```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 451 2010-12-02 03:38:07Z attie.co.uk $";
char svn_rev[128] = "\0";

#include "api.h"

void ISREADY(xbee_hnd xbee) {
  if (!xbee || !xbee->xbee_ready) {
    if (stderr) fprintf(stderr,"libxbee: Run xbee_setup() first!...\n");
#ifdef _WIN32
    MessageBox(0,"Run xbee_setup() first!...","libxbee",MB_OK);
#endif
    exit(1);
  }
}

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... */
    fprintf(stderr,"libxbee:realloc(): Returned NULL\n");
```

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

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

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

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

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

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

```
596:
597:   usleep(500);
598:   while (xbee->xbee_ready != -2) {
599:     usleep(500);
600:     xbee_log("Waiting for xbee_listen() to be ready...");
601:   }
602:
603:   /* allow other functions to be used! */
604:   xbee->xbee_ready = 1;
605:
606:   xbee_log("Linking xbee instance...");
607:   if (!default_xbee) {
608:     xbee_mutex_init(xbee_hnd_mutex);
609:     xbee_mutex_lock(xbee_hnd_mutex);
610:     default_xbee = xbee;
611:     xbee_mutex_unlock(xbee_hnd_mutex);
612:   } else {
613:     xbee_hnd xbeet;
614:     xbee_mutex_lock(xbee_hnd_mutex);
615:     xbeet = default_xbee;
616:     while (xbeet->next) {
617:       xbeet = xbeet->next;
618:     }
619:     xbeet->next = xbee;
620:     xbee_mutex_unlock(xbee_hnd_mutex);
621:   }
622:
623:   xbee_log("libxbee: Started!");
624:
625:   return xbee;
626: }
627:
628: /* ########################################################################
629:    xbee_con
630:    produces a connection to the specified device and frameID
631:    if a connection had already been made, then this connection will be returned */
632: xbee_con *xbee_newcon(unsigned char frameID, xbee_types type, ...) {
633:   xbee_con *ret;
634:   va_list ap;
635:
636:   /* xbee_vsenddata() wants a va_list... */
637:   va_start(ap, type);
638:   /* hand it over :) */
639:   ret = _xbee_vnewcon(default_xbee, frameID, type, ap);
640:   va_end(ap);
641:   return ret;
642: }
643: xbee_con *_xbee_newcon(xbee_hnd xbee, unsigned char frameID, xbee_types type, ...) {
644:   xbee_con *ret;
645:   va_list ap;
646:
647:   /* xbee_vsenddata() wants a va_list... */
648:   va_start(ap, type);
649:   /* hand it over :) */
650:   ret = _xbee_vnewcon(xbee, frameID, type, ap);
651:   va_end(ap);
652:   return ret;
653: }
654: xbee_con *_xbee_vnewcon(xbee_hnd xbee, unsigned char frameID, xbee_types type, va_list ap) {
655:   xbee_con *con, *ocon;
656:   unsigned char tAddr[8];
657:   int t;
658:   int i;
659:
660:   ISREADY(xbee);
661:
662:   if (!type || type == xbee_unknown) type = xbee_localAT; /* default to local AT */
663:   else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */
664:
665:   /* if: 64 bit address expected (2 ints) */
666:   if ((type == xbee_64bitRemoteAT) ||
667:       (type == xbee_64bitData) ||
668:       (type == xbee_64bitIO)) {
669:     t = va_arg(ap, int);
670:     tAddr[0] = (t >> 24) & 0xFF;
671:     tAddr[1] = (t >> 16) & 0xFF;
672:     tAddr[2] = (t >>  8) & 0xFF;
673:     tAddr[3] = (t      ) & 0xFF;
674:     t = va_arg(ap, int);
675:     tAddr[4] = (t >> 24) & 0xFF;
676:     tAddr[5] = (t >> 16) & 0xFF;
677:     tAddr[6] = (t >>  8) & 0xFF;
678:     tAddr[7] = (t      ) & 0xFF;
679:
680:     /* if: 16 bit address expected (1 int) */
```

```
681:   } else if ((type == xbee_16bitRemoteAT) ||
682:              (type == xbee_16bitData) ||
683:              (type == xbee_16bitIO)) {
684:     t = va_arg(ap, int);
685:     tAddr[0] = (t >>  8) & 0xFF;
686:     tAddr[1] = (t      ) & 0xFF;
687:     tAddr[2] = 0;
688:     tAddr[3] = 0;
689:     tAddr[4] = 0;
690:     tAddr[5] = 0;
691:     tAddr[6] = 0;
692:     tAddr[7] = 0;
693:
694:     /* otherwise clear the address */
695:   } else {
696:     memset(tAddr,0,8);
697:   }
698:
699:   /* lock the connection mutex */
700:   xbee_mutex_lock(xbee->conmutex);
701:
702:   /* are there any connections? */
703:   if (xbee->conlist) {
704:     con = xbee->conlist;
705:     while (con) {
706:       /* if: after a modemStatus, and the types match! */
707:       if ((type == xbee_modemStatus) &&
708:           (con->type == type)) {
709:         xbee_mutex_unlock(xbee->conmutex);
710:         return con;
711:
712:         /* if: after a txStatus and frameIDs match! */
713:       } else if ((type == xbee_txStatus) &&
714:                  (con->type == type) &&
715:                  (frameID == con->frameID)) {
716:         xbee_mutex_unlock(xbee->conmutex);
717:         return con;
718:
719:         /* if: after a localAT, and the frameIDs match! */
720:       } else if ((type == xbee_localAT) &&
721:                  (con->type == type) &&
722:                  (frameID == con->frameID)) {
723:         xbee_mutex_unlock(xbee->conmutex);
724:         return con;
725:
726:         /* if: connection types match, the frameIDs match, and the addresses match! */
727:       } else if ((type == con->type) &&
728:                  (frameID == con->frameID) &&
729:                  (!memcmp(tAddr,con->tAddr,8))) {
730:         xbee_mutex_unlock(xbee->conmutex);
731:         return con;
732:       }
733:
734:       /* if there are more, move along, dont want to loose that last item! */
735:       if (con->next == NULL) break;
736:       con = con->next;
737:     }
738:
739:     /* keep hold of the last connection... we will need to link it up later */
740:     ocon = con;
741:   }
742:
743:   /* create a new connection and set its attributes */
744:   con = Xcalloc(sizeof(xbee_con));
745:   con->type = type;
746:   /* is it a 64bit connection? */
747:   if ((type == xbee_64bitRemoteAT) ||
748:       (type == xbee_64bitData) ||
749:       (type == xbee_64bitIO)) {
750:     con->tAddr64 = TRUE;
751:   }
752:   con->atQueue = 0; /* queue AT commands? */
753:   con->txDisableACK = 0; /* disable ACKs? */
754:   con->txBroadcast = 0; /* broadcast? */
755:   con->frameID = frameID;
756:   con->waitforACK = 0;
757:   memcpy(con->tAddr,tAddr,8); /* copy in the remote address */
758:   xbee_mutex_init(con->callbackmutex);
759:   xbee_mutex_init(con->callbackListmutex);
760:   xbee_mutex_init(con->Txmutex);
761:   xbee_sem_init(con->waitforACKsem);
762:
763:   if (xbee->log) {
764:     switch(type) {
765:     case xbee_localAT:
```

```
766:       xbee_log("New local AT connection!");
767:       break;
768:     case xbee_16bitRemoteAT:
769:     case xbee_64bitRemoteAT:
770:       xbee_logc("New %d-bit remote AT connection! (to: ",(con->tAddr64?64:16));
771:       for (i=0;i<(con->tAddr64?8:2);i++) {
772:         fprintf(xbee->log,(i?":%02X":"%02X"),tAddr[i]);
773:       }
774:       fprintf(xbee->log,")");
775:       xbee_logcf(xbee);
776:       break;
777:     case xbee_16bitData:
778:     case xbee_64bitData:
779:       xbee_logc("New %d-bit data connection! (to: ",(con->tAddr64?64:16));
780:       for (i=0;i<(con->tAddr64?8:2);i++) {
781:         fprintf(xbee->log,(i?":%02X":"%02X"),tAddr[i]);
782:       }
783:       fprintf(xbee->log,")");
784:       xbee_logcf(xbee);
785:       break;
786:     case xbee_16bitIO:
787:     case xbee_64bitIO:
788:       xbee_logc("New %d-bit IO connection! (to: ",(con->tAddr64?64:16));
789:       for (i=0;i<(con->tAddr64?8:2);i++) {
790:         fprintf(xbee->log,(i?":%02X":"%02X"),tAddr[i]);
791:       }
792:       fprintf(xbee->log,")");
793:       xbee_logcf(xbee);
794:       break;
795:     case xbee_txStatus:
796:       xbee_log("New Tx status connection!");
797:       break;
798:     case xbee_modemStatus:
799:       xbee_log("New modem status connection!");
800:       break;
801:     case xbee_unknown:
802:     default:
803:       xbee_log("New unknown connection!");
804:     }
805:   }
806:
807:   /* make it the last in the list */
808:   con->next = NULL;
809:   /* add it to the list */
810:   if (xbee->conlist) {
811:     ocon->next = con;
812:   } else {
813:     xbee->conlist = con;
814:   }
815:
816:   /* unlock the mutex */
817:   xbee_mutex_unlock(xbee->conmutex);
818:   return con;
819: }
820:
821: /* ####################################################################
822:    xbee_conflush
823:    removes any packets that have been collected for the specified
824:    connection */
825: void xbee_flushcon(xbee_con *con) {
826:   _xbee_flushcon(default_xbee, con);
827: }
828: void _xbee_flushcon(xbee_hnd xbee, xbee_con *con) {
829:   xbee_pkt *r, *p, *n;
830:
831:   ISREADY(xbee);
832:
833:   /* lock the packet mutex */
834:   xbee_mutex_lock(xbee->pktmutex);
835:
836:   /* if: there are packets */
837:   if ((p = xbee->pktlist) != NULL) {
838:     r = NULL;
839:     /* get all packets for this connection */
840:     do {
841:       /* does the packet match the connection? */
842:       if (xbee_matchpktcon(xbee,p,con)) {
843:         /* if it was the first packet */
844:         if (!r) {
845:           /* move the chain along */
846:           xbee->pktlist = p->next;
847:         } else {
848:           /* otherwise relink the list */
849:           r->next = p->next;
850:         }
```

```
851:         xbee->pktcount--;
852:
853:         /* free this packet! */
854:         n = p->next;
855:         Xfree(p);
856:         /* move on */
857:         p = n;
858:       } else {
859:         /* move on */
860:         r = p;
861:         p = p->next;
862:       }
863:     } while (p);
864:     xbee->pktlast = r;
865:   }
866:
867:   /* unlock the packet mutex */
868:   xbee_mutex_unlock(xbee->pktmutex);
869: }
870:
871: /* ####################################################################
872:    xbee_endcon
873:    close the unwanted connection
874:    free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
875: void xbee_endcon2(xbee_con **con, int alreadyUnlinked) {
876:   _xbee_endcon2(default_xbee, con, alreadyUnlinked);
877: }
878: void _xbee_endcon2(xbee_hnd xbee, xbee_con **con, int alreadyUnlinked) {
879:   xbee_con *t, *u;
880:
881:   ISREADY(xbee);
882:
883:   /* lock the connection mutex */
884:   xbee_mutex_lock(xbee->conmutex);
885:
886:   u = t = xbee->conlist;
887:   while (t && t != *con) {
888:     u = t;
889:     t = t->next;
890:   }
891:   if (!t) {
892:     /* this could be true if comming from the destroySelf signal... */
893:     if (!alreadyUnlinked) {
894:       /* invalid connection given... */
895:       if (xbee->log) {
896:         xbee_log("Attempted to close invalid connection...");
897:       }
898:       /* unlock the connection mutex */
899:       xbee_mutex_unlock(xbee->conmutex);
900:       return;
901:     }
902:   } else {
903:     /* extract this connection from the list */
904:     if (t == xbee->conlist) {
905:       xbee->conlist = t->next;
906:     } else {
907:       u->next = t->next;
908:     }
909:   }
910:
911:   /* unlock the connection mutex */
912:   xbee_mutex_unlock(xbee->conmutex);
913:
914:   /* check if a callback thread is running... */
915:   if (t->callback && xbee_mutex_trylock(t->callbackmutex)) {
916:     /* if it is running... tell it to destroy the connection on completion */
917:     xbee_log("Attempted to close a connection with active callbacks... "
918:              "Connection will be destroied when callbacks have completeted...");
919:     t->destroySelf = 1;
920:     return;
921:   }
922:
923:   /* remove all packets for this connection */
924:   _xbee_flushcon(xbee,t);
925:
926:   /* destroy the callback mutex */
927:   xbee_mutex_destroy(t->callbackmutex);
928:   xbee_mutex_destroy(t->callbackListmutex);
929:   xbee_mutex_destroy(t->Txmutex);
930:   xbee_sem_destroy(t->waitforACKsem);
931:
932:   /* free the connection! */
933:   Xfree(*con);
934: }
935:
```

```
 936: /* ###################################################################
 937:    xbee_senddata
 938:    send the specified data to the provided connection */
 939: int xbee_senddata(xbee_con *con, char *format, ...) {
 940:   int ret;
 941:   va_list ap;
 942:
 943:   /* xbee_vsenddata() wants a va_list... */
 944:   va_start(ap, format);
 945:   /* hand it over :) */
 946:   ret = _xbee_vsenddata(default_xbee, con, format, ap);
 947:   va_end(ap);
 948:   return ret;
 949: }
 950: int _xbee_senddata(xbee_hnd xbee, xbee_con *con, char *format, ...) {
 951:   int ret;
 952:   va_list ap;
 953:
 954:   /* xbee_vsenddata() wants a va_list... */
 955:   va_start(ap, format);
 956:   /* hand it over :) */
 957:   ret = _xbee_vsenddata(xbee, con, format, ap);
 958:   va_end(ap);
 959:   return ret;
 960: }
 961:
 962: int xbee_vsenddata(xbee_con *con, char *format, va_list ap) {
 963:   return _xbee_vsenddata(default_xbee, con, format, ap);
 964: }
 965: int _xbee_vsenddata(xbee_hnd xbee, xbee_con *con, char *format, va_list ap) {
 966:   unsigned char data[128]; /* max payload is 100 bytes... plus a bit of fluff... */
 967:   int length;
 968:
 969:   /* make up the data and keep the length, its possible there are nulls in there */
 970:   length = vsnprintf((char *)data, 128, format, ap);
 971:
 972:   /* hand it over :) */
 973:   return _xbee_nsenddata(xbee, con, (char *)data, length);
 974: }
 975:
 976: /* returns:
 977:     1 - if NAC was recieved
 978:     0 - if packet was successfully sent (or just sent if waitforACK is off)
 979:    -1 - if there was an error building the packet
 980:    -2 - if the connection type was unknown */
 981: int xbee_nsenddata(xbee_con *con, char *data, int length) {
 982:   return _xbee_nsenddata(default_xbee, con, data, length);
 983: }
 984: int _xbee_nsenddata(xbee_hnd xbee, xbee_con *con, char *data, int length) {
 985:   t_data *pkt;
 986:   int i;
 987:   unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
 988:
 989:   ISREADY(xbee);
 990:
 991:   if (!con) return -1;
 992:   if (con->type == xbee_unknown) return -1;
 993:   if (length > 127) return -1;
 994:
 995:   if (xbee->log) {
 996:     xbee_log("--== TX Packet ============--");
 997:     xbee_logc("Connection Type: ");
 998:     switch (con->type) {
 999:     case xbee_unknown:       fprintf(xbee->log,"Unknown"); break;
1000:     case xbee_localAT:       fprintf(xbee->log,"Local AT"); break;
1001:     case xbee_remoteAT:      fprintf(xbee->log,"Remote AT"); break;
1002:     case xbee_16bitRemoteAT: fprintf(xbee->log,"Remote AT (16-bit)"); break;
1003:     case xbee_64bitRemoteAT: fprintf(xbee->log,"Remote AT (64-bit)"); break;
1004:     case xbee_16bitData:     fprintf(xbee->log,"Data (16-bit)"); break;
1005:     case xbee_64bitData:     fprintf(xbee->log,"Data (64-bit)"); break;
1006:     case xbee_16bitIO:       fprintf(xbee->log,"IO (16-bit)"); break;
1007:     case xbee_64bitIO:       fprintf(xbee->log,"IO (64-bit)"); break;
1008:     case xbee_txStatus:      fprintf(xbee->log,"Tx Status"); break;
1009:     case xbee_modemStatus:   fprintf(xbee->log,"Modem Status"); break;
1010:     }
1011:     xbee_logcf(xbee);
1012:     xbee_logc("Destination: ");
1013:     for (i=0;i<(con->tAddr64?8:2);i++) {
1014:       fprintf(xbee->log,(i?":%02X":"%02X"),con->tAddr[i]);
1015:     }
1016:     xbee_logcf(xbee);
1017:     xbee_log("Length: %d",length);
1018:     for (i=0;i<length;i++) {
1019:       xbee_logc("%3d | 0x%02X ",i,(unsigned char)data[i]);
1020:       if ((data[i] > 32) && (data[i] < 127)) {
```

```
1021:         fprintf(xbee->log,"'%c'",data[i]);
1022:       } else{
1023:         fprintf(xbee->log," _");
1024:       }
1025:       xbee_logcf(xbee);
1026:     }
1027:   }
1028:
1029:   /* ######################################### */
1030:   /* if: local AT */
1031:   if (con->type == xbee_localAT) {
1032:     /* AT commands are 2 chars long (plus optional parameter) */
1033:     if (length < 2) return -1;
1034:
1035:     /* use the command? */
1036:     buf[0] = ((!con->atQueue)?XBEE_LOCAL_ATREQ:XBEE_LOCAL_ATQUE);
1037:     buf[1] = con->frameID;
1038:
1039:     /* copy in the data */
1040:     for (i=0;i<length;i++) {
1041:       buf[i+2] = data[i];
1042:     }
1043:
1044:     /* setup the packet */
1045:     pkt = xbee_make_pkt(xbee, buf, i+2);
1046:     /* send it on */
1047:     return xbee_send_pkt(xbee, pkt, con);
1048:
1049:     /* ######################################### */
1050:     /* if: remote AT */
1051:   } else if ((con->type == xbee_16bitRemoteAT) ||
1052:              (con->type == xbee_64bitRemoteAT)) {
1053:     if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter) */
1054:     buf[0] = XBEE_REMOTE_ATREQ;
1055:     buf[1] = con->frameID;
1056:
1057:     /* copy in the relevant address */
1058:     if (con->tAddr64) {
1059:       memcpy(&buf[2],con->tAddr,8);
1060:       buf[10] = 0xFF;
1061:       buf[11] = 0xFE;
1062:     } else {
1063:       memset(&buf[2],0,8);
1064:       memcpy(&buf[10],con->tAddr,2);
1065:     }
1066:     /* queue the command? */
1067:     buf[12] = ((!con->atQueue)?0x02:0x00);
1068:
1069:     /* copy in the data */
1070:     for (i=0;i<length;i++) {
1071:       buf[i+13] = data[i];
1072:     }
1073:
1074:     /* setup the packet */
1075:     pkt = xbee_make_pkt(xbee, buf, i+13);
1076:     /* send it on */
1077:     return xbee_send_pkt(xbee, pkt, con);
1078:
1079:     /* ######################################### */
1080:     /* if: 16 or 64bit Data */
1081:   } else if ((con->type == xbee_16bitData) ||
1082:              (con->type == xbee_64bitData)) {
1083:     int offset;
1084:
1085:     /* if: 16bit Data */
1086:     if (con->type == xbee_16bitData) {
1087:       buf[0] = XBEE_16BIT_DATATX;
1088:       offset = 5;
1089:       /* copy in the address */
1090:       memcpy(&buf[2],con->tAddr,2);
1091:
1092:       /* if: 64bit Data */
1093:     } else { /* 64bit Data */
1094:       buf[0] = XBEE_64BIT_DATATX;
1095:       offset = 11;
1096:       /* copy in the address */
1097:       memcpy(&buf[2],con->tAddr,8);
1098:     }
1099:
1100:     /* copy frameID */
1101:     buf[1] = con->frameID;
1102:
1103:     /* disable ack? broadcast? */
1104:     buf[offset-1] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
1105:
```

```
1106:     /* copy in the data */
1107:     for (i=0;i<length;i++) {
1108:       buf[i+offset] = data[i];
1109:     }
1110:
1111:     /* setup the packet */
1112:     pkt = xbee_make_pkt(xbee, buf, i+offset);
1113:     /* send it on */
1114:     return xbee_send_pkt(xbee, pkt, con);
1115:
1116:     /* ######################################### */
1117:     /* if: I/O */
1118:   } else if ((con->type == xbee_64bitIO) ||
1119:              (con->type == xbee_16bitIO)) {
1120:     /* not currently implemented... is it even allowed? */
1121:     if (xbee->log) {
1122:       xbee_log("******* TODO ********\n");
1123:     }
1124:   }
1125:
1126:   return -2;
1127: }
1128:
1129: /* ################################################################
1130:    xbee_getpacket
1131:    retrieves the next packet destined for the given connection
1132:    once the packet has been retrieved, it is removed for the list! */
1133: xbee_pkt *xbee_getpacketwait(xbee_con *con) {
1134:   return _xbee_getpacketwait(default_xbee, con);
1135: }
1136: xbee_pkt *_xbee_getpacketwait(xbee_hnd xbee, xbee_con *con) {
1137:   xbee_pkt *p = NULL;
1138:   int i = 20;
1139:
1140:   /* 50ms * 20 = 1 second */
1141:   for (; i; i--) {
1142:     p = _xbee_getpacket(xbee, con);
1143:     if (p) break;
1144:     usleep(50000); /* 50ms */
1145:   }
1146:
1147:   return p;
1148: }
1149: xbee_pkt *xbee_getpacket(xbee_con *con) {
1150:   return _xbee_getpacket(default_xbee, con);
1151: }
1152: xbee_pkt *_xbee_getpacket(xbee_hnd xbee, xbee_con *con) {
1153:   xbee_pkt *l, *p, *q;
1154:
1155:   ISREADY(xbee);
1156:
1157:   /* lock the packet mutex */
1158:   xbee_mutex_lock(xbee->pktmutex);
1159:
1160:   /* if: there are no packets */
1161:   if ((p = xbee->pktlist) == NULL) {
1162:     xbee_mutex_unlock(xbee->pktmutex);
1163:     /*if (xbee->log) {
1164:       xbee_log("No packets avaliable...");
1165:       }*/
1166:     return NULL;
1167:   }
1168:
1169:   l = NULL;
1170:   q = NULL;
1171:   /* get the first avaliable packet for this connection */
1172:   do {
1173:     /* does the packet match the connection? */
1174:     if (xbee_matchpktcon(xbee, p, con)) {
1175:       q = p;
1176:       break;
1177:     }
1178:     /* move on */
1179:     l = p;
1180:     p = p->next;
1181:   } while (p);
1182:
1183:   /* if: no packet was found */
1184:   if (!q) {
1185:     xbee_mutex_unlock(xbee->pktmutex);
1186:     if (xbee->log) {
1187:       struct timeval tv;
1188:       xbee_log("--== Get Packet ==========--");
1189:       gettimeofday(&tv,NULL);
1190:       xbee_log("Didn't get a packet @ %ld.%06ld",tv.tv_sec,tv.tv_usec);
```

```
1191:     }
1192:     return NULL;
1193:   }
1194:
1195:   /* if it was the first packet */
1196:   if (l) {
1197:     /* relink the list */
1198:     l->next = p->next;
1199:     if (!l->next) xbee->pktlast = l;
1200:   } else {
1201:     /* move the chain along */
1202:     xbee->pktlist = p->next;
1203:     if (!xbee->pktlist) {
1204:       xbee->pktlast = NULL;
1205:     } else if (!xbee->pktlist->next) {
1206:       xbee->pktlast = xbee->pktlist;
1207:     }
1208:   }
1209:   xbee->pktcount--;
1210:
1211:   /* unlink this packet from the chain! */
1212:   q->next = NULL;
1213:
1214:   if (xbee->log) {
1215:     struct timeval tv;
1216:     xbee_log("--== Get Packet ==========--");
1217:     gettimeofday(&tv,NULL);
1218:     xbee_log("Got a packet @ %ld.%06ld",tv.tv_sec,tv.tv_usec);
1219:     xbee_log("Packets left: %d",xbee->pktcount);
1220:   }
1221:
1222:   /* unlock the packet mutex */
1223:   xbee_mutex_unlock(xbee->pktmutex);
1224:
1225:   /* and return the packet (must be free'd by caller!) */
1226:   return q;
1227: }
1228:
1229: /* ########################################################################
1230:    xbee_matchpktcon - INTERNAL
1231:    checks if the packet matches the connection */
1232: static int xbee_matchpktcon(xbee_hnd xbee, xbee_pkt *pkt, xbee_con *con) {
1233:   /* if: the connection type matches the packet type OR
1234:      the connection is 16/64bit remote AT, and the packet is a remote AT response */
1235:   if ((pkt->type == con->type) || /* -- */
1236:       ((pkt->type == xbee_remoteAT) && /* -- */
1237:        ((con->type == xbee_16bitRemoteAT) ||
1238:         (con->type == xbee_64bitRemoteAT)))) {
1239:
1240:
1241:     /* if: is a modem status (there can only be 1 modem status connection) */
1242:     if (pkt->type == xbee_modemStatus) return 1;
1243:
1244:     /* if: the packet is a txStatus or localAT and the frameIDs match */
1245:     if ((pkt->type == xbee_txStatus) ||
1246:         (pkt->type == xbee_localAT)) {
1247:       if (pkt->frameID == con->frameID) {
1248:         return 1;
1249:       }
1250:     /* if: the packet was sent as a 16bit remoteAT, and the 16bit addresss match */
1251:     } else if ((pkt->type == xbee_remoteAT) &&
1252:                (con->type == xbee_16bitRemoteAT) &&
1253:                !memcmp(pkt->Addr16,con->tAddr,2)) {
1254:       return 1;
1255:     /* if: the packet was sent as a 64bit remoteAT, and the 64bit addresss match */
1256:     } else if ((pkt->type == xbee_remoteAT) &&
1257:                (con->type == xbee_64bitRemoteAT) &&
1258:                !memcmp(pkt->Addr64,con->tAddr,8)) {
1259:       return 1;
1260:     /* if: the packet is 64bit addressed, and the addresses match */
1261:     } else if (pkt->sAddr64 && !memcmp(pkt->Addr64,con->tAddr,8)) {
1262:       return 1;
1263:     /* if: the packet is 16bit addressed, and the addresses match */
1264:     } else if (!pkt->sAddr64 && !memcmp(pkt->Addr16,con->tAddr,2)) {
1265:       return 1;
1266:     }
1267:   }
1268:   return 0;
1269: }
1270:
1271: /* ########################################################################
1272:    xbee_parse_io - INTERNAL
1273:    parses the data given into the packet io information */
1274: static int xbee_parse_io(xbee_hnd xbee, xbee_pkt *p, unsigned char *d,
1275:                          int maskOffset, int sampleOffset, int sample) {
```

```
1276:   xbee_sample *s = &(p->IOdata[sample]);
1277:
1278:   /* copy in the I/O data mask */
1279:   s->IOmask = (((d[maskOffset]<<8) | d[maskOffset + 1]) & 0x7FFF);
1280:
1281:   /* copy in the digital I/O data */
1282:   s->IOdigital = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x01FF);
1283:
1284:   /* advance over the digital data, if its there */
1285:   sampleOffset += ((s->IOmask & 0x01FF)?2:0);
1286:
1287:   /* copy in the analog I/O data */
1288:   if (s->IOmask & 0x0200) {
1289:     s->IOanalog[0] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1290:     sampleOffset+=2;
1291:   }
1292:   if (s->IOmask & 0x0400) {
1293:     s->IOanalog[1] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1294:     sampleOffset+=2;
1295:   }
1296:   if (s->IOmask & 0x0800) {
1297:     s->IOanalog[2] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1298:     sampleOffset+=2;
1299:   }
1300:   if (s->IOmask & 0x1000) {
1301:     s->IOanalog[3] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1302:     sampleOffset+=2;
1303:   }
1304:   if (s->IOmask & 0x2000) {
1305:     s->IOanalog[4] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1306:     sampleOffset+=2;
1307:   }
1308:   if (s->IOmask & 0x4000) {
1309:     s->IOanalog[5] = (((d[sampleOffset]<<8) | d[sampleOffset+1]) & 0x03FF);
1310:     sampleOffset+=2;
1311:   }
1312:
1313:   if (xbee->log) {
1314:     if (s->IOmask & 0x0001)
1315:       xbee_log("Digital 0: %c",((s->IOdigital & 0x0001)?'1':'0'));
1316:     if (s->IOmask & 0x0002)
1317:       xbee_log("Digital 1: %c",((s->IOdigital & 0x0002)?'1':'0'));
1318:     if (s->IOmask & 0x0004)
1319:       xbee_log("Digital 2: %c",((s->IOdigital & 0x0004)?'1':'0'));
1320:     if (s->IOmask & 0x0008)
1321:       xbee_log("Digital 3: %c",((s->IOdigital & 0x0008)?'1':'0'));
1322:     if (s->IOmask & 0x0010)
1323:       xbee_log("Digital 4: %c",((s->IOdigital & 0x0010)?'1':'0'));
1324:     if (s->IOmask & 0x0020)
1325:       xbee_log("Digital 5: %c",((s->IOdigital & 0x0020)?'1':'0'));
1326:     if (s->IOmask & 0x0040)
1327:       xbee_log("Digital 6: %c",((s->IOdigital & 0x0040)?'1':'0'));
1328:     if (s->IOmask & 0x0080)
1329:       xbee_log("Digital 7: %c",((s->IOdigital & 0x0080)?'1':'0'));
1330:     if (s->IOmask & 0x0100)
1331:       xbee_log("Digital 8: %c",((s->IOdigital & 0x0100)?'1':'0'));
1332:     if (s->IOmask & 0x0200)
1333:       xbee_log("Analog  0: %d (~%.2fv)",s->IOanalog[0],(3.3/1023)*s->IOanalog[0]);
1334:     if (s->IOmask & 0x0400)
1335:       xbee_log("Analog  1: %d (~%.2fv)",s->IOanalog[1],(3.3/1023)*s->IOanalog[1]);
1336:     if (s->IOmask & 0x0800)
1337:       xbee_log("Analog  2: %d (~%.2fv)",s->IOanalog[2],(3.3/1023)*s->IOanalog[2]);
1338:     if (s->IOmask & 0x1000)
1339:       xbee_log("Analog  3: %d (~%.2fv)",s->IOanalog[3],(3.3/1023)*s->IOanalog[3]);
1340:     if (s->IOmask & 0x2000)
1341:       xbee_log("Analog  4: %d (~%.2fv)",s->IOanalog[4],(3.3/1023)*s->IOanalog[4]);
1342:     if (s->IOmask & 0x4000)
1343:       xbee_log("Analog  5: %d (~%.2fv)",s->IOanalog[5],(3.3/1023)*s->IOanalog[5]);
1344:   }
1345:
1346:   return sampleOffset;
1347: }
1348:
1349: /* ####################################################################
1350:    xbee_listen_stop
1351:    stops the listen thread after the current packet has been processed */
1352: void xbee_listen_stop(xbee_hnd xbee) {
1353:   ISREADY(xbee);
1354:   xbee->listenrun = 0;
1355: }
1356:
1357: /* ####################################################################
1358:    xbee_listen_wrapper - INTERNAL
1359:    the xbee_listen wrapper. Prints an error when xbee_listen ends */
1360: static void xbee_listen_wrapper(t_LTinfo *info) {
```

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

```
1446:     for (i = 0; l > 1 && i < 128; l--, i++) {
1447:       /* get an unescaped byte */
1448:       c = xbee_getbyte(xbee);
1449:       d[i] = c;
1450:       chksum += c;
1451:       if (xbee->log) {
1452:         xbee_logc("%3d | 0x%02X | ",i,c);
1453:         if ((c > 32) && (c < 127)) fprintf(xbee->log,"'%c'",c); else fprintf(xbee->log," _ ");
1454:
1455:         if ((t == XBEE_64BIT_DATA && i == 10) ||
1456:             (t == XBEE_16BIT_DATA && i == 4) ||
1457:             (t == XBEE_64BIT_IO   && i == 13) ||
1458:             (t == XBEE_16BIT_IO   && i == 7) ||
1459:             (t == XBEE_LOCAL_AT   && i == 4) ||
1460:             (t == XBEE_REMOTE_AT  && i == 14)) {
1461:           /* mark the beginning of the 'data' bytes */
1462:           fprintf(xbee->log,"   <-- data starts");
1463:         } else if (t == XBEE_64BIT_IO) {
1464:           if (i == 10) fprintf(xbee->log,"   <-- sample count");
1465:           else if (i == 11) fprintf(xbee->log,"   <-- mask (msb)");
1466:           else if (i == 12) fprintf(xbee->log,"   <-- mask (lsb)");
1467:         } else if (t == XBEE_16BIT_IO) {
1468:           if (i == 4) fprintf(xbee->log,"   <-- sample count");
1469:           else if (i == 5) fprintf(xbee->log,"   <-- mask (msb)");
1470:           else if (i == 6) fprintf(xbee->log,"   <-- mask (lsb)");
1471:         }
1472:         xbee_logcf(xbee);
1473:       }
1474:     }
1475:     i--; /* it went up too many times!... */
1476:
1477:     /* add the checksum */
1478:     chksum += xbee_getbyte(xbee);
1479:
1480:     /* check if the whole packet was recieved, or something else occured... unlikely... */
1481:     if (l>1) {
1482:       if (xbee->log) {
1483:         xbee_log("Didn't get whole packet... :(");
1484:       }
1485:       continue;
1486:     }
1487:
1488:     /* check the checksum */
1489:     if ((chksum & 0xFF) != 0xFF) {
1490:       if (xbee->log) {
1491:         chksum &= 0xFF;
1492:         xbee_log("Invalid Checksum: 0x%02X",chksum);
1493:       }
1494:       continue;
1495:     }
1496:
1497:     /* make a new packet */
1498:     p = Xcalloc(sizeof(xbee_pkt));
1499:     q = NULL;
1500:     p->datalen = 0;
1501:
1502:     /* ########################################### */
1503:     /* if: modem status */
1504:     if (t == XBEE_MODEM_STATUS) {
1505:       if (xbee->log) {
1506:         xbee_log("Packet type: Modem Status (0x8A)");
1507:         xbee_logc("Event: ");
1508:         switch (d[0]) {
1509:         case 0x00: fprintf(xbee->log,"Hardware reset"); break;
1510:         case 0x01: fprintf(xbee->log,"Watchdog timer reset"); break;
1511:         case 0x02: fprintf(xbee->log,"Associated"); break;
1512:         case 0x03: fprintf(xbee->log,"Disassociated"); break;
1513:         case 0x04: fprintf(xbee->log,"Synchronization lost"); break;
1514:         case 0x05: fprintf(xbee->log,"Coordinator realignment"); break;
1515:         case 0x06: fprintf(xbee->log,"Coordinator started"); break;
1516:         }
1517:         fprintf(xbee->log,"... (0x%02X)",d[0]);
1518:         xbee_logcf(xbee);
1519:       }
1520:       p->type = xbee_modemStatus;
1521:
1522:       p->sAddr64 = FALSE;
1523:       p->dataPkt = FALSE;
1524:       p->txStatusPkt = FALSE;
1525:       p->modemStatusPkt = TRUE;
1526:       p->remoteATPkt = FALSE;
1527:       p->IOPkt = FALSE;
1528:
1529:       /* modem status can only ever give 1 'data' byte */
1530:       p->datalen = 1;
```

```
1531:       p->data[0] = d[0];
1532:
1533:       /* ######################################### */
1534:       /* if: local AT response */
1535:     } else if (t == XBEE_LOCAL_AT) {
1536:       if (xbee->log) {
1537:         xbee_log("Packet type: Local AT Response (0x88)");
1538:         xbee_log("FrameID: 0x%02X",d[0]);
1539:         xbee_log("AT Command: %c%c",d[1],d[2]);
1540:         xbee_logc("Status: ");
1541:         if (d[3] == 0) fprintf(xbee->log,"OK");
1542:         else if (d[3] == 1) fprintf(xbee->log,"Error");
1543:         else if (d[3] == 2) fprintf(xbee->log,"Invalid Command");
1544:         else if (d[3] == 3) fprintf(xbee->log,"Invalid Parameter");
1545:         fprintf(xbee->log," (0x%02X)",d[3]);
1546:         xbee_logcf(xbee);
1547:       }
1548:       p->type = xbee_localAT;
1549:
1550:       p->sAddr64 = FALSE;
1551:       p->dataPkt = FALSE;
1552:       p->txStatusPkt = FALSE;
1553:       p->modemStatusPkt = FALSE;
1554:       p->remoteATPkt = FALSE;
1555:       p->IOPkt = FALSE;
1556:
1557:       p->frameID = d[0];
1558:       p->atCmd[0] = d[1];
1559:       p->atCmd[1] = d[2];
1560:
1561:       p->status = d[3];
1562:
1563:       /* copy in the data */
1564:       p->datalen = i-3;
1565:       for (;i>3;i--) p->data[i-4] = d[i];
1566:
1567:       /* ######################################### */
1568:       /* if: remote AT response */
1569:     } else if (t == XBEE_REMOTE_AT) {
1570:       if (xbee->log) {
1571:         xbee_log("Packet type: Remote AT Response (0x97)");
1572:         xbee_log("FrameID: 0x%02X",d[0]);
1573:         xbee_logc("64-bit Address: ");
1574:         for (j=0;j<8;j++) {
1575:           fprintf(xbee->log,(j?":%02X":"%02X"),d[1+j]);
1576:         }
1577:         xbee_logcf(xbee);
1578:         xbee_logc("16-bit Address: ");
1579:         for (j=0;j<2;j++) {
1580:           fprintf(xbee->log,(j?":%02X":"%02X"),d[9+j]);
1581:         }
1582:         xbee_logcf(xbee);
1583:         xbee_log("AT Command: %c%c",d[11],d[12]);
1584:         xbee_logc("Status: ");
1585:         if (d[13] == 0) fprintf(xbee->log,"OK");
1586:         else if (d[13] == 1) fprintf(xbee->log,"Error");
1587:         else if (d[13] == 2) fprintf(xbee->log,"Invalid Command");
1588:         else if (d[13] == 3) fprintf(xbee->log,"Invalid Parameter");
1589:         else if (d[13] == 4) fprintf(xbee->log,"No Response");
1590:         fprintf(xbee->log," (0x%02X)",d[13]);
1591:         xbee_logcf(xbee);
1592:       }
1593:       p->type = xbee_remoteAT;
1594:
1595:       p->sAddr64 = FALSE;
1596:       p->dataPkt = FALSE;
1597:       p->txStatusPkt = FALSE;
1598:       p->modemStatusPkt = FALSE;
1599:       p->remoteATPkt = TRUE;
1600:       p->IOPkt = FALSE;
1601:
1602:       p->frameID = d[0];
1603:
1604:       p->Addr64[0] = d[1];
1605:       p->Addr64[1] = d[2];
1606:       p->Addr64[2] = d[3];
1607:       p->Addr64[3] = d[4];
1608:       p->Addr64[4] = d[5];
1609:       p->Addr64[5] = d[6];
1610:       p->Addr64[6] = d[7];
1611:       p->Addr64[7] = d[8];
1612:
1613:       p->Addr16[0] = d[9];
1614:       p->Addr16[1] = d[10];
1615:
```

```
1616:       p->atCmd[0] = d[11];
1617:       p->atCmd[1] = d[12];
1618:
1619:       p->status = d[13];
1620:
1621:       p->samples = 1;
1622:
1623:       if (p->status == 0x00 && p->atCmd[0] == 'I' && p->atCmd[1] == 'S') {
1624:         /* parse the io data */
1625:         xbee_log("--- Sample -----------------");
1626:         xbee_parse_io(xbee, p, d, 15, 17, 0);
1627:         xbee_log("----------------------------");
1628:       } else {
1629:         /* copy in the data */
1630:         p->datalen = i-13;
1631:         for (;i>13;i--) p->data[i-14] = d[i];
1632:       }
1633:
1634:       /* ######################################### */
1635:       /* if: TX status */
1636:     } else if (t == XBEE_TX_STATUS) {
1637:       if (xbee->log) {
1638:         xbee_log("Packet type: TX Status Report (0x89)");
1639:         xbee_log("FrameID: 0x%02X",d[0]);
1640:         xbee_logc("Status: ");
1641:         if (d[1] == 0) fprintf(xbee->log,"Success");
1642:         else if (d[1] == 1) fprintf(xbee->log,"No ACK");
1643:         else if (d[1] == 2) fprintf(xbee->log,"CCA Failure");
1644:         else if (d[1] == 3) fprintf(xbee->log,"Purged");
1645:         fprintf(xbee->log," (0x%02X)",d[1]);
1646:         xbee_logcf(xbee);
1647:       }
1648:       p->type = xbee_txStatus;
1649:
1650:       p->sAddr64 = FALSE;
1651:       p->dataPkt = FALSE;
1652:       p->txStatusPkt = TRUE;
1653:       p->modemStatusPkt = FALSE;
1654:       p->remoteATPkt = FALSE;
1655:       p->IOPkt = FALSE;
1656:
1657:       p->frameID = d[0];
1658:
1659:       p->status = d[1];
1660:
1661:       /* never returns data */
1662:       p->datalen = 0;
1663:
1664:       /* check for any connections waiting for a status update */
1665:       /* lock the connection mutex */
1666:       xbee_mutex_lock(xbee->conmutex);
1667:       xbee_log("Looking for a connection that wants a status update...");
1668:       con = xbee->conlist;
1669:       while (con) {
1670:         if ((con->frameID == p->frameID) &&
1671:             (con->ACKstatus == 255)) {
1672:           xbee_log("Found @ 0x%08X!",con);
1673:           con->ACKstatus = p->status;
1674:           xbee_sem_post(con->waitforACKsem);
1675:         }
1676:         con = con->next;
1677:       }
1678:
1679:       /* unlock the connection mutex */
1680:       xbee_mutex_unlock(xbee->conmutex);
1681:
1682:       /* ######################################### */
1683:       /* if: 16 / 64bit data recieve */
1684:     } else if ((t == XBEE_64BIT_DATA) ||
1685:                (t == XBEE_16BIT_DATA)) {
1686:       int offset;
1687:       if (t == XBEE_64BIT_DATA) { /* 64bit */
1688:         offset = 8;
1689:       } else { /* 16bit */
1690:         offset = 2;
1691:       }
1692:       if (xbee->log) {
1693:         xbee_log("Packet type: %d-bit RX Data (0x%02X)",((t == XBEE_64BIT_DATA)?64:16),t);
1694:         xbee_logc("%d-bit Address: ",((t == XBEE_64BIT_DATA)?64:16));
1695:         for (j=0;j<offset;j++) {
1696:           fprintf(xbee->log,(j?":%02X":"%02X"),d[j]);
1697:         }
1698:         xbee_logcf(xbee);
1699:         xbee_log("RSSI: -%ddB",d[offset]);
1700:         if (d[offset + 1] & 0x02) xbee_log("Options: Address Broadcast");
```

```
1701:         if (d[offset + 1] & 0x03) xbee_log("Options: PAN Broadcast");
1702:       }
1703:       p->dataPkt = TRUE;
1704:       p->txStatusPkt = FALSE;
1705:       p->modemStatusPkt = FALSE;
1706:       p->remoteATPkt = FALSE;
1707:       p->IOPkt = FALSE;
1708:
1709:       if (t == XBEE_64BIT_DATA) { /* 64bit */
1710:         p->type = xbee_64bitData;
1711:
1712:         p->sAddr64 = TRUE;
1713:
1714:         p->Addr64[0] = d[0];
1715:         p->Addr64[1] = d[1];
1716:         p->Addr64[2] = d[2];
1717:         p->Addr64[3] = d[3];
1718:         p->Addr64[4] = d[4];
1719:         p->Addr64[5] = d[5];
1720:         p->Addr64[6] = d[6];
1721:         p->Addr64[7] = d[7];
1722:       } else { /* 16bit */
1723:         p->type = xbee_16bitData;
1724:
1725:         p->sAddr64 = FALSE;
1726:
1727:         p->Addr16[0] = d[0];
1728:         p->Addr16[1] = d[1];
1729:       }
1730:
1731:       /* save the RSSI / signal strength
1732:          this can be used with printf as:
1733:          printf("-%ddB\n",p->RSSI); */
1734:       p->RSSI = d[offset];
1735:
1736:       p->status = d[offset + 1];
1737:
1738:       /* copy in the data */
1739:       p->datalen = i-(offset + 1);
1740:       for (;i>offset + 1;i--) p->data[i-(offset + 2)] = d[i];
1741:
1742:       /* ######################################### */
1743:       /* if: 16 / 64bit I/O recieve */
1744:     } else if ((t == XBEE_64BIT_IO) ||
1745:                (t == XBEE_16BIT_IO)) {
1746:       int offset,i2;
1747:       if (t == XBEE_64BIT_IO) { /* 64bit */
1748:         p->type = xbee_64bitIO;
1749:
1750:         p->sAddr64 = TRUE;
1751:
1752:         p->Addr64[0] = d[0];
1753:         p->Addr64[1] = d[1];
1754:         p->Addr64[2] = d[2];
1755:         p->Addr64[3] = d[3];
1756:         p->Addr64[4] = d[4];
1757:         p->Addr64[5] = d[5];
1758:         p->Addr64[6] = d[6];
1759:         p->Addr64[7] = d[7];
1760:
1761:         offset = 8;
1762:         p->samples = d[10];
1763:       } else { /* 16bit */
1764:         p->type = xbee_16bitIO;
1765:
1766:         p->sAddr64 = FALSE;
1767:
1768:         p->Addr16[0] = d[0];
1769:         p->Addr16[1] = d[1];
1770:
1771:         offset = 2;
1772:         p->samples = d[4];
1773:       }
1774:       if (p->samples > 1) {
1775:         p = Xrealloc(p, sizeof(xbee_pkt) + (sizeof(xbee_sample) * (p->samples - 1)));
1776:       }
1777:       if (xbee->log) {
1778:         xbee_log("Packet type: %d-bit RX I/O Data (0x%02X)",((t == XBEE_64BIT_IO)?64:16),t);
1779:         xbee_logc("%d-bit Address: ",((t == XBEE_64BIT_IO)?64:16));
1780:         for (j = 0; j < offset; j++) {
1781:           fprintf(xbee->log,(j?":%02X":"%02X"),d[j]);
1782:         }
1783:         xbee_logcf(xbee);
1784:         xbee_log("RSSI: -%ddB",d[offset]);
1785:         if (d[9] & 0x02) xbee_log("Options: Address Broadcast");
```

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

```
1871:       l->pkt = p;
1872:       if (!con->callbackList) {
1873:         con->callbackList = l;
1874:       } else {
1875:         q->next = l;
1876:       }
1877:       xbee_mutex_unlock(con->callbackListmutex);
1878:
1879:       xbee_log("Using callback function!");
1880:       xbee_log("  info block @ 0x%08X",l);
1881:       xbee_log("  function   @ 0x%08X",con->callback);
1882:       xbee_log("  connection @ 0x%08X",con);
1883:       xbee_log("  packet     @ 0x%08X",p);
1884:
1885:       /* if the callback thread not still running, then start a new one! */
1886:       if (!xbee_mutex_trylock(con->callbackmutex)) {
1887:         t_CBinfo info;
1888:         info.xbee = xbee;
1889:         info.con = con;
1890:         xbee_log("Starting new callback thread!");
1891:         xbee_thread_create(t,xbee_callbackWrapper,&info);
1892:       } else {
1893:         xbee_log("Using existing callback thread... callback has been scheduled.");
1894:       }
1895:       continue;
1896:     }
1897:
1898:     /* lock the packet mutex, so we can safely add the packet to the list */
1899:     xbee_mutex_lock(xbee->pktmutex);
1900:
1901:     /* if: the list is empty */
1902:     if (!xbee->pktlist) {
1903:       /* start the list! */
1904:       xbee->pktlist = p;
1905:     } else if (xbee->pktlast) {
1906:       /* add the packet to the end */
1907:       xbee->pktlast->next = p;
1908:     } else {
1909:       /* pktlast wasnt set... look for the end and then set it */
1910:       i = 0;
1911:       q = xbee->pktlist;
1912:       while (q->next) {
1913:         q = q->next;
1914:         i++;
1915:       }
1916:       q->next = p;
1917:       xbee->pktcount = i;
1918:     }
1919:     xbee->pktlast = p;
1920:     xbee->pktcount++;
1921:
1922:     /* unlock the packet mutex */
1923:     xbee_mutex_unlock(xbee->pktmutex);
1924:
1925:     xbee_log("--========================--");
1926:     xbee_log("Packets: %d",xbee->pktcount);
1927:
1928:     p = q = NULL;
1929:   }
1930:   return 0;
1931: }
1932:
1933: static void xbee_callbackWrapper(t_CBinfo *info) {
1934:   xbee_hnd xbee;
1935:   xbee_con *con;
1936:   xbee_pkt *pkt;
1937:   t_callback_list *temp;
1938:   xbee = info->xbee;
1939:   con = info->con;
1940:   /* dont forget! the callback mutex is already locked... by the parent thread :) */
1941:
1942:   xbee_mutex_lock(con->callbackListmutex);
1943:   while (con->callbackList) {
1944:     /* shift the list along 1 */
1945:     temp = con->callbackList;
1946:     con->callbackList = temp->next;
1947:     xbee_mutex_unlock(con->callbackListmutex);
1948:     /* get the packet */
1949:     pkt = temp->pkt;
1950:
1951:     xbee_log("Starting callback function...");
1952:     xbee_log("  info block @ 0x%08X",temp);
1953:     xbee_log("  function   @ 0x%08X",con->callback);
1954:     xbee_log("  connection @ 0x%08X",con);
1955:     xbee_log("  packet     @ 0x%08X",pkt);
```

```
1956:     Xfree(temp);
1957:     con->callback(con,pkt);
1958:     xbee_log("Callback complete!");
1959:     Xfree(pkt);
1960:
1961:     xbee_mutex_lock(con->callbackListmutex);
1962:   }
1963:   xbee_mutex_unlock(con->callbackListmutex);
1964:
1965:   xbee_log("Callback thread ending...");
1966:   /* releasing the thread mutex is the last thing we do! */
1967:   xbee_mutex_unlock(con->callbackmutex);
1968:
1969:   if (con->destroySelf) {
1970:     _xbee_endcon2(xbee,&con,1);
1971:   }
1972: }
1973:
1974: /* ####################################################################
1975:    xbee_getbyte - INTERNAL
1976:    waits for an escaped byte of data */
1977: static unsigned char xbee_getbyte(xbee_hnd xbee) {
1978:   unsigned char c;
1979:
1980:   /* take a byte */
1981:   c = xbee_getrawbyte(xbee);
1982:   /* if its escaped, take another and un-escape */
1983:   if (c == 0x7D) c = xbee_getrawbyte(xbee) ^ 0x20;
1984:
1985:   return (c & 0xFF);
1986: }
1987:
1988: /* ####################################################################
1989:    xbee_getrawbyte - INTERNAL
1990:    waits for a raw byte of data */
1991: static unsigned char xbee_getrawbyte(xbee_hnd xbee) {
1992:   int ret;
1993:   unsigned char c = 0x00;
1994:
1995:   /* the loop is just incase there actually isnt a byte there to be read... */
1996:   do {
1997:     /* wait for a read to be possible */
1998:     if ((ret = xbee_select(xbee,NULL)) == -1) {
1999:       perror("libxbee:xbee_getrawbyte()");
2000:       exit(1);
2001:     }
2002:     if (!xbee->listenrun) break;
2003:     if (ret == 0) continue;
2004:
2005:     /* read 1 character */
2006:     xbee_read(xbee,&c,1);
2007: #ifdef _WIN32 /* ---- */
2008:     ret = xbee->ttyr;
2009:     if (ret == 0) {
2010:       usleep(10);
2011:       continue;
2012:     }
2013: #endif /* ----------- */
2014:   } while (0);
2015:
2016:   return (c & 0xFF);
2017: }
2018:
2019: /* ####################################################################
2020:    xbee_send_pkt - INTERNAL
2021:    sends a complete packet of data */
2022: static int xbee_send_pkt(xbee_hnd xbee, t_data *pkt, xbee_con *con) {
2023:   int retval = 0;
2024:
2025:   /* lock connection mutex */
2026:   xbee_mutex_lock(con->Txmutex);
2027:   /* lock the send mutex */
2028:   xbee_mutex_lock(xbee->sendmutex);
2029:
2030:   /* write and flush the data */
2031:   xbee_write(xbee,pkt->data,pkt->length);
2032:
2033:   /* unlock the mutex */
2034:   xbee_mutex_unlock(xbee->sendmutex);
2035:
2036:   if (xbee->log) {
2037:     int i,x,y;
2038:     /* prints packet in hex byte-by-byte */
2039:     xbee_logc("TX Packet:");
2040:     for (i=0,x=0,y=0;i<pkt->length;i++,x--) {
```

```
2041:       if (x == 0) {
2042:         fprintf(xbee->log,"\n  0x%04X | ",y);
2043:         x = 0x8;
2044:         y += x;
2045:       }
2046:       if (x == 4) {
2047:         fprintf(xbee->log,"  ");
2048:       }
2049:       fprintf(xbee->log,"0x%02X ",pkt->data[i]);
2050:     }
2051:     xbee_logcf(xbee);
2052:   }
2053:
2054:   if (con->waitforACK &&
2055:       ((con->type == xbee_16bitData) ||
2056:        (con->type == xbee_64bitData))) {
2057:     con->ACKstatus = 255; /* waiting */
2058:     xbee_log("Waiting for ACK/NAK response...");
2059:     xbee_sem_wait(con->waitforACKsem);
2060:     switch (con->ACKstatus) {
2061:       case 0: xbee_log("ACK recieved!"); break;
2062:       case 1: xbee_log("NAK recieved..."); break;
2063:       case 2: xbee_log("CCA failure..."); break;
2064:       case 3: xbee_log("Purged..."); break;
2065:       case 255: default: xbee_log("Timeout...");
2066:     }
2067:     if (con->ACKstatus) retval = 1; /* error */
2068:   }
2069:
2070:   /* unlock connection mutex */
2071:   xbee_mutex_unlock(con->Txmutex);
2072:
2073:   /* free the packet */
2074:   Xfree(pkt);
2075:
2076:   return retval;
2077: }
2078:
2079: /* ###################################################################
2080:    xbee_make_pkt - INTERNAL
2081:    adds delimiter field
2082:    calculates length and checksum
2083:    escapes bytes */
2084: static t_data *xbee_make_pkt(xbee_hnd xbee, unsigned char *data, int length) {
2085:   t_data *pkt;
2086:   unsigned int l, i, o, t, x, m;
2087:   char d = 0;
2088:
2089:   /* check the data given isnt too long
2090:      100 bytes maximum payload + 12 bytes header information */
2091:   if (length > 100 + 12) return NULL;
2092:
2093:   /* calculate the length of the whole packet
2094:      start, length (MSB), length (LSB), DATA, checksum */
2095:   l = 3 + length + 1;
2096:
2097:   /* prepare memory */
2098:   pkt = Xcalloc(sizeof(t_data));
2099:
2100:   /* put start byte on */
2101:   pkt->data[0] = 0x7E;
2102:
2103:   /* copy data into packet */
2104:   for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
2105:     /* if: its time for the checksum */
2106:     if (i == length) d = M8((0xFF - M8(t)));
2107:     /* if: its time for the high length byte */
2108:     else if (m == 1) d = M8(length >> 8);
2109:     /* if: its time for the low length byte */
2110:     else if (m == 2) d = M8(length);
2111:     /* if: its time for the normal data */
2112:     else if (m > 2) d = data[i];
2113:
2114:     x = 0;
2115:     /* check for any escapes needed */
2116:     if ((d == 0x11) || /* XON */
2117:         (d == 0x13) || /* XOFF */
2118:         (d == 0x7D) || /* Escape */
2119:         (d == 0x7E)) { /* Frame Delimiter */
2120:       l++;
2121:       pkt->data[o++] = 0x7D;
2122:       x = 1;
2123:     }
2124:
2125:     /* move data in */
```

```
2126:     pkt->data[o] = ((!x)?d:d^0x20);
2127:     if (m > 2) {
2128:       i++;
2129:       t += d;
2130:     }
2131:   }
2132:
2133:   /* remember the length */
2134:   pkt->length = l;
2135:
2136:   return pkt;
2137: }
```