;
902:
903:     /* copy in the relevant address */
904:     if (con->tAddr64) {
905:       memcpy(&buf[2],con->tAddr,8);
906:       buf[10] = 0xFF;
907:       buf[11] = 0xFE;
908:     } else {
909:       memset(&buf[2],0,8);
910:       memcpy(&buf[10],con->tAddr,2);
911:     }
912:     /* queue the command? */
913:     buf[12] = ((!con->atQueue)?0x02:0x00);
914:
915:     /* copy in the data */
916:     for (i=0;i<length;i++) {
917:       buf[i+13] = data[i];
918:     }
919:
920:     /* setup the packet */
921:     pkt = xbee_make_pkt(buf,i+13);
922:     /* send it on */
923:     xbee_send_pkt(pkt);
924:
925:     return 0;
926:
927:     /* ######################################### */
928:     /* if: 16 or 64bit Data */
929:   } else if ((con->type == xbee_16bitData) ||
930:              (con->type == xbee_64bitData)) {
931:     int offset;
932:
933:     /* if: 16bit Data */
934:     if (con->type == xbee_16bitData) {
935:       buf[0] = XBEE_16BIT_DATATX;
```

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

```
1021:     l = p;
1022:     p = p->next;
1023:   } while (p);
1024:
1025:   /* if: no packet was found */
1026:   if (!q) {
1027:     xbee_mutex_unlock(xbee.pktmutex);
1028:     return NULL;
1029:   }
1030:
1031:   /* if it was the first packet */
1032:   if (l) {
1033:     /* relink the list */
1034:     l->next = p->next;
1035:     if (!l->next) xbee.pktlast = l;
1036:   } else {
1037:     /* move the chain along */
1038:     xbee.pktlist = p->next;
1039:     if (!xbee.pktlist) {
1040:       xbee.pktlast = NULL;
1041:     } else if (!xbee.pktlist->next) {
1042:       xbee.pktlast = xbee.pktlist;
1043:     }
1044:   }
1045:   xbee.pktcount--;
1046:
1047:   /* unlink this packet from the chain! */
1048:   q->next = NULL;
1049:
1050:   if (xbee.log) {
1051:     xbee_log("--== Get Packet ==========--");
1052:     xbee_log("Got a packet");
1053:     xbee_log("Packets left: %d",xbee.pktcount);
1054:   }
1055:
1056:   /* unlock the packet mutex */
1057:   xbee_mutex_unlock(xbee.pktmutex);
1058:
1059:   /* and return the packet (must be free'd by caller!) */
1060:   return q;
1061: }
1062:
1063: /* ########################################################################
1064:    xbee_matchpktcon - INTERNAL
1065:    checks if the packet matches the connection */
1066: static int xbee_matchpktcon(xbee_pkt *pkt, xbee_con *con) {
1067:   /* if: the connection type matches the packet type OR
1068:      the connection is 16/64bit remote AT, and the packet is a remote AT response */
1069:   if ((pkt->type == con->type) || /* -- */
1070:       ((pkt->type == xbee_remoteAT) && /* -- */
1071:        ((con->type == xbee_16bitRemoteAT) ||
1072:         (con->type == xbee_64bitRemoteAT)))) {
1073:
1074:     /* if: the packet is modem status OR
1075:        the packet is tx status or AT data and the frame IDs match OR
1076:        the addresses match */
1077:     if (pkt->type == xbee_modemStatus) return 1;
1078:
1079:     if ((pkt->type == xbee_txStatus) ||
1080:         (pkt->type == xbee_localAT) ||
1081:         (pkt->type == xbee_remoteAT)) {
1082:       if (pkt->frameID == con->frameID) {
1083:         return 1;
1084:       }
1085:     } else if (pkt->sAddr64 && !memcmp(pkt->Addr64,con->tAddr,8)) {
1086:       return 1;
1087:     } else if (!pkt->sAddr64 && !memcmp(pkt->Addr16,con->tAddr,2)) {
1088:       return 1;
1089:     }
1090:   }
1091:   return 0;
1092: }
1093:
1094: /* ########################################################################
1095:    xbee_parse_io - INTERNAL
1096:    parses the data given into the packet io information */
1097: static int xbee_parse_io(xbee_pkt *p, unsigned char *d, int maskOffset, int sampleOffset, int sample) {
1098:   xbee_sample *s = &(p->IOdata[sample]);
1099:
1100:   /* copy in the I/O data mask */
1101:   s->IOmask = (((d[maskOffset]<<8) | d[maskOffset + 1]) & 0x7FFF);
1102:
1103:   /* copy in the digital I/O data */
1104:   s->IOdigital = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x01FF);
1105:
```

```
1106:   /* advance over the digital data, if its there */
1107:   sampleOffset += ((s->IOmask & 0x01FF)?2:0);
1108:
1109:   /* copy in the analog I/O data */
1110:   if (s->IOmask & 0x0200) {
1111:     s->IOanalog[0] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1112:     sampleOffset+=2;
1113:   }
1114:   if (s->IOmask & 0x0400) {
1115:     s->IOanalog[1] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1116:     sampleOffset+=2;
1117:   }
1118:   if (s->IOmask & 0x0800) {
1119:     s->IOanalog[2] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1120:     sampleOffset+=2;
1121:   }
1122:   if (s->IOmask & 0x1000) {
1123:     s->IOanalog[3] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1124:     sampleOffset+=2;
1125:   }
1126:   if (s->IOmask & 0x2000) {
1127:     s->IOanalog[4] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1128:     sampleOffset+=2;
1129:   }
1130:   if (s->IOmask & 0x4000) {
1131:     s->IOanalog[5] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1132:     sampleOffset+=2;
1133:   }
1134:
1135:   if (xbee.log) {
1136:     if (s->IOmask & 0x0001)
1137:       xbee_log("Digital 0: %c",((s->IOdigital & 0x0001)?'1':'0'));
1138:     if (s->IOmask & 0x0002)
1139:       xbee_log("Digital 1: %c",((s->IOdigital & 0x0002)?'1':'0'));
1140:     if (s->IOmask & 0x0004)
1141:       xbee_log("Digital 2: %c",((s->IOdigital & 0x0004)?'1':'0'));
1142:     if (s->IOmask & 0x0008)
1143:       xbee_log("Digital 3: %c",((s->IOdigital & 0x0008)?'1':'0'));
1144:     if (s->IOmask & 0x0010)
1145:       xbee_log("Digital 4: %c",((s->IOdigital & 0x0010)?'1':'0'));
1146:     if (s->IOmask & 0x0020)
1147:       xbee_log("Digital 5: %c",((s->IOdigital & 0x0020)?'1':'0'));
1148:     if (s->IOmask & 0x0040)
1149:       xbee_log("Digital 6: %c",((s->IOdigital & 0x0040)?'1':'0'));
1150:     if (s->IOmask & 0x0080)
1151:       xbee_log("Digital 7: %c",((s->IOdigital & 0x0080)?'1':'0'));
1152:     if (s->IOmask & 0x0100)
1153:       xbee_log("Digital 8: %c",((s->IOdigital & 0x0100)?'1':'0'));
1154:     if (s->IOmask & 0x0200)
1155:       xbee_log("Analog  0: %d (~%.2fv)\n",s->IOanalog[0],(3.3/1023)*s->IOanalog[0]);
1156:     if (s->IOmask & 0x0400)
1157:       xbee_log("Analog  1: %d (~%.2fv)\n",s->IOanalog[1],(3.3/1023)*s->IOanalog[1]);
1158:     if (s->IOmask & 0x0800)
1159:       xbee_log("Analog  2: %d (~%.2fv)\n",s->IOanalog[2],(3.3/1023)*s->IOanalog[2]);
1160:     if (s->IOmask & 0x1000)
1161:       xbee_log("Analog  3: %d (~%.2fv)\n",s->IOanalog[3],(3.3/1023)*s->IOanalog[3]);
1162:     if (s->IOmask & 0x2000)
1163:       xbee_log("Analog  4: %d (~%.2fv)\n",s->IOanalog[4],(3.3/1023)*s->IOanalog[4]);
1164:     if (s->IOmask & 0x4000)
1165:       xbee_log("Analog  5: %d (~%.2fv)\n",s->IOanalog[5],(3.3/1023)*s->IOanalog[5]);
1166:   }
1167:
1168:   return sampleOffset;
1169: }
1170:
1171: /* ####################################################################
1172:    xbee_listen_stop
1173:    stops the listen thread after the current packet has been processed */
1174: void xbee_listen_stop(void) {
1175:   xbee.listenrun = 0;
1176: }
1177:
1178: /* ####################################################################
1179:    xbee_listen_wrapper - INTERNAL
1180:    the xbee_listen wrapper. Prints an error when xbee_listen ends */
1181: static void xbee_listen_wrapper(t_info *info) {
1182:   int ret;
1183:   /* just falls out if the proper 'go-ahead' isn't given */
1184:   if (xbee_ready != -1) return;
1185:   /* now allow the parent to continue */
1186:   xbee_ready = -2;
1187:
1188: #ifdef _WIN32 /* ---- */
1189:   /* win32 requires this delay... no idea why */
1190:   usleep(1000000);
```

```
1191: #endif /* ----------- */
1192:
1193:   while (xbee.listenrun) {
1194:     info->i = -1;
1195:     ret = xbee_listen(info);
1196:     if (!xbee.listenrun) break;
1197:     if (xbee.log) {
1198:       xbee_log("xbee_listen() returned [%d]... Restarting in 250ms!",ret);
1199:     }
1200:     usleep(25000);
1201:   }
1202: }
1203:
1204: /* xbee_listen - INTERNAL
1205:    the xbee xbee_listen thread
1206:    reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
1207: static int xbee_listen(t_info *info) {
1208:   unsigned char c, t, d[1024];
1209:   unsigned int l, i, chksum, o;
1210:   struct timeval tv;
1211:   int j;
1212:   xbee_pkt *p, *q;
1213:   xbee_con *con;
1214:   int hasCon;
1215:
1216:   /* just falls out if the proper 'go-ahead' isn't given */
1217:   if (info->i != -1) return -1;
1218:   /* do this forever :) */
1219:   while (xbee.listenrun) {
1220:     /* wait for a valid start byte */
1221:     if (xbee_getrawbyte() != 0x7E) continue;
1222:     if (!xbee.listenrun) return 0;
1223:
1224:     if (xbee.log) {
1225:       xbee_log("--== RX Packet ===========--");
1226:       gettimeofday(&tv,NULL);
1227:       xbee_log("Got a packet @ %ld.%06ld",tv.tv_sec,tv.tv_usec);
1228:     }
1229:
1230:     /* get the length */
1231:     l = xbee_getbyte() << 8;
1232:     l += xbee_getbyte();
1233:
1234:     /* check it is a valid length... */
1235:     if (!l) {
1236:       if (xbee.log) {
1237:         xbee_log("Recived zero length packet!");
1238:       }
1239:       continue;
1240:     }
1241:     if (l > 100) {
1242:       if (xbee.log) {
1243:         xbee_log("Recived oversized packet! Length: %d",l - 1);
1244:       }
1245:     }
1246:     if (l > sizeof(d) - 1) {
1247:       if (xbee.log) {
1248:         xbee_log("Recived packet larger than buffer! Discarding...");
1249:       }
1250:       continue;
1251:     }
1252:
1253:     if (xbee.log) {
1254:       xbee_log("Length: %d",l - 1);
1255:     }
1256:
1257:     /* get the packet type */
1258:     t = xbee_getbyte();
1259:
1260:     /* start the checksum */
1261:     chksum = t;
1262:
1263:     /* suck in all the data */
1264:     for (i = 0; l > 1 && i < 128; l--, i++) {
1265:       /* get an unescaped byte */
1266:       c = xbee_getbyte();
1267:       d[i] = c;
1268:       chksum += c;
1269:       if (xbee.log) {
1270:         xbee_logc("%3d | 0x%02X | ",i,c);
1271:         if ((c > 32) && (c < 127)) fprintf(xbee.log,"'%c'",c); else fprintf(xbee.log," _ ");
1272:
1273:         if ((t == XBEE_64BIT_DATA && i == 10) ||
1274:             (t == XBEE_16BIT_DATA && i == 4) ||
1275:             (t == XBEE_LOCAL_AT   && i == 4) ||
```

```
1276:             (t == XBEE_REMOTE_AT  && i == 14)) {
1277:           /* mark the beginning of the 'data' bytes */
1278:           fprintf(xbee.log,"   <-- data starts");
1279:         }
1280:         xbee_logcf();
1281:       }
1282:     }
1283:     i--; /* it went up too many times!... */
1284:
1285:     /* add the checksum */
1286:     chksum += xbee_getbyte();
1287:
1288:     /* check if the whole packet was recieved, or something else occured... unlikely... */
1289:     if (l>1) {
1290:       if (xbee.log) {
1291:         xbee_log("Didn't get whole packet... :(");
1292:       }
1293:       continue;
1294:     }
1295:
1296:     /* check the checksum */
1297:     if ((chksum & 0xFF) != 0xFF) {
1298:       if (xbee.log) {
1299:         xbee_log("Invalid Checksum: 0x%02X",chksum);
1300:       }
1301:       continue;
1302:     }
1303:
1304:     /* make a new packet */
1305:     p = Xcalloc(sizeof(xbee_pkt));
1306:     q = NULL;
1307:     p->datalen = 0;
1308:
1309:     /* ######################################### */
1310:     /* if: modem status */
1311:     if (t == XBEE_MODEM_STATUS) {
1312:       if (xbee.log) {
1313:         xbee_log("Packet type: Modem Status (0x8A)");
1314:         xbee_logc("Event: ");
1315:         switch (d[0]) {
1316:         case 0x00: fprintf(xbee.log,"Hardware reset"); break;
1317:         case 0x01: fprintf(xbee.log,"Watchdog timer reset"); break;
1318:         case 0x02: fprintf(xbee.log,"Associated"); break;
1319:         case 0x03: fprintf(xbee.log,"Disassociated"); break;
1320:         case 0x04: fprintf(xbee.log,"Synchronization lost"); break;
1321:         case 0x05: fprintf(xbee.log,"Coordinator realignment"); break;
1322:         case 0x06: fprintf(xbee.log,"Coordinator started"); break;
1323:         }
1324:         fprintf(xbee.log,"... (0x%02X)",d[0]);
1325:         xbee_logcf();
1326:       }
1327:       p->type = xbee_modemStatus;
1328:
1329:       p->sAddr64 = FALSE;
1330:       p->dataPkt = FALSE;
1331:       p->txStatusPkt = FALSE;
1332:       p->modemStatusPkt = TRUE;
1333:       p->remoteATPkt = FALSE;
1334:       p->IOPkt = FALSE;
1335:
1336:       /* modem status can only ever give 1 'data' byte */
1337:       p->datalen = 1;
1338:       p->data[0] = d[0];
1339:
1340:       /* ######################################### */
1341:       /* if: local AT response */
1342:     } else if (t == XBEE_LOCAL_AT) {
1343:       if (xbee.log) {
1344:         xbee_log("Packet type: Local AT Response (0x88)");
1345:         xbee_log("FrameID: 0x%02X",d[0]);
1346:         xbee_log("AT Command: %c%c",d[1],d[2]);
1347:         xbee_logc("Status: ");
1348:         if (d[3] == 0) fprintf(xbee.log,"OK");
1349:         else if (d[3] == 1) fprintf(xbee.log,"Error");
1350:         else if (d[3] == 2) fprintf(xbee.log,"Invalid Command");
1351:         else if (d[3] == 3) fprintf(xbee.log,"Invalid Parameter");
1352:         fprintf(xbee.log," (0x%02X)",d[3]);
1353:         xbee_logcf();
1354:       }
1355:       p->type = xbee_localAT;
1356:
1357:       p->sAddr64 = FALSE;
1358:       p->dataPkt = FALSE;
1359:       p->txStatusPkt = FALSE;
1360:       p->modemStatusPkt = FALSE;
```

```
1361:       p->remoteATPkt = FALSE;
1362:       p->IOPkt = FALSE;
1363:
1364:       p->frameID = d[0];
1365:       p->atCmd[0] = d[1];
1366:       p->atCmd[1] = d[2];
1367:
1368:       p->status = d[3];
1369:
1370:       /* copy in the data */
1371:       p->datalen = i-3;
1372:       for (;i>3;i--) p->data[i-4] = d[i];
1373:
1374:       /* ########################################## */
1375:       /* if: remote AT response */
1376:     } else if (t == XBEE_REMOTE_AT) {
1377:       if (xbee.log) {
1378:         xbee_log("Packet type: Remote AT Response (0x97)");
1379:         xbee_log("FrameID: 0x%02X",d[0]);
1380:         xbee_logc("64-bit Address: ");
1381:         for (j=0;j<8;j++) {
1382:           fprintf(xbee.log,(j?":%02X":"%02X"),d[1+j]);
1383:         }
1384:         xbee_logcf();
1385:         xbee_logc("16-bit Address: ");
1386:         for (j=0;j<2;j++) {
1387:           fprintf(xbee.log,(j?":%02X":"%02X"),d[9+j]);
1388:         }
1389:         xbee_logcf();
1390:         xbee_log("AT Command: %c%c",d[11],d[12]);
1391:         xbee_logc("Status: ");
1392:         if (d[13] == 0) fprintf(xbee.log,"OK");
1393:         else if (d[13] == 1) fprintf(xbee.log,"Error");
1394:         else if (d[13] == 2) fprintf(xbee.log,"Invalid Command");
1395:         else if (d[13] == 3) fprintf(xbee.log,"Invalid Parameter");
1396:         else if (d[13] == 4) fprintf(xbee.log,"No Response");
1397:         fprintf(xbee.log," (0x%02X)",d[13]);
1398:         xbee_logcf();
1399:       }
1400:       p->type = xbee_remoteAT;
1401:
1402:       p->sAddr64 = FALSE;
1403:       p->dataPkt = FALSE;
1404:       p->txStatusPkt = FALSE;
1405:       p->modemStatusPkt = FALSE;
1406:       p->remoteATPkt = TRUE;
1407:       p->IOPkt = FALSE;
1408:
1409:       p->frameID = d[0];
1410:
1411:       p->Addr64[0] = d[1];
1412:       p->Addr64[1] = d[2];
1413:       p->Addr64[2] = d[3];
1414:       p->Addr64[3] = d[4];
1415:       p->Addr64[4] = d[5];
1416:       p->Addr64[5] = d[6];
1417:       p->Addr64[6] = d[7];
1418:       p->Addr64[7] = d[8];
1419:
1420:       p->Addr16[0] = d[9];
1421:       p->Addr16[1] = d[10];
1422:
1423:       p->atCmd[0] = d[11];
1424:       p->atCmd[1] = d[12];
1425:
1426:       p->status = d[13];
1427:
1428:       p->samples = 1;
1429:
1430:       if (p->status == 0x00 && p->atCmd[0] == 'I' && p->atCmd[1] == 'S') {
1431:         /* parse the io data */
1432:         if (xbee.log) xbee_log("--- Sample -----------------");
1433:         xbee_parse_io(p, d, 15, 17, 0);
1434:         if (xbee.log) xbee_log("----------------------------");
1435:       } else {
1436:         /* copy in the data */
1437:         p->datalen = i-13;
1438:         for (;i>13;i--) p->data[i-14] = d[i];
1439:       }
1440:
1441:       /* ########################################## */
1442:       /* if: TX status */
1443:     } else if (t == XBEE_TX_STATUS) {
1444:       if (xbee.log) {
1445:         xbee_log("Packet type: TX Status Report (0x89)");
```

```
1446:         xbee_log("FrameID: 0x%02X",d[0]);
1447:         xbee_logc("Status: ");
1448:         if (d[1] == 0) fprintf(xbee.log,"Success");
1449:         else if (d[1] == 1) fprintf(xbee.log,"No ACK");
1450:         else if (d[1] == 2) fprintf(xbee.log,"CCA Failure");
1451:         else if (d[1] == 3) fprintf(xbee.log,"Purged");
1452:         fprintf(xbee.log," (0x%02X)",d[1]);
1453:         xbee_logcf();
1454:       }
1455:       p->type = xbee_txStatus;
1456:
1457:       p->sAddr64 = FALSE;
1458:       p->dataPkt = FALSE;
1459:       p->txStatusPkt = TRUE;
1460:       p->modemStatusPkt = FALSE;
1461:       p->remoteATPkt = FALSE;
1462:       p->IOPkt = FALSE;
1463:
1464:       p->frameID = d[0];
1465:
1466:       p->status = d[1];
1467:
1468:       /* never returns data */
1469:       p->datalen = 0;
1470:
1471:       /* ######################################### */
1472:       /* if: 16 / 64bit data recieve */
1473:     } else if ((t == XBEE_64BIT_DATA) ||
1474:                (t == XBEE_16BIT_DATA)) {
1475:       int offset;
1476:       if (t == XBEE_64BIT_DATA) { /* 64bit */
1477:         offset = 8;
1478:       } else { /* 16bit */
1479:         offset = 2;
1480:       }
1481:       if (xbee.log) {
1482:         xbee_log("Packet type: %d-bit RX Data (0x%02X)",((t == XBEE_64BIT_DATA)?64:16),t);
1483:         xbee_logc("%d-bit Address: ",((t == XBEE_64BIT_DATA)?64:16));
1484:         for (j=0;j<offset;j++) {
1485:           fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
1486:         }
1487:         xbee_logcf();
1488:         xbee_log("RSSI: -%ddB",d[offset]);
1489:         if (d[offset + 1] & 0x02) xbee_log("Options: Address Broadcast");
1490:         if (d[offset + 1] & 0x03) xbee_log("Options: PAN Broadcast");
1491:       }
1492:       p->dataPkt = TRUE;
1493:       p->txStatusPkt = FALSE;
1494:       p->modemStatusPkt = FALSE;
1495:       p->remoteATPkt = FALSE;
1496:       p->IOPkt = FALSE;
1497:
1498:       if (t == XBEE_64BIT_DATA) { /* 64bit */
1499:         p->type = xbee_64bitData;
1500:
1501:         p->sAddr64 = TRUE;
1502:
1503:         p->Addr64[0] = d[0];
1504:         p->Addr64[1] = d[1];
1505:         p->Addr64[2] = d[2];
1506:         p->Addr64[3] = d[3];
1507:         p->Addr64[4] = d[4];
1508:         p->Addr64[5] = d[5];
1509:         p->Addr64[6] = d[6];
1510:         p->Addr64[7] = d[7];
1511:       } else { /* 16bit */
1512:         p->type = xbee_16bitData;
1513:
1514:         p->sAddr64 = FALSE;
1515:
1516:         p->Addr16[0] = d[0];
1517:         p->Addr16[1] = d[1];
1518:       }
1519:
1520:       /* save the RSSI / signal strength
1521:          this can be used with printf as:
1522:          printf("-%ddB\n",p->RSSI); */
1523:       p->RSSI = d[offset];
1524:
1525:       p->status = d[offset + 1];
1526:
1527:       /* copy in the data */
1528:       p->datalen = i-(offset + 1);
1529:       for (;i>offset + 1;i--) p->data[i-(offset + 2)] = d[i];
1530:
```

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

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

```
1701:         q = q->next;
1702:         i++;
1703:       }
1704:       q->next = p;
1705:       xbee.pktcount = i;
1706:     }
1707:     xbee.pktlast = p;
1708:     xbee.pktcount++;
1709:
1710:     /* unlock the packet mutex */
1711:     xbee_mutex_unlock(xbee.pktmutex);
1712:
1713:     if (xbee.log) {
1714:       xbee_log("--========================--");
1715:       xbee_log("Packets: %d",xbee.pktcount);
1716:     }
1717:
1718:     p = q = NULL;
1719:   }
1720:   return 0;
1721: }
1722: static void xbee_callbackWrapper(xbee_con *con) {
1723:   xbee_pkt *pkt;
1724:   t_callback_list *temp;
1725:   /* dont forget! the callback mutex is already locked... by the parent thread :) */
1726:
1727:   xbee_mutex_lock(con->callbackListmutex);
1728:   while (con->callbackList) {
1729:     temp = con->callbackList;
1730:     /* get the packet */
1731:     pkt = temp->pkt;
1732:     /* shift the list along 1 */
1733:     con->callbackList = temp->next;
1734:     Xfree(temp);
1735:     xbee_mutex_unlock(con->callbackListmutex);
1736:
1737:     xbee_log("Starting callback function...");
1738:     con->callback(con,pkt);
1739:     xbee_log("Callback complete!");
1740:     Xfree(pkt);
1741:
1742:     xbee_mutex_lock(con->callbackListmutex);
1743:   }
1744:   xbee_mutex_unlock(con->callbackListmutex);
1745:
1746:   xbee_log("Callback thread ending...");
1747:   /* releasing the thread mutex is the last thing we do! */
1748:   xbee_mutex_unlock(con->callbackmutex);
1749:
1750:   if (con->destroySelf) {
1751:     xbee_endcon2(&con,1);
1752:   }
1753: }
1754:
1755: /* ####################################################################
1756:    xbee_getbyte - INTERNAL
1757:    waits for an escaped byte of data */
1758: static unsigned char xbee_getbyte(void) {
1759:   unsigned char c;
1760:
1761:   ISREADY;
1762:
1763:   /* take a byte */
1764:   c = xbee_getrawbyte();
1765:   /* if its escaped, take another and un-escape */
1766:   if (c == 0x7D) c = xbee_getrawbyte() ^ 0x20;
1767:
1768:   return (c & 0xFF);
1769: }
1770:
1771: /* ####################################################################
1772:    xbee_getrawbyte - INTERNAL
1773:    waits for a raw byte of data */
1774: static unsigned char xbee_getrawbyte(void) {
1775:   int ret;
1776:   unsigned char c = 0x00;
1777:
1778:   ISREADY;
1779:
1780:   /* the loop is just incase there actually isnt a byte there to be read... */
1781:   do {
1782:     /* wait for a read to be possible */
1783:     if ((ret = xbee_select(NULL)) == -1) {
1784:       perror("libxbee:xbee_getrawbyte()");
1785:       exit(1);
```

```
1786:     }
1787:     if (!xbee.listenrun) break;
1788:     if (ret == 0) continue;
1789:
1790:     /* read 1 character */
1791:     xbee_read(&c,1);
1792: #ifdef _WIN32 /* ---- */
1793:     ret = xbee.ttyr;
1794:     if (ret == 0) {
1795:       usleep(10);
1796:       continue;
1797:     }
1798: #endif /* ----------- */
1799:   } while (0);
1800:
1801:   return (c & 0xFF);
1802: }
1803:
1804: /* ####################################################################
1805:    xbee_send_pkt - INTERNAL
1806:    sends a complete packet of data */
1807: static void xbee_send_pkt(t_data *pkt) {
1808:   ISREADY;
1809:
1810:   /* lock the send mutex */
1811:   xbee_mutex_lock(xbee.sendmutex);
1812:
1813:   /* write and flush the data */
1814:   xbee_write(pkt->data,pkt->length);
1815:
1816:   /* unlock the mutex */
1817:   xbee_mutex_unlock(xbee.sendmutex);
1818:
1819:   if (xbee.log) {
1820:     int i,x,y;
1821:     /* prints packet in hex byte-by-byte */
1822:     xbee_logc("TX Packet:");
1823:     for (i=0,x=0,y=0;i<pkt->length;i++,x--) {
1824:       if (x == 0) {
1825:         fprintf(xbee.log,"\n  0x%04X | ",y);
1826:         x = 0x8;
1827:         y += x;
1828:       }
1829:       if (x == 4) {
1830:         fprintf(xbee.log,"  ");
1831:       }
1832:       fprintf(xbee.log,"0x%02X ",pkt->data[i]);
1833:     }
1834:     xbee_logcf();
1835:   }
1836:
1837:   /* free the packet */
1838:   Xfree(pkt);
1839: }
1840:
1841: /* ####################################################################
1842:    xbee_make_pkt - INTERNAL
1843:    adds delimiter field
1844:    calculates length and checksum
1845:    escapes bytes */
1846: static t_data *xbee_make_pkt(unsigned char *data, int length) {
1847:   t_data *pkt;
1848:   unsigned int l, i, o, t, x, m;
1849:   char d = 0;
1850:
1851:   ISREADY;
1852:
1853:   /* check the data given isnt too long
1854:      100 bytes maximum payload + 12 bytes header information */
1855:   if (length > 100 + 12) return NULL;
1856:
1857:   /* calculate the length of the whole packet
1858:      start, length (MSB), length (LSB), DATA, checksum */
1859:   l = 3 + length + 1;
1860:
1861:   /* prepare memory */
1862:   pkt = Xcalloc(sizeof(t_data));
1863:
1864:   /* put start byte on */
1865:   pkt->data[0] = 0x7E;
1866:
1867:   /* copy data into packet */
1868:   for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
1869:     /* if: its time for the checksum */
1870:     if (i == length) d = M8((0xFF - M8(t)));
```

```
1871:     /* if: its time for the high length byte */
1872:     else if (m == 1) d = M8(length >> 8);
1873:     /* if: its time for the low length byte */
1874:     else if (m == 2) d = M8(length);
1875:     /* if: its time for the normal data */
1876:     else if (m > 2) d = data[i];
1877:
1878:     x = 0;
1879:     /* check for any escapes needed */
1880:     if ((d == 0x11) || /* XON */
1881:         (d == 0x13) || /* XOFF */
1882:         (d == 0x7D) || /* Escape */
1883:         (d == 0x7E)) { /* Frame Delimiter */
1884:       l++;
1885:       pkt->data[o++] = 0x7D;
1886:       x = 1;
1887:     }
1888:
1889:     /* move data in */
1890:     pkt->data[o] = ((!x)?d:d^0x20);
1891:     if (m > 2) {
1892:       i++;
1893:       t += d;
1894:     }
1895:   }
1896:
1897:   /* remember the length */
1898:   pkt->length = l;
1899:
1900:   return pkt;
1901: }
```