```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 403 2010-08-02 09:15:39Z 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 *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;
```

```
 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(const char *logformat, int unlock, const char *file,
138:                       const int line, const char *function, char *format, ...) {
139:   char buf[128];
140:   va_list ap;
141:   FILE *log;
142:   va_start(ap,format);
143:   vsnprintf(buf,127,format,ap);
144:   va_end(ap);
145:   if (xbee.log) {
146:     log = xbee.log;
147:   } else {
148:     log = stderr;
149:   }
150:   xbee_mutex_lock(xbee.logmutex);
151:   fprintf(log,logformat,file,line,function,buf);
152:   if (unlock) xbee_mutex_unlock(xbee.logmutex);
153: }
154:
155: /* ###################################################################
156:    xbee_sendAT - INTERNAL
157:    allows for an at command to be send, and the reply to be captured */
158: static int xbee_sendAT(char *command, char *retBuf, int retBuflen) {
159:   return xbee_sendATdelay(0,command,retBuf, retBuflen);
160: }
161: static int xbee_sendATdelay(int guartTime, char *command, char *retBuf, int retBuflen) {
162:   struct timeval to;
163:
164:   int ret;
165:   int bufi = 0;
166:
167:   /* if there is a guartTime given, then use it and a bit more */
168:   if (guartTime) usleep(guartTime * 1200);
169:
170:   /* get rid of any pre-command sludge... */
```

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

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

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

```c

  if (xbee.log) xbee_log("libxbee: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
  if (xbee.log) xbee_log("libxbee: Starting...");
  if (xbee.log) xbee_log("libxbee: SVN Info: %s",xbee_svn_version());
  if (xbee.log) xbee_log("libxbee: Build Info: %s",xbee_build_info());
  if (xbee.log) xbee_log("libxbee: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

  /* setup the connection stuff */
  xbee.conlist = NULL;

  /* setup the packet stuff */
  xbee.pktlist = NULL;
  xbee.pktlast = NULL;
  xbee.pktcount = 0;
  xbee.listenrun = 1;

  /* setup the mutexes */
  if (xbee_mutex_init(xbee.conmutex)) {
    perror("xbee_setup():xbee_mutex_init(conmutex)");
    if (xbee.log) fclose(xbee.log);
    return -1;
  }
  if (xbee_mutex_init(xbee.pktmutex)) {
    perror("xbee_setup():xbee_mutex_init(pktmutex)");
    if (xbee.log) fclose(xbee.log);
    xbee_mutex_destroy(xbee.conmutex);
    return -1;
  }
  if (xbee_mutex_init(xbee.sendmutex)) {
    perror("xbee_setup():xbee_mutex_init(sendmutex)");
    if (xbee.log) fclose(xbee.log);
    xbee_mutex_destroy(xbee.conmutex);
    xbee_mutex_destroy(xbee.pktmutex);
    return -1;
  }

  /* take a copy of the XBee device path */
  if ((xbee.path = Xmalloc(sizeof(char) * (strlen(path) + 1))) == NULL) {
    perror("xbee_setup():Xmalloc(path)");
    if (xbee.log) fclose(xbee.log);
    xbee_mutex_destroy(xbee.conmutex);
    xbee_mutex_destroy(xbee.pktmutex);
    xbee_mutex_destroy(xbee.sendmutex);
    return -1;
  }
  strcpy(xbee.path,path);
  if (xbee.log) xbee_log("Opening serial port '%s'...",xbee.path);

  /* call the relevant init function */
  if ((ret = init_serial(baudrate)) != 0) {
    xbee_log("Something failed while opening the serial port...");
    if (xbee.log) fclose(xbee.log);
    xbee_mutex_destroy(xbee.conmutex);
    xbee_mutex_destroy(xbee.pktmutex);
    xbee_mutex_destroy(xbee.sendmutex);
    Xfree(xbee.path);
    return ret;
  }

  /* when xbee_end() is called, if this is not 2 then ATAP will be set to this value */
  xbee.oldAPI = 2;
  xbee.cmdSeq = cmdSeq;
  xbee.cmdTime = cmdTime;
  if (xbee.cmdSeq && xbee.cmdTime) {
    if (xbee_startAPI()) {
      if (xbee.log) {
        xbee_log("Couldn't communicate with XBee...");
        fclose(xbee.log);
      }
      xbee_mutex_destroy(xbee.conmutex);
      xbee_mutex_destroy(xbee.pktmutex);
      xbee_mutex_destroy(xbee.sendmutex);
      Xfree(xbee.path);
#ifdef __GNUC__ /* ---- */
      close(xbee.ttyfd);
#endif /* ------------- */
      xbee_close(xbee.tty);
      return -1;
    }
  }

  /* allow the listen thread to start */
  xbee_ready = -1;

  /* can start xbee_listen thread now */
```

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

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

```
681:     case xbee_64bitIO:
682:       xbee_logc("New %d-bit IO connection! (to: ",(con->tAddr64?64:16));
683:       for (i=0;i<(con->tAddr64?8:2);i++) {
684:         fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
685:       }
686:       fprintf(xbee.log,")");
687:       xbee_logcf();
688:       break;
689:     case xbee_txStatus:
690:       xbee_log("New Tx status connection!");
691:       break;
692:     case xbee_modemStatus:
693:       xbee_log("New modem status connection!");
694:       break;
695:     case xbee_unknown:
696:     default:
697:       xbee_log("New unknown connection!");
698:     }
699:   }
700:
701:   /* make it the last in the list */
702:   con->next = NULL;
703:   /* add it to the list */
704:   if (xbee.conlist) {
705:     ocon->next = con;
706:   } else {
707:     xbee.conlist = con;
708:   }
709:
710:   /* unlock the mutex */
711:   xbee_mutex_unlock(xbee.conmutex);
712:   return con;
713: }
714:
715: /* ########################################################################
716:    xbee_conflush
717:    removes any packets that have been collected for the specified
718:    connection */
719: void xbee_flushcon(xbee_con *con) {
720:   xbee_pkt *r, *p, *n;
721:
722:   /* lock the packet mutex */
723:   xbee_mutex_lock(xbee.pktmutex);
724:
725:   /* if: there are packets */
726:   if ((p = xbee.pktlist) != NULL) {
727:     r = NULL;
728:     /* get all packets for this connection */
729:     do {
730:       /* does the packet match the connection? */
731:       if (xbee_matchpktcon(p,con)) {
732:         /* if it was the first packet */
733:         if (!r) {
734:           /* move the chain along */
735:           xbee.pktlist = p->next;
736:         } else {
737:           /* otherwise relink the list */
738:           r->next = p->next;
739:         }
740:         xbee.pktcount--;
741:
742:         /* free this packet! */
743:         n = p->next;
744:         Xfree(p);
745:         /* move on */
746:         p = n;
747:       } else {
748:         /* move on */
749:         r = p;
750:         p = p->next;
751:       }
752:     } while (p);
753:     xbee.pktlast = r;
754:   }
755:
756:   /* unlock the packet mutex */
757:   xbee_mutex_unlock(xbee.pktmutex);
758: }
759:
760: /* ########################################################################
761:    xbee_endcon
762:    close the unwanted connection
763:    free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
764: void xbee_endcon2(xbee_con **con, int skipUnlink) {
765:   xbee_con *t, *u;
```

```
766:
767:   if (!skipUnlink) {
768:     /* lock the connection mutex */
769:     xbee_mutex_lock(xbee.conmutex);
770:
771:     u = t = xbee.conlist;
772:     while (t && t != *con) {
773:       u = t;
774:       t = t->next;
775:     }
776:     if (!t) {
777:       /* invalid connection given... */
778:       if (xbee.log) {
779:         xbee_log("Attempted to close invalid connection...");
780:       }
781:       /* unlock the connection mutex */
782:       xbee_mutex_unlock(xbee.conmutex);
783:       return;
784:     }
785:     /* extract this connection from the list */
786:     u->next = t->next;
787:
788:     /* unlock the connection mutex */
789:     xbee_mutex_unlock(xbee.conmutex);
790:   }
791:
792:   /* check if a callback thread is running... */
793:   if (t->callback && xbee_mutex_trylock(t->callbackmutex)) {
794:     /* if it is running... tell it to destroy the connection on completion */
795:     xbee_log("Attempted to close a connection with active callbacks... "
796:              "Connection will be destroied when callbacks have completeted...");
797:     t->destroySelf = 1;
798:     return;
799:   }
800:
801:   /* remove all packets for this connection */
802:   xbee_flushcon(t);
803:
804:   /* destroy the callback mutex */
805:   xbee_mutex_destroy(t->callbackmutex);
806:   xbee_mutex_destroy(t->callbackListmutex);
807:   xbee_mutex_destroy(t->Txmutex);
808:   xbee_sem_destroy(t->waitforACKsem);
809:
810:   /* free the connection! */
811:   Xfree(*con);
812: }
813:
814: /* ####################################################################
815:    xbee_senddata
816:    send the specified data to the provided connection */
817: int xbee_senddata(xbee_con *con, char *format, ...) {
818:   int ret;
819:   va_list ap;
820:
821:   ISREADY;
822:
823:   /* xbee_vsenddata() wants a va_list... */
824:   va_start(ap, format);
825:   /* hand it over :) */
826:   ret = xbee_vsenddata(con,format,ap);
827:   va_end(ap);
828:   return ret;
829: }
830:
831: int xbee_vsenddata(xbee_con *con, char *format, va_list ap) {
832:   unsigned char data[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
833:   int length;
834:
835:   ISREADY;
836:
837:   /* make up the data and keep the length, its possible there are nulls in there */
838:   length = vsnprintf((char *)data,128,format,ap);
839:
840:   /* hand it over :) */
841:   return xbee_nsenddata(con,(char *)data,length);
842: }
843:
844: /* returns:
845:     1 - if NAC was recieved
846:     0 - if packet was successfully sent (or just sent if waitforACK is off)
847:    -1 - if there was an error building the packet
848:    -2 - if the connection type was unknown */
849: int xbee_nsenddata(xbee_con *con, char *data, int length) {
850:   t_data *pkt;
```

```
851:   int i;
852:   unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
853:
854:   ISREADY;
855:
856:   if (!con) return -1;
857:   if (con->type == xbee_unknown) return -1;
858:   if (length > 127) return -1;
859:
860:   if (xbee.log) {
861:     xbee_log("--== TX Packet ============--");
862:     xbee_logc("Connection Type: ");
863:     switch (con->type) {
864:     case xbee_unknown:       fprintf(xbee.log,"Unknown"); break;
865:     case xbee_localAT:       fprintf(xbee.log,"Local AT"); break;
866:     case xbee_remoteAT:      fprintf(xbee.log,"Remote AT"); break;
867:     case xbee_16bitRemoteAT: fprintf(xbee.log,"Remote AT (16-bit)"); break;
868:     case xbee_64bitRemoteAT: fprintf(xbee.log,"Remote AT (64-bit)"); break;
869:     case xbee_16bitData:     fprintf(xbee.log,"Data (16-bit)"); break;
870:     case xbee_64bitData:     fprintf(xbee.log,"Data (64-bit)"); break;
871:     case xbee_16bitIO:       fprintf(xbee.log,"IO (16-bit)"); break;
872:     case xbee_64bitIO:       fprintf(xbee.log,"IO (64-bit)"); break;
873:     case xbee_txStatus:      fprintf(xbee.log,"Tx Status"); break;
874:     case xbee_modemStatus:   fprintf(xbee.log,"Modem Status"); break;
875:     }
876:     xbee_logcf();
877:     xbee_logc("Destination: ");
878:     for (i=0;i<(con->tAddr64?8:2);i++) {
879:       fprintf(xbee.log,(i?":%02X":"%02X"),con->tAddr[i]);
880:     }
881:     xbee_logcf();
882:     xbee_log("Length: %d",length);
883:     for (i=0;i<length;i++) {
884:       xbee_logc("%3d | 0x%02X ",i,(unsigned char)data[i]);
885:       if ((data[i] > 32) && (data[i] < 127)) {
886:         fprintf(xbee.log,"'%c'",data[i]);
887:       } else{
888:         fprintf(xbee.log," _");
889:       }
890:       xbee_logcf();
891:     }
892:   }
893:
894:   /* ########################################## */
895:   /* if: local AT */
896:   if (con->type == xbee_localAT) {
897:     /* AT commands are 2 chars long (plus optional parameter) */
898:     if (length < 2) return -1;
899:
900:     /* use the command? */
901:     buf[0] = ((!con->atQueue)?XBEE_LOCAL_ATREQ:XBEE_LOCAL_ATQUE);
902:     buf[1] = con->frameID;
903:
904:     /* copy in the data */
905:     for (i=0;i<length;i++) {
906:       buf[i+2] = data[i];
907:     }
908:
909:     /* setup the packet */
910:     pkt = xbee_make_pkt(buf,i+2);
911:     /* send it on */
912:     return xbee_send_pkt(pkt,con);
913:
914:     /* ########################################## */
915:     /* if: remote AT */
916:   } else if ((con->type == xbee_16bitRemoteAT) ||
917:              (con->type == xbee_64bitRemoteAT)) {
918:     if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter) */
919:     buf[0] = XBEE_REMOTE_ATREQ;
920:     buf[1] = con->frameID;
921:
922:     /* copy in the relevant address */
923:     if (con->tAddr64) {
924:       memcpy(&buf[2],con->tAddr,8);
925:       buf[10] = 0xFF;
926:       buf[11] = 0xFE;
927:     } else {
928:       memset(&buf[2],0,8);
929:       memcpy(&buf[10],con->tAddr,2);
930:     }
931:     /* queue the command? */
932:     buf[12] = ((!con->atQueue)?0x02:0x00);
933:
934:     /* copy in the data */
935:     for (i=0;i<length;i++) {
```

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

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

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

```
1191: }
1192:
1193: /* ###################################################################
1194:    xbee_listen_wrapper - INTERNAL
1195:    the xbee_listen wrapper. Prints an error when xbee_listen ends */
1196: static void xbee_listen_wrapper(t_info *info) {
1197:   int ret;
1198:   /* just falls out if the proper 'go-ahead' isn't given */
1199:   if (xbee_ready != -1) return;
1200:   /* now allow the parent to continue */
1201:   xbee_ready = -2;
1202:
1203: #ifdef _WIN32 /* ---- */
1204:   /* win32 requires this delay... no idea why */
1205:   usleep(1000000);
1206: #endif /* ----------- */
1207:
1208:   while (xbee.listenrun) {
1209:     info->i = -1;
1210:     ret = xbee_listen(info);
1211:     if (!xbee.listenrun) break;
1212:     if (xbee.log) {
1213:       xbee_log("xbee_listen() returned [%d]... Restarting in 250ms!",ret);
1214:     }
1215:     usleep(25000);
1216:   }
1217: }
1218:
1219: /* xbee_listen - INTERNAL
1220:    the xbee xbee_listen thread
1221:    reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
1222: static int xbee_listen(t_info *info) {
1223:   unsigned char c, t, d[1024];
1224:   unsigned int l, i, chksum, o;
1225:   struct timeval tv;
1226:   int j;
1227:   xbee_pkt *p, *q;
1228:   xbee_con *con;
1229:   int hasCon;
1230:
1231:   /* just falls out if the proper 'go-ahead' isn't given */
1232:   if (info->i != -1) return -1;
1233:   /* do this forever :) */
1234:   while (xbee.listenrun) {
1235:     /* wait for a valid start byte */
1236:     if ((c = xbee_getrawbyte()) != 0x7E) {
1237:       if (xbee.log) xbee_log("***** Unexpected byte (0x%02X)... *****",c);
1238:       continue;
1239:     }
1240:     if (!xbee.listenrun) return 0;
1241:
1242:     if (xbee.log) {
1243:       xbee_log("--== RX Packet ===========--");
1244:       gettimeofday(&tv,NULL);
1245:       xbee_log("Got a packet @ %ld.%06ld",tv.tv_sec,tv.tv_usec);
1246:     }
1247:
1248:     /* get the length */
1249:     l = xbee_getbyte() << 8;
1250:     l += xbee_getbyte();
1251:
1252:     /* check it is a valid length... */
1253:     if (!l) {
1254:       if (xbee.log) {
1255:         xbee_log("Recived zero length packet!");
1256:       }
1257:       continue;
1258:     }
1259:     if (l > 100) {
1260:       if (xbee.log) {
1261:         xbee_log("Recived oversized packet! Length: %d",l - 1);
1262:       }
1263:     }
1264:     if (l > sizeof(d) - 1) {
1265:       if (xbee.log) {
1266:         xbee_log("Recived packet larger than buffer! Discarding...");
1267:       }
1268:       continue;
1269:     }
1270:
1271:     if (xbee.log) {
1272:       xbee_log("Length: %d",l - 1);
1273:     }
1274:
1275:     /* get the packet type */
```

```
1276:     t = xbee_getbyte();
1277:
1278:     /* start the checksum */
1279:     chksum = t;
1280:
1281:     /* suck in all the data */
1282:     for (i = 0; l > 1 && i < 128; l--, i++) {
1283:       /* get an unescaped byte */
1284:       c = xbee_getbyte();
1285:       d[i] = c;
1286:       chksum += c;
1287:       if (xbee.log) {
1288:         xbee_logc("%3d | 0x%02X | ",i,c);
1289:         if ((c > 32) && (c < 127)) fprintf(xbee.log,"'%c'",c); else fprintf(xbee.log," _ ");
1290:
1291:         if ((t == XBEE_64BIT_DATA && i == 10) ||
1292:             (t == XBEE_16BIT_DATA && i == 4) ||
1293:             (t == XBEE_64BIT_IO   && i == 13) ||
1294:             (t == XBEE_16BIT_IO   && i == 7) ||
1295:             (t == XBEE_LOCAL_AT   && i == 4) ||
1296:             (t == XBEE_REMOTE_AT  && i == 14)) {
1297:           /* mark the beginning of the 'data' bytes */
1298:           fprintf(xbee.log,"   <-- data starts");
1299:         } else if (t == XBEE_64BIT_IO) {
1300:           if (i == 10) fprintf(xbee.log,"   <-- sample count");
1301:           else if (i == 11) fprintf(xbee.log,"   <-- mask (msb)");
1302:           else if (i == 12) fprintf(xbee.log,"   <-- mask (lsb)");
1303:         } else if (t == XBEE_16BIT_IO) {
1304:           if (i == 4) fprintf(xbee.log,"   <-- sample count");
1305:           else if (i == 5) fprintf(xbee.log,"   <-- mask (msb)");
1306:           else if (i == 6) fprintf(xbee.log,"   <-- mask (lsb)");
1307:         }
1308:         xbee_logcf();
1309:       }
1310:     }
1311:     i--; /* it went up too many times!... */
1312:
1313:     /* add the checksum */
1314:     chksum += xbee_getbyte();
1315:
1316:     /* check if the whole packet was recieved, or something else occured... unlikely... */
1317:     if (l>1) {
1318:       if (xbee.log) {
1319:         xbee_log("Didn't get whole packet... :(");
1320:       }
1321:       continue;
1322:     }
1323:
1324:     /* check the checksum */
1325:     if ((chksum & 0xFF) != 0xFF) {
1326:       if (xbee.log) {
1327:         chksum &= 0xFF;
1328:         xbee_log("Invalid Checksum: 0x%02X",chksum);
1329:       }
1330:       continue;
1331:     }
1332:
1333:     /* make a new packet */
1334:     p = Xcalloc(sizeof(xbee_pkt));
1335:     q = NULL;
1336:     p->datalen = 0;
1337:
1338:     /* ######################################### */
1339:     /* if: modem status */
1340:     if (t == XBEE_MODEM_STATUS) {
1341:       if (xbee.log) {
1342:         xbee_log("Packet type: Modem Status (0x8A)");
1343:         xbee_logc("Event: ");
1344:         switch (d[0]) {
1345:         case 0x00: fprintf(xbee.log,"Hardware reset"); break;
1346:         case 0x01: fprintf(xbee.log,"Watchdog timer reset"); break;
1347:         case 0x02: fprintf(xbee.log,"Associated"); break;
1348:         case 0x03: fprintf(xbee.log,"Disassociated"); break;
1349:         case 0x04: fprintf(xbee.log,"Synchronization lost"); break;
1350:         case 0x05: fprintf(xbee.log,"Coordinator realignment"); break;
1351:         case 0x06: fprintf(xbee.log,"Coordinator started"); break;
1352:         }
1353:         fprintf(xbee.log,"... (0x%02X)",d[0]);
1354:         xbee_logcf();
1355:       }
1356:       p->type = xbee_modemStatus;
1357:
1358:       p->sAddr64 = FALSE;
1359:       p->dataPkt = FALSE;
1360:       p->txStatusPkt = FALSE;
```

```
1361:       p->modemStatusPkt = TRUE;
1362:       p->remoteATPkt = FALSE;
1363:       p->IOPkt = FALSE;
1364:
1365:       /* modem status can only ever give 1 'data' byte */
1366:       p->datalen = 1;
1367:       p->data[0] = d[0];
1368:
1369:       /* ######################################### */
1370:       /* if: local AT response */
1371:     } else if (t == XBEE_LOCAL_AT) {
1372:       if (xbee.log) {
1373:         xbee_log("Packet type: Local AT Response (0x88)");
1374:         xbee_log("FrameID: 0x%02X",d[0]);
1375:         xbee_log("AT Command: %c%c",d[1],d[2]);
1376:         xbee_logc("Status: ");
1377:         if (d[3] == 0) fprintf(xbee.log,"OK");
1378:         else if (d[3] == 1) fprintf(xbee.log,"Error");
1379:         else if (d[3] == 2) fprintf(xbee.log,"Invalid Command");
1380:         else if (d[3] == 3) fprintf(xbee.log,"Invalid Parameter");
1381:         fprintf(xbee.log," (0x%02X)",d[3]);
1382:         xbee_logcf();
1383:       }
1384:       p->type = xbee_localAT;
1385:
1386:       p->sAddr64 = FALSE;
1387:       p->dataPkt = FALSE;
1388:       p->txStatusPkt = FALSE;
1389:       p->modemStatusPkt = FALSE;
1390:       p->remoteATPkt = FALSE;
1391:       p->IOPkt = FALSE;
1392:
1393:       p->frameID = d[0];
1394:       p->atCmd[0] = d[1];
1395:       p->atCmd[1] = d[2];
1396:
1397:       p->status = d[3];
1398:
1399:       /* copy in the data */
1400:       p->datalen = i-3;
1401:       for (;i>3;i--) p->data[i-4] = d[i];
1402:
1403:       /* ######################################### */
1404:       /* if: remote AT response */
1405:     } else if (t == XBEE_REMOTE_AT) {
1406:       if (xbee.log) {
1407:         xbee_log("Packet type: Remote AT Response (0x97)");
1408:         xbee_log("FrameID: 0x%02X",d[0]);
1409:         xbee_logc("64-bit Address: ");
1410:         for (j=0;j<8;j++) {
1411:           fprintf(xbee.log,(j?":%02X":"%02X"),d[1+j]);
1412:         }
1413:         xbee_logcf();
1414:         xbee_logc("16-bit Address: ");
1415:         for (j=0;j<2;j++) {
1416:           fprintf(xbee.log,(j?":%02X":"%02X"),d[9+j]);
1417:         }
1418:         xbee_logcf();
1419:         xbee_log("AT Command: %c%c",d[11],d[12]);
1420:         xbee_logc("Status: ");
1421:         if (d[13] == 0) fprintf(xbee.log,"OK");
1422:         else if (d[13] == 1) fprintf(xbee.log,"Error");
1423:         else if (d[13] == 2) fprintf(xbee.log,"Invalid Command");
1424:         else if (d[13] == 3) fprintf(xbee.log,"Invalid Parameter");
1425:         else if (d[13] == 4) fprintf(xbee.log,"No Response");
1426:         fprintf(xbee.log," (0x%02X)",d[13]);
1427:         xbee_logcf();
1428:       }
1429:       p->type = xbee_remoteAT;
1430:
1431:       p->sAddr64 = FALSE;
1432:       p->dataPkt = FALSE;
1433:       p->txStatusPkt = FALSE;
1434:       p->modemStatusPkt = FALSE;
1435:       p->remoteATPkt = TRUE;
1436:       p->IOPkt = FALSE;
1437:
1438:       p->frameID = d[0];
1439:
1440:       p->Addr64[0] = d[1];
1441:       p->Addr64[1] = d[2];
1442:       p->Addr64[2] = d[3];
1443:       p->Addr64[3] = d[4];
1444:       p->Addr64[4] = d[5];
1445:       p->Addr64[5] = d[6];
```

```
1446:       p->Addr64[6] = d[7];
1447:       p->Addr64[7] = d[8];
1448:
1449:       p->Addr16[0] = d[9];
1450:       p->Addr16[1] = d[10];
1451:
1452:       p->atCmd[0] = d[11];
1453:       p->atCmd[1] = d[12];
1454:
1455:       p->status = d[13];
1456:
1457:       p->samples = 1;
1458:
1459:       if (p->status == 0x00 && p->atCmd[0] == 'I' && p->atCmd[1] == 'S') {
1460:         /* parse the io data */
1461:         if (xbee.log) xbee_log("--- Sample -----------------");
1462:         xbee_parse_io(p, d, 15, 17, 0);
1463:         if (xbee.log) xbee_log("----------------------------");
1464:       } else {
1465:         /* copy in the data */
1466:         p->datalen = i-13;
1467:         for (;i>13;i--) p->data[i-14] = d[i];
1468:       }
1469:
1470:       /* ######################################### */
1471:       /* if: TX status */
1472:     } else if (t == XBEE_TX_STATUS) {
1473:       if (xbee.log) {
1474:         xbee_log("Packet type: TX Status Report (0x89)");
1475:         xbee_log("FrameID: 0x%02X",d[0]);
1476:         xbee_logc("Status: ");
1477:         if (d[1] == 0) fprintf(xbee.log,"Success");
1478:         else if (d[1] == 1) fprintf(xbee.log,"No ACK");
1479:         else if (d[1] == 2) fprintf(xbee.log,"CCA Failure");
1480:         else if (d[1] == 3) fprintf(xbee.log,"Purged");
1481:         fprintf(xbee.log," (0x%02X)",d[1]);
1482:         xbee_logcf();
1483:       }
1484:       p->type = xbee_txStatus;
1485:
1486:       p->sAddr64 = FALSE;
1487:       p->dataPkt = FALSE;
1488:       p->txStatusPkt = TRUE;
1489:       p->modemStatusPkt = FALSE;
1490:       p->remoteATPkt = FALSE;
1491:       p->IOPkt = FALSE;
1492:
1493:       p->frameID = d[0];
1494:
1495:       p->status = d[1];
1496:
1497:       /* never returns data */
1498:       p->datalen = 0;
1499:
1500:       /* check for any connections waiting for a status update */
1501:       /* lock the connection mutex */
1502:       xbee_mutex_lock(xbee.conmutex);
1503:
1504:       con = xbee.conlist;
1505:       while (con) {
1506:         if ((con->frameID == p->frameID) &&
1507:             (con->ACKstatus == 1)) {
1508:           con->ACKstatus = ((p->status == 0)?2:3);
1509:           xbee_sem_post(con->waitforACKsem);
1510:         }
1511:         con = con->next;
1512:       }
1513:
1514:       /* unlock the connection mutex */
1515:       xbee_mutex_unlock(xbee.conmutex);
1516:
1517:       /* ######################################### */
1518:       /* if: 16 / 64bit data recieve */
1519:     } else if ((t == XBEE_64BIT_DATA) ||
1520:                (t == XBEE_16BIT_DATA)) {
1521:       int offset;
1522:       if (t == XBEE_64BIT_DATA) { /* 64bit */
1523:         offset = 8;
1524:       } else { /* 16bit */
1525:         offset = 2;
1526:       }
1527:       if (xbee.log) {
1528:         xbee_log("Packet type: %d-bit RX Data (0x%02X)",((t == XBEE_64BIT_DATA)?64:16),t);
1529:         xbee_logc("%d-bit Address: ",((t == XBEE_64BIT_DATA)?64:16));
1530:         for (j=0;j<offset;j++) {
```

```
1531:           fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
1532:         }
1533:         xbee_logcf();
1534:         xbee_log("RSSI: -%ddB",d[offset]);
1535:         if (d[offset + 1] & 0x02) xbee_log("Options: Address Broadcast");
1536:         if (d[offset + 1] & 0x03) xbee_log("Options: PAN Broadcast");
1537:       }
1538:       p->dataPkt = TRUE;
1539:       p->txStatusPkt = FALSE;
1540:       p->modemStatusPkt = FALSE;
1541:       p->remoteATPkt = FALSE;
1542:       p->IOPkt = FALSE;
1543:
1544:       if (t == XBEE_64BIT_DATA) { /* 64bit */
1545:         p->type = xbee_64bitData;
1546:
1547:         p->sAddr64 = TRUE;
1548:
1549:         p->Addr64[0] = d[0];
1550:         p->Addr64[1] = d[1];
1551:         p->Addr64[2] = d[2];
1552:         p->Addr64[3] = d[3];
1553:         p->Addr64[4] = d[4];
1554:         p->Addr64[5] = d[5];
1555:         p->Addr64[6] = d[6];
1556:         p->Addr64[7] = d[7];
1557:       } else { /* 16bit */
1558:         p->type = xbee_16bitData;
1559:
1560:         p->sAddr64 = FALSE;
1561:
1562:         p->Addr16[0] = d[0];
1563:         p->Addr16[1] = d[1];
1564:       }
1565:
1566:       /* save the RSSI / signal strength
1567:          this can be used with printf as:
1568:          printf("-%ddB\n",p->RSSI); */
1569:       p->RSSI = d[offset];
1570:
1571:       p->status = d[offset + 1];
1572:
1573:       /* copy in the data */
1574:       p->datalen = i-(offset + 1);
1575:       for (;i>offset + 1;i--) p->data[i-(offset + 2)] = d[i];
1576:
1577:       /* ############################################ */
1578:       /* if: 16 / 64bit I/O recieve */
1579:     } else if ((t == XBEE_64BIT_IO) ||
1580:                (t == XBEE_16BIT_IO)) {
1581:       int offset,i2;
1582:       if (t == XBEE_64BIT_IO) { /* 64bit */
1583:         p->type = xbee_64bitIO;
1584:
1585:         p->sAddr64 = TRUE;
1586:
1587:         p->Addr64[0] = d[0];
1588:         p->Addr64[1] = d[1];
1589:         p->Addr64[2] = d[2];
1590:         p->Addr64[3] = d[3];
1591:         p->Addr64[4] = d[4];
1592:         p->Addr64[5] = d[5];
1593:         p->Addr64[6] = d[6];
1594:         p->Addr64[7] = d[7];
1595:
1596:         offset = 8;
1597:         p->samples = d[10];
1598:       } else { /* 16bit */
1599:         p->type = xbee_16bitIO;
1600:
1601:         p->sAddr64 = FALSE;
1602:
1603:         p->Addr16[0] = d[0];
1604:         p->Addr16[1] = d[1];
1605:
1606:         offset = 2;
1607:         p->samples = d[4];
1608:       }
1609:       if (p->samples > 1) {
1610:         p = Xrealloc(p, sizeof(xbee_pkt) + (sizeof(xbee_sample) * (p->samples - 1)));
1611:       }
1612:       if (xbee.log) {
1613:         xbee_log("Packet type: %d-bit RX I/O Data (0x%02X)",((t == XBEE_64BIT_IO)?64:16),t);
1614:         xbee_logc("%d-bit Address: ",((t == XBEE_64BIT_IO)?64:16));
1615:         for (j = 0; j < offset; j++) {
```

```
1616:           fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
1617:         }
1618:         xbee_logcf();
1619:         xbee_log("RSSI: -%ddB",d[offset]);
1620:         if (d[9] & 0x02) xbee_log("Options: Address Broadcast");
1621:         if (d[9] & 0x02) xbee_log("Options: PAN Broadcast");
1622:         xbee_log("Samples: %d",d[offset + 2]);
1623:       }
1624:       i2 = offset + 5;
1625:
1626:       /* never returns data */
1627:       p->datalen = 0;
1628:
1629:       p->dataPkt = FALSE;
1630:       p->txStatusPkt = FALSE;
1631:       p->modemStatusPkt = FALSE;
1632:       p->remoteATPkt = FALSE;
1633:       p->IOPkt = TRUE;
1634:
1635:       /* save the RSSI / signal strength
1636:          this can be used with printf as:
1637:          printf("-%ddB\n",p->RSSI); */
1638:       p->RSSI = d[offset];
1639:
1640:       p->status = d[offset + 1];
1641:
1642:       /* each sample is split into its own packet here, for simplicity */
1643:       for (o = 0; o < p->samples; o++) {
1644:         if (i2 >= i) {
1645:           if (xbee.log) xbee_log("Invalid I/O data! Actually contained %d samples...",o);
1646:           p = Xrealloc(p, sizeof(xbee_pkt) + (sizeof(xbee_sample) * ((o>1)?o:1)));
1647:           p->samples = o;
1648:           break;
1649:         }
1650:         if (xbee.log) {
1651:           xbee_log("--- Sample %3d -------------", o);
1652:         }
1653:
1654:         /* parse the io data */
1655:         i2 = xbee_parse_io(p, d, offset + 3, i2, o);
1656:       }
1657:       if (xbee.log) {
1658:         xbee_log("----------------------------");
1659:       }
1660:
1661:       /* ############################################ */
1662:       /* if: Unknown */
1663:     } else {
1664:       if (xbee.log) {
1665:         xbee_log("Packet type: Unknown (0x%02X)",t);
1666:       }
1667:       p->type = xbee_unknown;
1668:     }
1669:     p->next = NULL;
1670:
1671:     /* lock the connection mutex */
1672:     xbee_mutex_lock(xbee.conmutex);
1673:
1674:     con = xbee.conlist;
1675:     hasCon = 0;
1676:     while (con) {
1677:       if (xbee_matchpktcon(p,con)) {
1678:         hasCon = 1;
1679:         break;
1680:       }
1681:       con = con->next;
1682:     }
1683:
1684:     /* unlock the connection mutex */
1685:     xbee_mutex_unlock(xbee.conmutex);
1686:
1687:     /* if the packet doesn't have a connection, don't add it! */
1688:     if (!hasCon) {
1689:       Xfree(p);
1690:       if (xbee.log) {
1691:         xbee_log("Connectionless packet... discarding!");
1692:       }
1693:       continue;
1694:     }
1695:
1696:     /* if the connection has a callback function then it is passed the packet
1697:        and the packet is not added to the list */
1698:     if (con && con->callback) {
1699: #ifdef __GNUC__
1700:       pthread_t t;
```

```
1701: #else
1702:       HANDLE t;
1703: #endif
1704:       t_callback_list *l, *q;
1705:
1706:       xbee_mutex_lock(con->callbackListmutex);
1707:       l = con->callbackList;
1708:       q = NULL;
1709:       while (l) {
1710:         q = l;
1711:         l = l->next;
1712:       }
1713:       l = Xcalloc(sizeof(t_callback_list));
1714:       l->pkt = p;
1715:       if (!con->callbackList) {
1716:         con->callbackList = l;
1717:       } else {
1718:         q->next = l;
1719:       }
1720:       xbee_mutex_unlock(con->callbackListmutex);
1721:
1722:       if (xbee.log) {
1723:         xbee_log("Using callback function!");
1724:         xbee_log("  info block @ 0x%08X",l);
1725:         xbee_log("  function   @ 0x%08X",con->callback);
1726:         xbee_log("  connection @ 0x%08X",con);
1727:         xbee_log("  packet     @ 0x%08X",p);
1728:       }
1729:
1730:       /* if the callback thread not still running, then start a new one! */
1731:       if (!xbee_mutex_trylock(con->callbackmutex)) {
1732:         if (xbee.log) xbee_log("Starting new callback thread!");
1733:         xbee_thread_create(t,xbee_callbackWrapper,con);
1734:       } else if (xbee.log) {
1735:         xbee_log("Using existing callback thread");
1736:       }
1737:       continue;
1738:     }
1739:
1740:     /* lock the packet mutex, so we can safely add the packet to the list */
1741:     xbee_mutex_lock(xbee.pktmutex);
1742:
1743:     /* if: the list is empty */
1744:     if (!xbee.pktlist) {
1745:       /* start the list! */
1746:       xbee.pktlist = p;
1747:     } else if (xbee.pktlast) {
1748:       /* add the packet to the end */
1749:       xbee.pktlast->next = p;
1750:     } else {
1751:       /* pktlast wasnt set... look for the end and then set it */
1752:       i = 0;
1753:       q = xbee.pktlist;
1754:       while (q->next) {
1755:         q = q->next;
1756:         i++;
1757:       }
1758:       q->next = p;
1759:       xbee.pktcount = i;
1760:     }
1761:     xbee.pktlast = p;
1762:     xbee.pktcount++;
1763:
1764:     /* unlock the packet mutex */
1765:     xbee_mutex_unlock(xbee.pktmutex);
1766:
1767:     if (xbee.log) {
1768:       xbee_log("--========================--");
1769:       xbee_log("Packets: %d",xbee.pktcount);
1770:     }
1771:
1772:     p = q = NULL;
1773:   }
1774:   return 0;
1775: }
1776: static void xbee_callbackWrapper(xbee_con *con) {
1777:   xbee_pkt *pkt;
1778:   t_callback_list *temp;
1779:   /* dont forget! the callback mutex is already locked... by the parent thread :) */
1780:
1781:   xbee_mutex_lock(con->callbackListmutex);
1782:   while (con->callbackList) {
1783:     /* shift the list along 1 */
1784:     temp = con->callbackList;
1785:     con->callbackList = temp->next;
```

```
1786:     /* get the packet */
1787:     pkt = temp->pkt;
1788:     xbee_mutex_unlock(con->callbackListmutex);
1789:
1790:     if (xbee.log) {
1791:       xbee_log("Starting callback function...");
1792:       xbee_log("  info block @ 0x%08X",temp);
1793:       xbee_log("  function   @ 0x%08X",con->callback);
1794:       xbee_log("  connection @ 0x%08X",con);
1795:       xbee_log("  packet     @ 0x%08X",pkt);
1796:     }
1797:     Xfree(temp);
1798:     con->callback(con,pkt);
1799:     if (xbee.log) xbee_log("Callback complete!");
1800:     Xfree(pkt);
1801:
1802:     xbee_mutex_lock(con->callbackListmutex);
1803:   }
1804:   xbee_mutex_unlock(con->callbackListmutex);
1805:
1806:   if (xbee.log) xbee_log("Callback thread ending...");
1807:   /* releasing the thread mutex is the last thing we do! */
1808:   xbee_mutex_unlock(con->callbackmutex);
1809:
1810:   if (con->destroySelf) {
1811:     xbee_endcon2(&con,1);
1812:   }
1813: }
1814:
1815: /* ################################################################
1816:    xbee_getbyte - INTERNAL
1817:    waits for an escaped byte of data */
1818: static unsigned char xbee_getbyte(void) {
1819:   unsigned char c;
1820:
1821:   ISREADY;
1822:
1823:   /* take a byte */
1824:   c = xbee_getrawbyte();
1825:   /* if its escaped, take another and un-escape */
1826:   if (c == 0x7D) c = xbee_getrawbyte() ^ 0x20;
1827:
1828:   return (c & 0xFF);
1829: }
1830:
1831: /* ################################################################
1832:    xbee_getrawbyte - INTERNAL
1833:    waits for a raw byte of data */
1834: static unsigned char xbee_getrawbyte(void) {
1835:   int ret;
1836:   unsigned char c = 0x00;
1837:
1838:   ISREADY;
1839:
1840:   /* the loop is just incase there actually isnt a byte there to be read... */
1841:   do {
1842:     /* wait for a read to be possible */
1843:     if ((ret = xbee_select(NULL)) == -1) {
1844:       perror("libxbee:xbee_getrawbyte()");
1845:       exit(1);
1846:     }
1847:     if (!xbee.listenrun) break;
1848:     if (ret == 0) continue;
1849:
1850:     /* read 1 character */
1851:     xbee_read(&c,1);
1852: #ifdef _WIN32 /* ---- */
1853:     ret = xbee.ttyr;
1854:     if (ret == 0) {
1855:       usleep(10);
1856:       continue;
1857:     }
1858: #endif /* ----------- */
1859:   } while (0);
1860:
1861:   return (c & 0xFF);
1862: }
1863:
1864: /* ################################################################
1865:    xbee_send_pkt - INTERNAL
1866:    sends a complete packet of data */
1867: static int xbee_send_pkt(t_data *pkt, xbee_con *con) {
1868:   int retval = 0;
1869:   ISREADY;
1870:
```

```
1871:   /* lock connection mutex */
1872:   xbee_mutex_lock(con->Txmutex);
1873:   /* lock the send mutex */
1874:   xbee_mutex_lock(xbee.sendmutex);
1875:
1876:   /* write and flush the data */
1877:   xbee_write(pkt->data,pkt->length);
1878:
1879:   /* unlock the mutex */
1880:   xbee_mutex_unlock(xbee.sendmutex);
1881:
1882:   if (xbee.log) {
1883:     int i,x,y;
1884:     /* prints packet in hex byte-by-byte */
1885:     xbee_logc("TX Packet:");
1886:     for (i=0,x=0,y=0;i<pkt->length;i++,x--) {
1887:       if (x == 0) {
1888:         fprintf(xbee.log,"\n  0x%04X | ",y);
1889:         x = 0x8;
1890:         y += x;
1891:       }
1892:       if (x == 4) {
1893:         fprintf(xbee.log,"  ");
1894:       }
1895:       fprintf(xbee.log,"0x%02X ",pkt->data[i]);
1896:     }
1897:     xbee_logcf();
1898:   }
1899:
1900:   if (con->waitforACK &&
1901:       ((con->type == xbee_16bitData) ||
1902:        (con->type == xbee_64bitData))) {
1903:     con->ACKstatus = 1;
1904:     if (xbee.log) xbee_log("Waiting for ACK/NAK response...");
1905:     xbee_sem_wait(con->waitforACKsem);
1906:     if (con->ACKstatus == 2) {
1907:       if (xbee.log) xbee_log("ACK recieved!");
1908:     } else {
1909:       retval = 1; /* error */
1910:       if (xbee.log) xbee_log("NAK recieved...");
1911:     }
1912:   }
1913:
1914:   /* unlock connection mutex */
1915:   xbee_mutex_unlock(con->Txmutex);
1916:
1917:   /* free the packet */
1918:   Xfree(pkt);
1919:
1920:   return retval;
1921: }
1922:
1923: /* ########################################################################
1924:    xbee_make_pkt - INTERNAL
1925:    adds delimiter field
1926:    calculates length and checksum
1927:    escapes bytes */
1928: static t_data *xbee_make_pkt(unsigned char *data, int length) {
1929:   t_data *pkt;
1930:   unsigned int l, i, o, t, x, m;
1931:   char d = 0;
1932:
1933:   ISREADY;
1934:
1935:   /* check the data given isnt too long
1936:      100 bytes maximum payload + 12 bytes header information */
1937:   if (length > 100 + 12) return NULL;
1938:
1939:   /* calculate the length of the whole packet
1940:      start, length (MSB), length (LSB), DATA, checksum */
1941:   l = 3 + length + 1;
1942:
1943:   /* prepare memory */
1944:   pkt = Xcalloc(sizeof(t_data));
1945:
1946:   /* put start byte on */
1947:   pkt->data[0] = 0x7E;
1948:
1949:   /* copy data into packet */
1950:   for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
1951:     /* if: its time for the checksum */
1952:     if (i == length) d = M8((0xFF - M8(t)));
1953:     /* if: its time for the high length byte */
1954:     else if (m == 1) d = M8(length >> 8);
1955:     /* if: its time for the low length byte */
```

```
1956:     else if (m == 2) d = M8(length);
1957:     /* if: its time for the normal data */
1958:     else if (m > 2) d = data[i];
1959:
1960:     x = 0;
1961:     /* check for any escapes needed */
1962:     if ((d == 0x11) || /* XON */
1963:         (d == 0x13) || /* XOFF */
1964:         (d == 0x7D) || /* Escape */
1965:         (d == 0x7E)) { /* Frame Delimiter */
1966:       l++;
1967:       pkt->data[o++] = 0x7D;
1968:       x = 1;
1969:     }
1970:
1971:     /* move data in */
1972:     pkt->data[o] = ((!x)?d:d^0x20);
1973:     if (m > 2) {
1974:       i++;
1975:       t += d;
1976:     }
1977:   }
1978:
1979:   /* remember the length */
1980:   pkt->length = l;
1981:
1982:   return pkt;
1983: }
```