Robotech 2021 - XBee

Généré par Doxygen 1.9.3

Chapitre 1

Index hiérarchique

1.1 Hiérarchie des classes

Cette liste d'héritage est classée approximativement par ordre alphabétique :

Chapitre 2

Index des classes

2.1 Liste des classes

Liste des classes, structures, unions et interfaces avec une brève description :

Chapitre 3

Index des fichiers

3.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

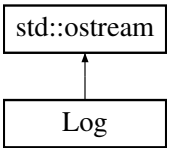
Chapitre 4

Documentation des classes

4.1 Référence de la classe Log

`#include <loglib.h>`

Graphe d'héritage de Log:



Fonctions membres publiques

- Log (std::string nom)
- int save (int data)

Attributs publics

- std::string name

Attributs privés

- std::stringstream ss

Amis

- template<typename T >
  Log & operator<< (Log &log, const T &classObj)
- Log & operator<< (Log &log, const Mendl &data)

4.1.1 Description détaillée

Définition à la ligne 18 du fichier loglib.h.

4.1.2 Documentation des constructeurs et destructeur

4.1.2.1 Log()

```
Log::Log (
            std::string nom )
```

Définition à la ligne 14 du fichier loglib.cpp.

```
00014                   {
00015     name = nom;
00016     stringstream ss;
00017 }
```

4.1.3 Documentation des fonctions membres

4.1.3.1 save()

```
int Log::save (
            int data )
```

4.1.4 Documentation des fonctions amies et associées

4.1.4.1 operator<< [1/2]

```
Log & operator<< (
            Log & log,
            const Mendl & data )  [friend]
```

Définition à la ligne 20 du fichier loglib.cpp.

```
00020                                       {
00021     time_t now = time(0);
00022     tm *ltm = localtime(&now);
00023     cout « endl;
00024     stringstream cmd;
00025     cmd « "echo \"" « "["«ltm->tm_hour « ":" « ltm->tm_min « ":" « ltm->tm_sec « " - "« log.name «"] "
      «log.ss.str()«"\" » log.log";
00026     log.ss.str("");
00027     system(stringToChar(cmd.str()));
00028
00029     return log;
00030
00031 }
```

4.1.4.2 operator<< [2/2]

```
template<typename T >
Log & operator<< (
            Log & log,
            const T & classObj )  [friend]
```

Définition à la ligne 34 du fichier loglib.h.

```
00035 {
00036     log.ss « data;
00037     std::cout « data;
00038     return log;
00039 }
```

4.1.5 Documentation des données membres

4.1.5.1 name

```
std::string Log::name
```

Définition à la ligne 23 du fichier loglib.h.

4.1.5.2 ss

```
std::stringstream Log::ss  [private]
```

Définition à la ligne 20 du fichier loglib.h.

La documentation de cette classe a été générée à partir du fichier suivant :

- loglib.h
- loglib.cpp

4.2 Référence de la structure Mendl

```
#include <loglib.h>
```

4.2.1 Description détaillée

Définition à la ligne 10 du fichier loglib.h.

La documentation de cette structure a été générée à partir du fichier suivant :

- loglib.h

4.3 Référence de la classe serialib

This class is used for communication over a serial device.

`#include <serialib.h>`

Fonctions membres publiques

- serialib ()
  Constructor of the class serialib.
- ~serialib ()
  Destructor of the class serialib. It close the connection.
- int openDevice (const char *Device, const unsigned int Bauds, SerialDataBits Databits=SERIAL_DATABITS_8, SerialParity Parity=SERIAL_PARITY_NONE, SerialStopBits Stopbits=SERIAL_STOPBITS_1)
  Open the serial port.
- bool isDeviceOpen ()
- void closeDevice ()
  Close the connection with the current device.
- char writeChar (char)
  Write a char on the current serial port.
- char readChar (char *pByte, const unsigned int timeOut_ms=0)
  Wait for a byte from the serial device and return the data read.
- char writeString (const char *String)
  Write a string on the current serial port.
- int readString (char *receivedString, char finalChar, unsigned int maxNbBytes, const unsigned int timeOut_ms=0)
  Read a string from the serial device (with timeout)
- char writeBytes (const void *Buffer, const unsigned int NbBytes)
  Write an array of data on the current serial port.
- int readBytes (void *buffer, unsigned int maxNbBytes, const unsigned int timeOut_ms=0, unsigned int sleepDuration_us=100)
  Read an array of bytes from the serial device (with timeout)
- char flushReceiver ()
  Empty receiver buffer.
- int available ()
  Return the number of bytes in the received buffer (UNIX only)
- bool DTR (bool status)
  Set or unset the bit DTR (pin 4) DTR stands for Data Terminal Ready Convenience method :This method calls setDTR and clearDTR.
- bool setDTR ()
  Set the bit DTR (pin 4) DTR stands for Data Terminal Ready.
- bool clearDTR ()
  Clear the bit DTR (pin 4) DTR stands for Data Terminal Ready.
- bool RTS (bool status)
  Set or unset the bit RTS (pin 7) RTS stands for Data Termina Ready Convenience method :This method calls setDTR and clearDTR.
- bool setRTS ()
  Set the bit RTS (pin 7) RTS stands for Data Terminal Ready.
- bool clearRTS ()
  Clear the bit RTS (pin 7) RTS stands for Data Terminal Ready.
- bool isRI ()
  Get the RING's status (pin 9) Ring Indicator.
- bool isDCD ()
  Get the DCD's status (pin 1) CDC stands for Data Carrier Detect.
- bool isCTS ()
  Get the CTS's status (pin 8) CTS stands for Clear To Send.
- bool isDSR ()
  Get the DSR's status (pin 6) DSR stands for Data Set Ready.
- bool isRTS ()
  Get the RTS's status (pin 7) RTS stands for Request To Send May behave abnormally on Windows.
- bool isDTR ()
  Get the DTR's status (pin 4) DTR stands for Data Terminal Ready May behave abnormally on Windows.

Fonctions membres privées

— int readStringNoTimeOut (char *String, char FinalChar, unsigned int MaxNbBytes)
Read a string from the serial device (without TimeOut)

Attributs privés

— bool currentStateRTS
— bool currentStateDTR

4.3.1 Description détaillée

This class is used for communication over a serial device.

Définition à la ligne 92 du fichier serialib.h.

4.3.2 Documentation des constructeurs et destructeur

4.3.2.1 serialib()

```
serialib::serialib ( )
```

Constructor of the class serialib.

Définition à la ligne 30 du fichier serialib.cpp.

```
00031 {
00032 #if defined (_WIN32) || defined( _WIN64)
00033     // Set default value for RTS and DTR (Windows only)
00034     currentStateRTS=true;
00035     currentStateDTR=true;
00036     hSerial = INVALID_HANDLE_VALUE;
00037 #endif
00038 #if defined (__linux__) || defined(__APPLE__)
00039     fd = -1;
00040 #endif
00041 }
```

4.3.2.2 ~serialib()

```
serialib::~serialib ( )
```

Destructor of the class serialib. It close the connection.

Définition à la ligne 48 du fichier serialib.cpp.

```
00049 {
00050     closeDevice();
00051 }
```

4.3.3 Documentation des fonctions membres

4.3.3.1 available()

```
int serialib::available ( )
```

Return the number of bytes in the received buffer (UNIX only)

Renvoie

The number of bytes received by the serial provider but not yet read.

Définition à la ligne 702 du fichier serialib.cpp.

```
00703 {
00704 #if defined (_WIN32) || defined(_WIN64)
00705     // Device errors
00706     DWORD commErrors;
00707     // Device status
00708     COMSTAT commStatus;
00709     // Read status
00710     ClearCommError(hSerial, &commErrors, &commStatus);
00711     // Return the number of pending bytes
00712     return commStatus.cbInQue;
00713 #endif
00714 #if defined (__linux__) || defined(__APPLE__)
00715     int nBytes=0;
00716     // Return number of pending bytes in the receiver
00717     ioctl(fd, FIONREAD, &nBytes);
00718     return nBytes;
00719 #endif
00720
00721 }
```

4.3.3.2 clearDTR()

```
bool serialib::clearDTR ( )
```

Clear the bit DTR (pin 4) DTR stands for Data Terminal Ready.

Renvoie

If the function fails, the return value is false If the function succeeds, the return value is true.

Définition à la ligne 777 du fichier serialib.cpp.

```
00778 {
00779 #if defined (_WIN32) || defined(_WIN64)
00780     // Clear DTR
00781     currentStateDTR=true;
00782     return EscapeCommFunction(hSerial,CLRDTR);
00783 #endif
00784 #if defined (__linux__) || defined(__APPLE__)
00785     // Clear DTR
00786     int status_DTR=0;
00787     ioctl(fd, TIOCMGET, &status_DTR);
00788     status_DTR &= ~TIOCM_DTR;
00789     ioctl(fd, TIOCMSET, &status_DTR);
00790     return true;
00791 #endif
00792 }
```

4.3.3.3 clearRTS()

```
bool serialib::clearRTS ( )
```

Clear the bit RTS (pin 7) RTS stands for Data Terminal Ready.

Renvoie

If the function fails, the return value is false If the function succeeds, the return value is true.

Définition à la ligne 847 du fichier serialib.cpp.

```
00848 {
00849 #if defined (_WIN32) || defined(_WIN64)
00850     // Clear RTS
00851     currentStateRTS=false;
00852     return EscapeCommFunction(hSerial,CLRRTS);
00853 #endif
00854 #if defined (__linux__) || defined(__APPLE__)
00855     // Clear RTS
00856     int status_RTS=0;
00857     ioctl(fd, TIOCMGET, &status_RTS);
00858     status_RTS &= ~TIOCM_RTS;
00859     ioctl(fd, TIOCMSET, &status_RTS);
00860     return true;
00861 #endif
00862 }
```

4.3.3.4 closeDevice()

```
void serialib::closeDevice ( )
```

Close the connection with the current device.

Définition à la ligne 317 du fichier serialib.cpp.

```
00318 {
00319 #if defined (_WIN32) || defined( _WIN64)
00320     CloseHandle(hSerial);
00321     hSerial = INVALID_HANDLE_VALUE;
00322 #endif
00323 #if defined (__linux__) || defined(__APPLE__)
00324     close (fd);
00325     fd = -1;
00326 #endif
00327 }
```

4.3.3.5 DTR()

```
bool serialib::DTR (
            bool status )
```

Set or unset the bit DTR (pin 4) DTR stands for Data Terminal Ready Convenience method :This method calls setDTR and clearDTR.

Paramètres

status	= true set DTR status = false unset DTR

Renvoie

If the function fails, the return value is false If the function succeeds, the return value is true.

Définition à la ligne 737 du fichier serialib.cpp.

```
00738 {
00739     if (status)
00740         // Set DTR
00741         return this->setDTR();
00742     else
00743         // Unset DTR
00744         return this->clearDTR();
00745 }
```

4.3.3.6 flushReceiver()

```
char serialib::flushReceiver ( )
```

Empty receiver buffer.

Renvoie

If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

Définition à la ligne 683 du fichier serialib.cpp.

```
00684 {
00685 #if defined (_WIN32) || defined(_WIN64)
00686     // Purge receiver
00687     return PurgeComm (hSerial, PURGE_RXCLEAR);
00688 #endif
00689 #if defined (__linux__) || defined(__APPLE__)
00690     // Purge receiver
00691     tcflush(fd,TCIFLUSH);
00692     return true;
00693 #endif
00694 }
```

4.3.3.7 isCTS()

```
bool serialib::isCTS ( )
```

Get the CTS's status (pin 8) CTS stands for Clear To Send.

Renvoie

Return true if CTS is set otherwise false

Définition à la ligne 872 du fichier serialib.cpp.

```
00873 {
00874 #if defined (_WIN32) || defined(_WIN64)
00875     DWORD modemStat;
00876     GetCommModemStatus(hSerial, &modemStat);
00877     return modemStat & MS_CTS_ON;
00878 #endif
00879 #if defined (__linux__) || defined(__APPLE__)
00880     int status=0;
00881     //Get the current status of the CTS bit
00882     ioctl(fd, TIOCMGET, &status);
00883     return status & TIOCM_CTS;
00884 #endif
00885 }
```

4.3.3.8 isDCD()

```
bool serialib::isDCD ( )
```

Get the DCD's status (pin 1) CDC stands for Data Carrier Detect.

Renvoie

true if DCD is set
false otherwise

Définition à la ligne 920 du fichier serialib.cpp.

```
00921 {
00922 #if defined (_WIN32) || defined(_WIN64)
00923     DWORD modemStat;
00924     GetCommModemStatus(hSerial, &modemStat);
00925     return modemStat & MS_RLSD_ON;
00926 #endif
00927 #if defined (__linux__) || defined(__APPLE__)
00928     int status=0;
00929     //Get the current status of the DCD bit
00930     ioctl(fd, TIOCMGET, &status);
00931     return status & TIOCM_CAR;
00932 #endif
00933 }
```

4.3.3.9 isDeviceOpen()

```
bool serialib::isDeviceOpen ( )
```

Définition à la ligne 304 du fichier serialib.cpp.

```
00305 {
00306 #if defined (_WIN32) || defined( _WIN64)
00307     return hSerial != INVALID_HANDLE_VALUE;
00308 #endif
00309 #if defined (__linux__) || defined(__APPLE__)
00310     return fd >= 0;
00311 #endif
00312 }
```

4.3.3.10 isDSR()

```
bool serialib::isDSR ( )
```

Get the DSR's status (pin 6) DSR stands for Data Set Ready.

Renvoie

Return true if DTR is set otherwise false

Définition à la ligne 894 du fichier serialib.cpp.

```
00895 {
00896 #if defined (_WIN32) || defined(_WIN64)
00897     DWORD modemStat;
00898     GetCommModemStatus(hSerial, &modemStat);
00899     return modemStat & MS_DSR_ON;
00900 #endif
00901 #if defined (__linux__) || defined(__APPLE__)
00902     int status=0;
00903     //Get the current status of the DSR bit
00904     ioctl(fd, TIOCMGET, &status);
00905     return status & TIOCM_DSR;
00906 #endif
00907 }
```

4.3.3.11 isDTR()

```
bool serialib::isDTR ( )
```

Get the DTR's status (pin 4) DTR stands for Data Terminal Ready May behave abnormally on Windows.

Renvoie

Return true if CTS is set otherwise false

Définition à la ligne 963 du fichier serialib.cpp.

```
00964 {
00965 #if defined (_WIN32) || defined( _WIN64)
00966     return currentStateDTR;
00967 #endif
00968 #if defined (__linux__) || defined(__APPLE__)
00969     int status=0;
00970     //Get the current status of the DTR bit
00971     ioctl(fd, TIOCMGET, &status);
00972     return status & TIOCM_DTR  ;
00973 #endif
00974 }
```

4.3.3.12 isRI()

```
bool serialib::isRI ( )
```

Get the RING's status (pin 9) Ring Indicator.

Renvoie

Return true if RING is set otherwise false

Définition à la ligne 941 du fichier serialib.cpp.

```
00942 {
00943 #if defined (_WIN32) || defined(_WIN64)
00944     DWORD modemStat;
00945     GetCommModemStatus(hSerial, &modemStat);
00946     return modemStat & MS_RING_ON;
00947 #endif
00948 #if defined (__linux__) || defined(__APPLE__)
00949     int status=0;
00950     //Get the current status of the RING bit
00951     ioctl(fd, TIOCMGET, &status);
00952     return status & TIOCM_RNG;
00953 #endif
00954 }
```

4.3.3.13 isRTS()

```
bool serialib::isRTS ( )
```

Get the RTS's status (pin 7) RTS stands for Request To Send May behave abnormally on Windows.

Renvoie

Return true if RTS is set otherwise false

Définition à la ligne 984 du fichier serialib.cpp.

```
00985 {
00986 #if defined (_WIN32) || defined(_WIN64)
00987     return currentStateRTS;
00988 #endif
00989 #if defined (__linux__) || defined(__APPLE__)
00990     int status=0;
00991     //Get the current status of the CTS bit
00992     ioctl(fd, TIOCMGET, &status);
00993     return status & TIOCM_RTS;
00994 #endif
00995 }
```

4.3.3.14 openDevice()

```
int serialib::openDevice (
            const char * Device,
            const unsigned int Bauds,
            SerialDataBits Databits = SERIAL_DATABITS_8,
            SerialParity Parity = SERIAL_PARITY_NONE,
            SerialStopBits Stopbits = SERIAL_STOPBITS_1 )
```

Open the serial port.

Paramètres

Device	: Port name (COM1, COM2, ... for Windows) or (/dev/ttyS0, /dev/ttyACM0, /dev/ttyUSB0 ... for linux)
Bauds	: Baud rate of the serial port. \n Supported baud rate for Windows : - 110 - 300 - 600 - 1200 - 2400 - 4800 - 9600 - 14400 - 19200 - 38400 - 56000 - 57600 - 115200 - 128000 - 256000 \n Supported baud rate for Linux :\n - 110 - 300 - 600 - 1200 - 2400 - 4800 - 9600 - 19200 - 38400 - 57600 - 115200
Databits	: Number of data bits in one UART transmission. \n Supported values: \n - SERIAL_DATABITS_5 (5) - SERIAL_DATABITS_6 (6) - SERIAL_DATABITS_7 (7) - SERIAL_DATABITS_8 (8) - SERIAL_DATABITS_16 (16) (not supported on Unix)
Parity	Parity type \n Supported values: \n - SERIAL_PARITY_NONE (N) - SERIAL_PARITY_EVEN (E) - SERIAL_PARITY_ODD (O) - SERIAL_PARITY_MARK (MARK) (not supported on Unix) - SERIAL_PARITY_SPACE (SPACE) (not supported on Unix)
Stopbit	Number of stop bits \n Supported values: - SERIAL_STOPBITS_1 (1) - SERIAL_STOPBITS_1_5 (1.5) (not supported on Unix) - SERIAL_STOPBITS_2 (2)

Renvoie

500 success
-501 device not found
-502 error while opening the device
-503 error while getting port parameters
-504 Speed (Bauds) not recognized
-505 error while writing port parameters
-506 error while writing timeout parameters
-507 Databits not recognized
-508 Stopbits not recognized
-509 Parity not recognized

Définition à la ligne 129 du fichier serialib.cpp.

```
00132                                                    {
00133 #if defined (_WIN32) || defined( _WIN64)
00134     // Open serial port
00135     hSerial = CreateFileA(Device,GENERIC_READ |
       GENERIC_WRITE,0,0,OPEN_EXISTING,/*FILE_ATTRIBUTE_NORMAL*/0,0);
00136     if(hSerial==INVALID_HANDLE_VALUE) {
00137         if(GetLastError()==ERROR_FILE_NOT_FOUND)
00138             return XB_SER_E_NOT_FOUND; // Device not found
00139
00140         // Error while opening the device
00141         return XB_SER_E_OPEN;
00142     }
00143
00144     // Set parameters
00145
00146     // Structure for the port parameters
00147     DCB dcbSerialParams;
00148     dcbSerialParams.DCBlength=sizeof(dcbSerialParams);
00149
00150     // Get the port parameters
00151     if (!GetCommState(hSerial, &dcbSerialParams)) return XB_SER_E_PARAM;
00152
00153     // Set the speed (Bauds)
00154     switch (Bauds)
00155     {
00156     case 110  :     dcbSerialParams.BaudRate=CBR_110; break;
00157     case 300  :     dcbSerialParams.BaudRate=CBR_300; break;
00158     case 600  :     dcbSerialParams.BaudRate=CBR_600; break;
00159     case 1200 :     dcbSerialParams.BaudRate=CBR_1200; break;
00160     case 2400 :     dcbSerialParams.BaudRate=CBR_2400; break;
00161     case 4800 :     dcbSerialParams.BaudRate=CBR_4800; break;
00162     case 9600 :     dcbSerialParams.BaudRate=CBR_9600; break;
00163     case 14400 :    dcbSerialParams.BaudRate=CBR_14400; break;
00164     case 19200 :    dcbSerialParams.BaudRate=CBR_19200; break;
00165     case 38400 :    dcbSerialParams.BaudRate=CBR_38400; break;
00166     case 56000 :    dcbSerialParams.BaudRate=CBR_56000; break;
00167     case 57600 :    dcbSerialParams.BaudRate=CBR_57600; break;
00168     case 115200 :   dcbSerialParams.BaudRate=CBR_115200; break;
00169     case 128000 :   dcbSerialParams.BaudRate=CBR_128000; break;
00170     case 256000 :   dcbSerialParams.BaudRate=CBR_256000; break;
00171     default : return XB_SER_E_UKN_BAUDRATE;
00172     }
00173     //select data size
00174     BYTE bytesize = 0;
00175     switch(Databits) {
00176         case SERIAL_DATABITS_5: bytesize = 5; break;
00177         case SERIAL_DATABITS_6: bytesize = 6; break;
00178         case SERIAL_DATABITS_7: bytesize = 7; break;
00179         case SERIAL_DATABITS_8: bytesize = 8; break;
00180         case SERIAL_DATABITS_16: bytesize = 16; break;
00181         default: return XB_SER_E_UKN_DATABITS;
00182     }
00183     BYTE stopBits = 0;
00184     switch(Stopbits) {
00185         case SERIAL_STOPBITS_1: stopBits = ONESTOPBIT; break;
00186         case SERIAL_STOPBITS_1_5: stopBits = ONE5STOPBITS; break;
00187         case SERIAL_STOPBITS_2: stopBits = TWOSTOPBITS; break;
00188         default: return XB_SER_E_UKN_STOPBITS;
00189     }
00190     BYTE parity = 0;
00191     switch(Parity) {
00192         case SERIAL_PARITY_NONE: parity = NOPARITY; break;
00193         case SERIAL_PARITY_EVEN: parity = EVENPARITY; break;
00194         case SERIAL_PARITY_ODD: parity = ODDPARITY; break;
00195         case SERIAL_PARITY_MARK: parity = MARKPARITY; break;
```

```
00196         case SERIAL_PARITY_SPACE: parity = SPACEPARITY; break;
00197         default: return XB_SER_E_UKN_PARITY;
00198     }
00199     // configure byte size
00200     dcbSerialParams.ByteSize = bytesize;
00201     // configure stop bits
00202     dcbSerialParams.StopBits = stopBits;
00203     // configure parity
00204     dcbSerialParams.Parity = parity;
00205
00206     // Write the parameters
00207     if(!SetCommState(hSerial, &dcbSerialParams)) return XB_SER_E_CONFIG;
00208
00209     // Set TimeOut
00210
00211     // Set the Timeout parameters
00212     timeouts.ReadIntervalTimeout=0;
00213     // No TimeOut
00214     timeouts.ReadTotalTimeoutConstant=MAXDWORD;
00215     timeouts.ReadTotalTimeoutMultiplier=0;
00216     timeouts.WriteTotalTimeoutConstant=MAXDWORD;
00217     timeouts.WriteTotalTimeoutMultiplier=0;
00218
00219     // Write the parameters
00220     if(!SetCommTimeouts(hSerial, &timeouts)) return XB_SER_E_TIMOUT;
00221
00222     // Opening successfull
00223     return 1;
00224 #endif
00225 #if defined (__linux__) || defined(__APPLE__)
00226     // Structure with the device's options
00227     struct termios options;
00228
00229
00230     // Open device
00231     fd = open(Device, O_RDWR | O_NOCTTY | O_NDELAY);
00232     // If the device is not open, return -1
00233     if (fd == -1) return XB_SER_E_OPEN;
00234     // Open the device in nonblocking mode
00235     fcntl(fd, F_SETFL, FNDELAY);
00236
00237
00238     // Get the current options of the port
00239     tcgetattr(fd, &options);
00240     // Clear all the options
00241     bzero(&options, sizeof(options));
00242
00243     // Prepare speed (Bauds)
00244     speed_t         Speed;
00245     switch (Bauds)
00246     {
00247     case 110  :     Speed=B110; break;
00248     case 300  :     Speed=B300; break;
00249     case 600  :     Speed=B600; break;
00250     case 1200 :     Speed=B1200; break;
00251     case 2400 :     Speed=B2400; break;
00252     case 4800 :     Speed=B4800; break;
00253     case 9600 :     Speed=B9600; break;
00254     case 19200 :    Speed=B19200; break;
00255     case 38400 :    Speed=B38400; break;
00256     case 57600 :    Speed=B57600; break;
00257     case 115200 :   Speed=B115200; break;
00258     default : return XB_SER_E_UKN_BAUDRATE;
00259     }
00260     int databits_flag = 0;
00261     switch(Databits) {
00262         case SERIAL_DATABITS_5: databits_flag = CS5; break;
00263         case SERIAL_DATABITS_6: databits_flag = CS6; break;
00264         case SERIAL_DATABITS_7: databits_flag = CS7; break;
00265         case SERIAL_DATABITS_8: databits_flag = CS8; break;
00266         //16 bits and everything else not supported
00267         default: return XB_SER_E_UKN_DATABITS;
00268     }
00269     int stopbits_flag = 0;
00270     switch(Stopbits) {
00271         case SERIAL_STOPBITS_1: stopbits_flag = 0; break;
00272         case SERIAL_STOPBITS_2: stopbits_flag = CSTOPB; break;
00273         //1.5 stopbits and everything else not supported
00274         default: return XB_SER_E_UKN_STOPBITS;
00275     }
00276     int parity_flag = 0;
00277     switch(Parity) {
00278         case SERIAL_PARITY_NONE: parity_flag = 0; break;
00279         case SERIAL_PARITY_EVEN: parity_flag = PARENB; break;
00280         case SERIAL_PARITY_ODD: parity_flag = (PARENB | PARODD); break;
00281         //mark and space parity not supported
00282         default: return XB_SER_E_UKN_PARITY;
```

```
00283     }
00284
00285     // Set the baud rate
00286     cfsetispeed(&options, Speed);
00287     cfsetospeed(&options, Speed);
00288     // Configure the device : data bits, stop bits, parity, no control flow
00289     // Ignore modem control lines (CLOCAL) and Enable receiver (CREAD)
00290     options.c_cflag |= ( CLOCAL | CREAD | databits_flag | parity_flag | stopbits_flag);
00291     options.c_iflag |= ( IGNPAR | IGNBRK );
00292     // Timer unused
00293     options.c_cc[VTIME]=0;
00294     // At least on character before satisfy reading
00295     options.c_cc[VMIN]=0;
00296     // Activate the settings
00297     tcsetattr(fd, TCSANOW, &options);
00298     // Success
00299     return (XB_SER_E_SUCCESS);
00300 #endif
00301
00302 }
```

4.3.3.15 readBytes()

```
int serialib::readBytes (
            void * buffer,
            unsigned int maxNbBytes,
            const unsigned int timeOut_ms = 0,
            unsigned int sleepDuration_us = 100 )
```

Read an array of bytes from the serial device (with timeout)

Paramètres

buffer	: array of bytes read from the serial device
maxNbBytes	: maximum allowed number of bytes read
timeOut_ms	: delay of timeout before giving up the reading
sleepDuration_us	: delay of CPU relaxing in microseconds (Linux only) In the reading loop, a sleep can be performed after each reading This allows CPU to perform other tasks

Renvoie

>=0 return the number of bytes read before timeout or requested data is completed
-1 error while setting the Timeout
-2 error while reading the byte

Définition à la ligne 615 du fichier serialib.cpp.

```
00616 {
00617 #if defined (_WIN32) || defined(_WIN64)
00618     // Avoid warning while compiling
00619     UNUSED(sleepDuration_us);
00620
00621     // Number of bytes read
00622     DWORD dwBytesRead = 0;
00623
00624     // Set the TimeOut
00625     timeouts.ReadTotalTimeoutConstant=(DWORD)timeOut_ms;
00626
00627     // Write the parameters and return -1 if an error occrured
00628     if(!SetCommTimeouts(hSerial, &timeouts)) return -1;
00629
00630
00631     // Read the bytes from the serial device, return -2 if an error occured
00632     if(!ReadFile(hSerial,buffer,(DWORD)maxNbBytes,&dwBytesRead, NULL))  return -2;
00633
00634     // Return the byte read
```

```
00635     return dwBytesRead;
00636 #endif
00637 #if defined (__linux__) || defined(__APPLE__)
00638     // Timer used for timeout
00639     timeOut           timer;
00640     // Initialise the timer
00641     timer.initTimer();
00642     unsigned int      NbByteRead=0;
00643     // While Timeout is not reached
00644     while (timer.elapsedTime_ms()<timeOut_ms || timeOut_ms==0)
00645     {
00646         // Compute the position of the current byte
00647         unsigned char* Ptr=(unsigned char*)buffer+NbByteRead;
00648         // Try to read a byte on the device
00649         int Ret=read(fd,(void*)Ptr,maxNbBytes-NbByteRead);
00650         // Error while reading
00651         if (Ret==-1) return -2;
00652
00653         // One or several byte(s) has been read on the device
00654         if (Ret>0)
00655         {
00656             // Increase the number of read bytes
00657             NbByteRead+=Ret;
00658             // Success : bytes has been read
00659             if (NbByteRead>=maxNbBytes)
00660                 return NbByteRead;
00661         }
00662         // Suspend the loop to avoid charging the CPU
00663         usleep (sleepDuration_us);
00664     }
00665     // Timeout reached, return the number of bytes read
00666     return NbByteRead;
00667 #endif
00668 }
```

4.3.3.16 readChar()

```
char serialib::readChar (
            char * pByte,
            const unsigned int timeOut_ms = 0 )
```

Wait for a byte from the serial device and return the data read.

Paramètres

pByte	: data read on the serial device
timeOut_ms	: delay of timeout before giving up the reading If set to zero, timeout is disable (Optional)

Renvoie

1 success

0 Timeout reached

-1 error while setting the Timeout

-2 error while reading the byte

Définition à la ligne 441 du fichier serialib.cpp.

```
00442 {
00443 #if defined (_WIN32) || defined(_WIN64)
00444     // Number of bytes read
00445     DWORD dwBytesRead = 0;
00446
00447     // Set the TimeOut
00448     timeouts.ReadTotalTimeoutConstant=timeOut_ms;
00449
00450     // Write the parameters, return -1 if an error occured
00451     if(!SetCommTimeouts(hSerial, &timeouts)) return -1;
00452
```

```
00453     // Read the byte, return -2 if an error occured
00454     if(!ReadFile(hSerial,pByte, 1, &dwBytesRead, NULL)) return -2;
00455
00456     // Return 0 if the timeout is reached
00457     if (dwBytesRead==0) return 0;
00458
00459     // The byte is read
00460     return 1;
00461 #endif
00462 #if defined (__linux__) || defined(__APPLE__)
00463     // Timer used for timeout
00464     timeOut         timer;
00465     // Initialise the timer
00466     timer.initTimer();
00467     // While Timeout is not reached
00468     while (timer.elapsedTime_ms()<timeOut_ms || timeOut_ms==0)
00469     {
00470         // Try to read a byte on the device
00471         switch (read(fd,pByte,1)) {
00472         case 1  : return 1; // Read successfull
00473         case -1 : return -2; // Error while reading
00474         }
00475     }
00476     return 0;
00477 #endif
00478 }
```

4.3.3.17 readString()

```
int serialib::readString (
              char * receivedString,
              char finalChar,
              unsigned int maxNbBytes,
              const unsigned int timeOut_ms = 0 )
```

Read a string from the serial device (with timeout)

Paramètres

receivedString	: string read on the serial device
finalChar	: final char of the string
maxNbBytes	: maximum allowed number of bytes read
timeOut_ms	: delay of timeout before giving up the reading (optional)

Renvoie

>0 success, return the number of bytes read
0 timeout is reached
-1 error while setting the Timeout
-2 error while reading the byte
-3 MaxNbBytes is reached

Définition à la ligne 541 du fichier serialib.cpp.

```
00542 {
00543     // Check if timeout is requested
00544     if (timeOut_ms==0) return readStringNoTimeOut(receivedString,finalChar,maxNbBytes);
00545
00546     // Number of bytes read
00547     unsigned int    nbBytes=0;
00548     // Character read on serial device
00549     char            charRead;
00550     // Timer used for timeout
00551     timeOut         timer;
00552     long int        timeOutParam;
```

```
00553
00554     // Initialize the timer (for timeout)
00555     timer.initTimer();
00556
00557     // While the buffer is not full
00558     while (nbBytes<maxNbBytes)
00559     {
00560         // Compute the TimeOut for the next call of ReadChar
00561         timeOutParam = timeOut_ms-timer.elapsedTime_ms();
00562
00563         // If there is time remaining
00564         if (timeOutParam>0)
00565         {
00566             // Wait for a byte on the serial link with the remaining time as timeout
00567             charRead=readChar(&receivedString[nbBytes],timeOutParam);
00568
00569             // If a byte has been received
00570             if (charRead==1)
00571             {
00572                 // Check if the character received is the final one
00573                 if (receivedString[nbBytes]==finalChar)
00574                 {
00575                     // Final character: add the end character 0
00576                     receivedString  [++nbBytes]=0;
00577                     // Return the number of bytes read
00578                     return nbBytes;
00579                 }
00580                 // This is not the final character, just increase the number of bytes read
00581                 nbBytes++;
00582             }
00583             // Check if an error occured during reading char
00584             // If an error occurend, return the error number
00585             if (charRead<0) return charRead;
00586         }
00587         // Check if timeout is reached
00588         if (timer.elapsedTime_ms()>timeOut_ms)
00589         {
00590             // Add the end caracter
00591             receivedString[nbBytes]=0;
00592             // Return 0 (timeout reached)
00593             return 0;
00594         }
00595     }
00596
00597     // Buffer is full : return -3
00598     return -3;
00599 }
```

4.3.3.18 readStringNoTimeOut()

```
int serialib::readStringNoTimeOut (
            char * receivedString,
            char finalChar,
            unsigned int maxNbBytes )  [private]
```

Read a string from the serial device (without TimeOut)

Paramètres

receivedString	: string read on the serial device
FinalChar	: final char of the string
MaxNbBytes	: maximum allowed number of bytes read

Renvoie

>0 success, return the number of bytes read
-1 error while setting the Timeout
-2 error while reading the byte
-3 MaxNbBytes is reached

Définition à la ligne 492 du fichier serialib.cpp.

```
00493 {
00494     // Number of characters read
00495     unsigned int    NbBytes=0;
00496     // Returned value from Read
00497     char            charRead;
00498
00499     // While the buffer is not full
00500     while (NbBytes<maxNbBytes)
00501     {
00502         // Read a character with the restant time
00503         charRead=readChar(&receivedString[NbBytes]);
00504
00505         // Check a character has been read
00506         if (charRead==1)
00507         {
00508             // Check if this is the final char
00509             if (receivedString[NbBytes]==finalChar)
00510             {
00511                 // This is the final char, add zero (end of string)
00512                 receivedString  [++NbBytes]=0;
00513                 // Return the number of bytes read
00514                 return NbBytes;
00515             }
00516
00517             // The character is not the final char, increase the number of bytes read
00518             NbBytes++;
00519         }
00520
00521         // An error occured while reading, return the error number
00522         if (charRead<0) return charRead;
00523     }
00524     // Buffer is full : return -3
00525     return -3;
00526 }
```

4.3.3.19 RTS()

```
bool serialib::RTS (
            bool status )
```

Set or unset the bit RTS (pin 7) RTS stands for Data Termina Ready Convenience method :This method calls setDTR and clearDTR.

Paramètres

status	= true set DTR status = false unset DTR

Renvoie

false if the function fails

true if the function succeeds

Définition à la ligne 805 du fichier serialib.cpp.

```
00806 {
00807     if (status)
00808         // Set RTS
00809         return this->setRTS();
00810     else
00811         // Unset RTS
00812         return this->clearRTS();
00813 }
```

4.3.3.20 setDTR()

```
bool serialib::setDTR ( )
```

Set the bit DTR (pin 4) DTR stands for Data Terminal Ready.

Renvoie

If the function fails, the return value is false If the function succeeds, the return value is true.

Définition à la ligne 754 du fichier serialib.cpp.

```
00755 {
00756 #if defined (_WIN32) || defined(_WIN64)
00757     // Set DTR
00758     currentStateDTR=true;
00759     return EscapeCommFunction(hSerial,SETDTR);
00760 #endif
00761 #if defined (__linux__) || defined(__APPLE__)
00762     // Set DTR
00763     int status_DTR=0;
00764     ioctl(fd, TIOCMGET, &status_DTR);
00765     status_DTR |= TIOCM_DTR;
00766     ioctl(fd, TIOCMSET, &status_DTR);
00767     return true;
00768 #endif
00769 }
```

4.3.3.21 setRTS()

```
bool serialib::setRTS ( )
```

Set the bit RTS (pin 7) RTS stands for Data Terminal Ready.

Renvoie

If the function fails, the return value is false If the function succeeds, the return value is true.

Définition à la ligne 822 du fichier serialib.cpp.

```
00823 {
00824 #if defined (_WIN32) || defined(_WIN64)
00825     // Set RTS
00826     currentStateRTS=false;
00827     return EscapeCommFunction(hSerial,SETRTS);
00828 #endif
00829 #if defined (__linux__) || defined(__APPLE__)
00830     // Set RTS
00831     int status_RTS=0;
00832     ioctl(fd, TIOCMGET, &status_RTS);
00833     status_RTS |= TIOCM_RTS;
00834     ioctl(fd, TIOCMSET, &status_RTS);
00835     return true;
00836 #endif
00837 }
```

4.3.3.22 writeBytes()

```
char serialib::writeBytes (
            const void * Buffer,
            const unsigned int NbBytes )
```

Write an array of data on the current serial port.

Paramètres

Buffer	: array of bytes to send on the port
NbBytes	: number of byte to send

Renvoie

1 success

-1 error while writting data

Définition à la ligne 409 du fichier serialib.cpp.

```
00410 {
00411 #if defined (_WIN32) || defined( _WIN64)
00412     // Number of bytes written
00413     DWORD dwBytesWritten;
00414     // Write data
00415     if(!WriteFile(hSerial, Buffer, NbBytes, &dwBytesWritten, NULL))
00416         // Error while writing, return -1
00417         return -1;
00418     // Write operation successfull
00419     return 1;
00420 #endif
00421 #if defined (__linux__) || defined(__APPLE__)
00422     // Write data
00423     if (write (fd,Buffer,NbBytes)!=(ssize_t)NbBytes) return -1;
00424     // Write operation successfull
00425     return 1;
00426 #endif
00427 }
```

4.3.3.23 writeChar()

```
char serialib::writeChar (
            char Byte )
```

Write a char on the current serial port.

Paramètres

Byte	: char to send on the port (must be terminated by '\0')

Renvoie

1 success

-1 error while writting data

Définition à la ligne 343 du fichier serialib.cpp.

```
00344 {
00345 #if defined (_WIN32) || defined( _WIN64)
00346     // Number of bytes written
00347     DWORD dwBytesWritten;
00348     // Write the char to the serial device
00349     // Return -1 if an error occured
00350     if(!WriteFile(hSerial,&Byte,1,&dwBytesWritten,NULL)) return -1;
00351     // Write operation successfull
00352     return 1;
00353 #endif
00354 #if defined (__linux__) || defined(__APPLE__)
00355     // Write the char
00356     if (write(fd,&Byte,1)!=1) return -1;
00357
00358     // Write operation successfull
```

```
00359     return 1;
00360 #endif
00361 }
```

4.3.3.24 writeString()

```
char serialib::writeString (
             const char * receivedString )
```

Write a string on the current serial port.

Paramètres

receivedString	: string to send on the port (must be terminated by '\0')

Renvoie

1 success

-1 error while writting data

Définition à la ligne 375 du fichier serialib.cpp.

```
00376 {
00377 #if defined (_WIN32) || defined( _WIN64)
00378     // Number of bytes written
00379     DWORD dwBytesWritten;
00380     // Write the string
00381     if(!WriteFile(hSerial,receivedString,strlen(receivedString),&dwBytesWritten,NULL))
00382         // Error while writing, return -1
00383         return -1;
00384     // Write operation successfull
00385     return 1;
00386 #endif
00387 #if defined (__linux__) || defined(__APPLE__)
00388     // Lenght of the string
00389     int Lenght=strlen(receivedString);
00390     // Write the string
00391     if (write(fd,receivedString,Lenght)!=Lenght) return -1;
00392     // Write operation successfull
00393     return 1;
00394 #endif
00395 }
```

4.3.4 Documentation des données membres

4.3.4.1 currentStateDTR

```
bool serialib::currentStateDTR  [private]
```

Définition à la ligne 221 du fichier serialib.h.

4.3.4.2 currentStateRTS

```
bool serialib::currentStateRTS  [private]
```

Définition à la ligne 220 du fichier serialib.h.

La documentation de cette classe a été générée à partir du fichier suivant :
- serialib.h
- serialib.cpp

4.4 Référence de la classe timeOut

This class can manage a timer which is used as a timeout.

```
#include <serialib.h>
```

Fonctions membres publiques

- timeOut ()
  Constructor of the class timeOut.
- void initTimer ()
  Initialise the timer. It writes the current time of the day in the structure PreviousTime.
- unsigned long int elapsedTime_ms ()
  Returns the time elapsed since initialization. It write the current time of the day in the structure CurrentTime. Then it returns the difference between CurrentTime and PreviousTime.

Attributs privés

- struct timeval previousTime

4.4.1 Description détaillée

This class can manage a timer which is used as a timeout.

Définition à la ligne 245 du fichier serialib.h.

4.4.2 Documentation des constructeurs et destructeur

4.4.2.1 timeOut()

```
timeOut::timeOut ( )
```

Constructor of the class timeOut.

Définition à la ligne 1011 du fichier serialib.cpp.
```
01012 {}
```

4.4.3 Documentation des fonctions membres

4.4.3.1 elapsedTime_ms()

```
unsigned long int timeOut::elapsedTime_ms ( )
```

Returns the time elapsed since initialization. It write the current time of the day in the structure CurrentTime. Then it returns the difference between CurrentTime and PreviousTime.

Renvoie

The number of microseconds elapsed since the functions InitTimer was called.

Définition à la ligne 1039 du fichier serialib.cpp.

```
01040 {
01041 #if defined (NO_POSIX_TIME)
01042     // Current time
01043     LARGE_INTEGER CurrentTime;
01044     // Number of ticks since last call
01045     int sec;
01046
01047     // Get current time
01048     QueryPerformanceCounter(&CurrentTime);
01049
01050     // Compute the number of ticks elapsed since last call
01051     sec=CurrentTime.QuadPart-previousTime;
01052
01053     // Return the elapsed time in milliseconds
01054     return sec/(counterFrequency/1000);
01055 #else
01056     // Current time
01057     struct timeval CurrentTime;
01058     // Number of seconds and microseconds since last call
01059     int sec,usec;
01060
01061     // Get current time
01062     gettimeofday(&CurrentTime, NULL);
01063
01064     // Compute the number of seconds and microseconds elapsed since last call
01065     sec=CurrentTime.tv_sec-previousTime.tv_sec;
01066     usec=CurrentTime.tv_usec-previousTime.tv_usec;
01067
01068     // If the previous usec is higher than the current one
01069     if (usec<0)
01070     {
01071         // Recompute the microseonds and substract one second
01072         usec=1000000-previousTime.tv_usec+CurrentTime.tv_usec;
01073         sec--;
01074     }
01075
01076     // Return the elapsed time in milliseconds
01077     return sec*1000+usec/1000;
01078 #endif
01079 }
```

4.4.3.2 initTimer()

```
void timeOut::initTimer ( )
```

Initialise the timer. It writes the current time of the day in the structure PreviousTime.

Définition à la ligne 1019 du fichier serialib.cpp.

```
01020 {
01021 #if defined (NO_POSIX_TIME)
01022     LARGE_INTEGER tmp;
01023     QueryPerformanceFrequency(&tmp);
01024     counterFrequency = tmp.QuadPart;
01025     // Used to store the previous time (for computing timeout)
01026     QueryPerformanceCounter(&tmp);
01027     previousTime = tmp.QuadPart;
01028 #else
01029     gettimeofday(&previousTime, NULL);
01030 #endif
01031 }
```

4.4.4 Documentation des données membres

4.4.4.1 previousTime

```
struct timeval timeOut::previousTime  [private]
```

Définition à la ligne 265 du fichier serialib.h.

La documentation de cette classe a été générée à partir du fichier suivant :

- serialib.h
- serialib.cpp

4.5 Référence de la structure XBee::Trame_t

Attributs publics

- int start_seq
- int adr_emetteur
- int adr_dest
- int id_trame_low
- int id_trame_high
- int nb_octets_msg
- int code_fct
- std::vector< int > param
- int crc_low
- int crc_high
- int end_seq

4.5.1 Description détaillée

Définition à la ligne 83 du fichier xbeelib.h.

4.5.2 Documentation des données membres

4.5.2.1 adr_dest

```
int XBee::Trame_t::adr_dest
```

Adresse du destinataire de la trame

Définition à la ligne 86 du fichier xbeelib.h.

4.5.2.2 adr_emetteur

```
int XBee::Trame_t::adr_emetteur
```

Adresse de l'émetteur de la trame

Définition à la ligne 85 du fichier xbeelib.h.

4.5.2.3 code_fct

```
int XBee::Trame_t::code_fct
```

Code fonction de la trame

Définition à la ligne 90 du fichier xbeelib.h.

4.5.2.4 crc_high

```
int XBee::Trame_t::crc_high
```

Bits de poids fort du CRC

Définition à la ligne 93 du fichier xbeelib.h.

4.5.2.5 crc_low

```
int XBee::Trame_t::crc_low
```

Bits de poids faible du CRC

Définition à la ligne 92 du fichier xbeelib.h.

4.5.2.6 end_seq

```
int XBee::Trame_t::end_seq
```

Caractère de fin de trame

Définition à la ligne 94 du fichier xbeelib.h.

4.5.2.7 id_trame_high

```
int XBee::Trame_t::id_trame_high
```

Bits de poids fort de l'ID de la trame

Définition à la ligne 88 du fichier xbeelib.h.

4.5.2.8 id_trame_low

```
int XBee::Trame_t::id_trame_low
```

Bits de poids faible de l'ID de la trame

Définition à la ligne 87 du fichier xbeelib.h.

4.5.2.9 nb_octets_msg

```
int XBee::Trame_t::nb_octets_msg
```

Nombre d'octets du champ data + code fonction

Définition à la ligne 89 du fichier xbeelib.h.

4.5.2.10 param

```
std::vector<int> XBee::Trame_t::param
```

Data de la trame

Définition à la ligne 91 du fichier xbeelib.h.

4.5.2.11 start_seq

```
int XBee::Trame_t::start_seq
```

Caractère de début de trame

Définition à la ligne 84 du fichier xbeelib.h.

La documentation de cette structure a été générée à partir du fichier suivant :

— xbeelib.h

4.6 Référence de la classe XBee

Cette classe est utilisée pour la communication entre un module XBee et une RaspberryPi et entre plusieurs modules XBee.

`#include <xbeelib.h>`

Classes

— struct Trame_t

Fonctions membres publiques

— XBee ()
Constructeur de la classe xbee.
— ∼XBee ()
Destructeur de la classe xbee.
— int openSerialConnection (int mode=0)
Nettoyage du buffer et ouverture de la connexion UART entre la RaspberryPi et le module XBee.
— void closeSerialConnection ()
Nettoyage du buffer et fermeture de la connexion UART entre la RaspberryPi et le module XBee.
— int checkATConfig ()
Vérification et paramétrage de la configuration par défaut pour le module XBee.
— bool readATResponse (const char ∗value=XB_AT_R_EMPTY, int mode=0)
Fonction permettant de lire la réponse à un envoi de commande AT au module XBee.
— bool sendATCommand (const char ∗command, const char ∗value, unsigned int mode=XB_AT_M_SET)
Fonction permettant d'envoyer en UART via le port série une commmande AT.
— bool writeATConfig ()
Fonction permettant d'écrire dans la mémoire flash du module XBee, les paramètres AT définis.
— int sendTrame (uint8_t ad_dest, uint8_t code_fct, char ∗data=0x00)
Fonction permettant d'envoyer une trame de message structurée via UART en XBee.
— void sendMsg (std::string msg)
Permet d'envoyer un mesage ASCII sans format particulier via XBee.
— void waitForATrame ()
Permet l'attente et la vérification régulée d'une trame en entrée dans le buffer du port Rx de la RaspberryPi et d'appeler la fonction de découpe des trames.
— void sendHeartbeat ()
Permet d'envoyer des demandes de battements de coeur au second robot afin de savoir s'il est toujours opérationnel.
— int isXbeeResponding ()
Permet de vérifier si un message envoyé a reçu une réponse.
— std::string charToString (char ∗message)
Permet de convertir une chaine de caractère standard C en objet de type string.
— char ∗ stringToChar (std::string chaine)
Permet de convertir un objet de type string en chaine de caractère standard C.
— void print (const std::vector< int > &v)
Fonction d'affichage des valeurs contenues dans un vecteur d'entiers.

Fonctions membres privées

- bool enterATMode ()
  Fonction permettant d'entrer dans le mode AT.
- bool exitATMode ()
  Fonction permettant de sortir du mode AT.
- bool discoverXbeeNetwork ()
  Recherche du module XBee distant de l'autre robot.
- bool isExpCorrect (int exp)
  Vérifie si une adresse d'expéditeur est correcte.
- bool isDestCorrect (int dest)
  Vérifie si une adresse de destination est correcte.
- bool isCodeFctCorrect (int code_fct)
  Vérifie si le code fonction donné est présent dans le fichier define.h.
- bool isTrameSizeCorrect (std::vector< int > trame)
  Vérifie si la taille de la trame est cohérente.
- int subTrame (std::vector< int > msg_recu)
  Découpe le résultat de la lecture du buffer en différentes trames avant le traitement.
- int processCodeFct (int code_fct, int exp)
  Interprète le code fonction issu d'une trame reçue.
- void afficherTrameRecue (Trame_t trame)
  Fonction d'affichage des données découpées d'une structure de type Trame.
- int processTrame (std::vector< int > trame)
  Découpe une trame reçue en fonction de ses paramètres et interprete son code fonction.
- std::vector< int > readBuffer ()
  Permet de lire l'intégralité du buffer Rx de la RaspberryPi.
- std::string readString ()
  Permet de lire l'intégralité du contenu du buffer Rx de la RaspberryPi et de le renvoyer sous forme d'objet string.
- int crc16 (int trame[ ], uint8_t taille)
  Fonction permettant de calculer le CRC16 Modbus de la trame XBee envoyée.
- bool isStartSeqCorrect (int value)
  Vérifie si le caractère de début de la trame correpond à celui attendu.
- bool isEndSeqCorrect (int value)
  Vérifie si le caractère de fin de la trame correpond à celui attendu.
- bool isCRCCorrect (uint8_t crc_low, uint8_t crc_high, int trame[ ], int trame_size)
  Vérifie si le CRC reçu est cohérent avec la trame reçue.
- void delay (unsigned int time)
  Fonction permettant de retarder l'exécution du code.
- std::vector< int > slice (const std::vector< int > &v, int a, int b)
  Fonction de traitement permettant d'extraire un sous-vecteur d'entiers d'un vecteur d'entiers.

Attributs privés

- int ID_TRAME = 0
- int MODE = 0
- std::vector< int > trames_envoyees = {}

4.6.1 Description détaillée

Cette classe est utilisée pour la communication entre un module XBee et une RaspberryPi et entre plusieurs modules XBee.

Définition à la ligne 25 du fichier xbeelib.h.

4.6.2 Documentation des constructeurs et destructeur

4.6.2.1 XBee()

```
XBee::XBee ( )
```

Constructeur de la classe xbee.

Définition à la ligne 21 du fichier xbeelib.cpp.

```
00021 { }
```

4.6.2.2 ~XBee()

```
XBee::~XBee ( )
```

Destructeur de la classe xbee.

Définition à la ligne 26 du fichier xbeelib.cpp.

```
00026 { }
```

4.6.3 Documentation des fonctions membres

4.6.3.1 afficherTrameRecue()

```
void XBee::afficherTrameRecue (
            Trame_t trame )  [private]
```

Fonction d'affichage des données découpées d'une structure de type Trame.

Définition à la ligne 847 du fichier xbeelib.cpp.

```
00847                                           {
00848     cout « hex « showbase;
00849     cout « "\t-> Start seq : " « trame.start_seq « endl;
00850     cout « "\t-> Emetteur : " « trame.adr_emetteur « endl;
00851     cout « "\t-> Destinataire : " « trame.adr_dest « endl;
00852     cout « "\t-> Id trame  : " « trame.id_trame_low « " " « trame.crc_high « endl;
00853     cout « "\t-> Taille msg : " « trame.nb_octets_msg - 3 « endl;
00854     cout « "\t-> Code fct : " « trame.code_fct « endl;
00855     cout « "\t-> Data : ";
00856     print(trame.param);
00857     cout « "\t-> CRC : " « trame.crc_low « " " « trame.crc_high « endl;
00858     cout « "\t-> End seq : " « trame.end_seq « endl;
00859 }
```

4.6.3.2 charToString()

```
string XBee::charToString (
            char * message )
```

Permet de convertir une chaine de caractère standard C en objet de type string.

Paramètres

message	: la chaine de caractère à convertir

Renvoie

chaine : l'objet de type string converti

Définition à la ligne 839 du fichier xbeelib.cpp.

```
00839                                       {
00840     string chaine = string(message);
00841     return chaine;
00842 }
```

4.6.3.3 checkATConfig()

```
int XBee::checkATConfig ( )
```

Vérification et paramétrage de la configuration par défaut pour le module XBee.

Renvoie

400 succès
-401 impossible d'entrer dans le mode AT
-402 impossible de configurer le mode API
-403 impossible de configurer le baudrate
-404 impossible de configurer le paramètre de chiffrement AES
-405 impossible de configurer la clé de chiffrement AES
-406 impossible de configurer le canal de découverte réseau
-407 impossible de configurer l'ID du réseau
-408 impossible de configurer le mode coordinateur
-409 impossible de configurer le nombre de bits de parité
-410 impossible de configurer l'adresse source 16bits
-411 impossible de configuer l'adresse de destination
-412 impossible de sortir du mode AT
-413 impossible d'écrire les paramètres dans la mémoire flash
-414 impossible d'établir une connexion avec le module XBee distant

Définition à la ligne 104 du fichier xbeelib.cpp.

```
00104                                   {
00105     if(!enterATMode()){
00106         logXbee « "/!\\ (config AT) erreur " « XB_AT_E_ENTER « " : impossible d'entrer dans le mode
      AT" « mendl;
00107         closeSerialConnection();
00108         if(MODE == 0){
00109             MODE = 1;
00110             openSerialConnection(1);
00111         }else{
00112             MODE = 0;
00113             openSerialConnection();
00114         }
00115         return XB_AT_E_ENTER;
00116     }
00117     else logXbee « "(config AT) entrée dans le mode AT" « mendl;
00118
00119     if(!sendATCommand(XB_AT_CMD_BAUDRATE, XB_AT_V_BAUDRATE, XB_AT_M_GET)){
00120         if(!sendATCommand(XB_AT_CMD_BAUDRATE, XB_AT_V_BAUDRATE)){
00121             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_BAUDRATE « " : impossible de configurer le
      baudrate" « mendl;
00122             return XB_AT_E_BAUDRATE;
```

```
00123         }
00124         logXbee « "(config AT) baudrate configuré avec succès" « mendl;
00125     }
00126     else logXbee « "(config AT) baudrate vérifié avec succès" « mendl;
00127 
00128     if(!sendATCommand(XB_AT_CMD_PARITY, XB_AT_V_PARITY, XB_AT_M_GET)){
00129         if(!sendATCommand(XB_AT_CMD_PARITY, XB_AT_V_PARITY)){
00130             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_PARITY « " : impossible de configurer le
    nombre de bits de parité" « mendl;
00131             return XB_AT_E_PARITY;
00132         }
00133         logXbee « "(config AT) nombre de bits de parité configuré avec succès" « mendl;
00134 
00135         if(!writeATConfig()){
00136             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_WRITE_CONFIG « " : impossible d'écrire les
    paramètres dans la mémoire flash" « mendl;
00137             return XB_AT_E_WRITE_CONFIG;
00138         }
00139 
00140         closeSerialConnection();
00141         if(MODE == 0){
00142             MODE = 1;
00143             openSerialConnection(1);
00144         }else{
00145             MODE = 0;
00146             openSerialConnection();
00147         }
00148     }
00149     else logXbee « "(config AT) nombre de bits de parité vérifié avec succès" « mendl;
00150 
00151     if(!sendATCommand(XB_AT_CMD_API, XB_AT_V_API, XB_AT_M_GET)){
00152         if(!sendATCommand(XB_AT_CMD_API, XB_AT_V_API)){
00153             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_API « " : impossible de configurer le mode
    API" « mendl;
00154             return XB_AT_E_API;
00155         }
00156         logXbee « "(config AT) mode API configuré avec succès" « mendl;
00157     }
00158     else logXbee « "(config AT) mode API vérifié avec succès" « mendl;
00159 
00160     if(!sendATCommand(XB_AT_CMD_AES, XB_AT_V_AES, XB_AT_M_GET)){
00161         if(!sendATCommand(XB_AT_CMD_AES, XB_AT_V_AES)){
00162             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_AES « " : impossible de configurer le
    paramètre de chiffrement AES" « mendl;
00163             return XB_AT_E_AES;
00164         }
00165         logXbee « "(config AT) chiffrement AES configuré avec succès" « mendl;
00166     }
00167     else logXbee « "(config AT) chiffrement AES vérifié avec succès" « mendl;
00168 
00169     if(!sendATCommand(XB_AT_CMD_AES_KEY, XB_AT_V_AES_KEY)){
00170         logXbee « "/!\\ (config AT) erreur " « XB_AT_E_AES_KEY « " : impossible de configurer la clé
    de chiffrement AES" « mendl;
00171         return XB_AT_E_AES_KEY;
00172     }
00173 
00174     if(!sendATCommand(XB_AT_CMD_CHANEL, XB_AT_V_CHANEL, XB_AT_M_GET)){
00175         if(!sendATCommand(XB_AT_CMD_CHANEL, XB_AT_V_CHANEL)){
00176             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_CHANEL « " : impossible de configurer le
    canal de découverte réseau" « mendl;
00177             return XB_AT_E_CHANEL;
00178         }
00179         logXbee « "(config AT) canal de découverte réseau configuré avec succès" « mendl;
00180     }
00181     else logXbee « "(config AT) canal de découverte réseau vérifié avec succès" « mendl;
00182 
00183     if(!sendATCommand(XB_AT_CMD_PAN_ID, XB_AT_V_PAN_ID, XB_AT_M_GET)){
00184         if(!sendATCommand(XB_AT_CMD_PAN_ID, XB_AT_V_PAN_ID)){
00185             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_PAN_ID « " : impossible de configurer l'ID
    du réseau" « mendl;
00186             return XB_AT_E_PAN_ID;
00187         }
00188         logXbee « "(config AT) ID du réseau configuré avec succès" « mendl;
00189     }
00190     else logXbee « "(config AT) ID du réseau vérifié avec succès" « mendl;
00191 
00192     if(!sendATCommand(XB_AT_CMD_COORDINATOR, XB_AT_V_COORDINATOR, XB_AT_M_GET)){
00193         if(!sendATCommand(XB_AT_CMD_COORDINATOR, XB_AT_V_COORDINATOR)){
00194             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_COORDINATOR « " : impossible de configurer
    le mode coordinateur" « mendl;
00195             return XB_AT_E_COORDINATOR;
00196         }
00197         logXbee « "(config AT) mode coordinateur configuré avec succès" « mendl;
00198     }
00199     else logXbee « "(config AT) mode coordinateur vérifié avec succès" « mendl;
00200 
00201     if(!sendATCommand(XB_AT_CMD_16BIT_SOURCE_ADDR, XB_AT_V_16BIT_SOURCE_ADDR, XB_AT_M_GET)){
```

```
00202         if(!sendATCommand(XB_AT_CMD_16BIT_SOURCE_ADDR, XB_AT_V_16BIT_SOURCE_ADDR)){
00203             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_16BIT_SOURCE_ADDR « " : impossible de
      configurer l'adresse source 16bits" « mendl;
00204             return XB_AT_E_16BIT_SOURCE_ADDR;
00205         }
00206         logXbee « "(config AT) adresse source 16bits configurée avec succès" « mendl;
00207     }
00208     else logXbee « "(config AT) adresse source 16bits vérifiée avec succès" « mendl;
00209
00210     if(!sendATCommand(XB_AT_CMD_LOW_DEST_ADDR, XB_AT_V_LOW_DEST_ADDR, XB_AT_M_GET)){
00211         if(!sendATCommand(XB_AT_CMD_LOW_DEST_ADDR, XB_AT_V_LOW_DEST_ADDR)){
00212             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_LOW_DEST_ADDR « " : impossible de
      configurer l'adresse de destination" « mendl;
00213             return XB_AT_E_LOW_DEST_ADDR;
00214         }
00215         logXbee « "(config AT) adresse de destination configurée avec succès" « mendl;
00216     }
00217     else logXbee « "(config AT) adresse de destination vérifiée avec succès" « mendl;
00218
00219     if(!writeATConfig()){
00220         logXbee « "/!\\ (config AT) erreur " « XB_AT_E_WRITE_CONFIG « " : impossible d'écrire les
      paramètres dans la mémoire flash" « mendl;
00221         return XB_AT_E_WRITE_CONFIG;
00222     }
00223     else logXbee « "(config AT) configuration AT enregistrée dans la mémoire du module" « mendl;
00224
00225     if(!discoverXbeeNetwork()){
00226         logXbee « "/!\\ (config AT) erreur " « XB_AT_E_DISCOVER_NETWORK « " : impossible d'établir une
      connexion avec le module XBee distant" « mendl;
00227         return XB_AT_E_DISCOVER_NETWORK;
00228     }
00229     else logXbee « "(config AT) connexion XBee établie avec succès avec le module distant" « mendl;
00230
00231     if(!exitATMode()){
00232         logXbee « "/!\\ (config AT) erreur " « XB_AT_E_EXIT « " : impossible de sortir du mode AT" «
      mendl;
00233         return XB_AT_E_EXIT;
00234     }
00235
00236     logXbee « "(config AT) configuration AT réalisée avec succès" « mendl;
00237     return XB_AT_E_SUCCESS;
00238 }
```

4.6.3.4 closeSerialConnection()

```
void XBee::closeSerialConnection ( )
```

Nettoyage du buffer et fermeture de la connexion UART entre la RaspberryPi et le module XBee.

Définition à la ligne 75 du fichier xbeelib.cpp.

```
00075                                {
00076     serial.flushReceiver();
00077     logXbee « "(serial) buffer Rx nettoyé avec succès" « mendl;
00078
00079     serial.closeDevice();
00080     logXbee « "(serial) connexion série fermée avec succès" « mendl;
00081 }
```

4.6.3.5 crc16()

```
int XBee::crc16 (
            int trame[],
            uint8_t taille )  [private]
```

Fonction permettant de calculer le CRC16 Modbus de la trame XBee envoyée.

Paramètres

trame	: la trame XBee complète sauf le CRC et le caractère de fin de trame
taille	: la taille de la trame

Renvoie

la valeur entière du crc calculée sur 16 bits

Définition à la ligne 351 du fichier xbeelib.cpp.

```
00351                                                {
00352     int crc = 0xFFFF, count = 0;
00353     int octet_a_traiter;
00354     const int POLYNOME = 0xA001;
00355
00356     octet_a_traiter = trame[0];
00357
00358     do{
00359         crc ^= octet_a_traiter;
00360         for(uint8_t i = 0; i < 8; i++){
00361
00362             if((crc%2)!=0)
00363             crc = (crc » 1) ^ POLYNOME;
00364
00365             else
00366                 crc = (crc » 1);
00367
00368         }
00369         count++;
00370         octet_a_traiter = trame[count];
00371
00372     }while(count < taille);
00373
00374     return crc;
00375 }
```

4.6.3.6 delay()

```
void XBee::delay (
             unsigned int time )  [private]
```

Fonction permettant de retarder l'exécution du code.

Paramètres

time	: temps du retard en secondes

Définition à la ligne 244 du fichier xbeelib.cpp.

```
00244 { std::this_thread::sleep_for(std::chrono::milliseconds(time*1000)); }
```

4.6.3.7 discoverXbeeNetwork()

```
bool XBee::discoverXbeeNetwork ( )  [private]
```

Recherche du module XBee distant de l'autre robot.

Renvoie

true le bon module XBee est détecté
false aucun module XBee détecté ou module XBee incorrect détecté

Définition à la ligne 297 du fichier xbeelib.cpp.

```
00297                                {
00298     serial.writeString(XB_AT_CMD_DISCOVER_NETWORK);
00299     serial.writeString(XB_AT_V_END_LINE);
00300     logXbee « "lancement de la découverte réseau XBee" « mendl;
00301     return readATResponse(XB_AT_V_DISCOVER_NETWORK);
00302 }
```

4.6.3.8 enterATMode()

```
bool XBee::enterATMode ( )  [private]
```

Fonction permettant d'entrer dans le mode AT.

Renvoie

true la réponse du module XBee est celle attendue
false la réponse du module XBee n'est pas celle attendue

Définition à la ligne 272 du fichier xbeelib.cpp.

```
00272                         {
00273     serial.writeString(XB_AT_CMD_ENTER);
00274     delay(3);
00275     serial.writeString(XB_AT_V_END_LINE);
00276     logXbee « "entrée en mode AT" « mendl;
00277     return readATResponse(XB_AT_R_SUCCESS);
00278 }
```

4.6.3.9 exitATMode()

```
bool XBee::exitATMode ( )  [private]
```

Fonction permettant de sortir du mode AT.

Renvoie

true la réponse du module XBee est celle attendue
false la réponse du module XBee n'est pas celle attendue

Définition à la ligne 285 du fichier xbeelib.cpp.

```
00285                         {
00286     serial.writeString(XB_AT_CMD_EXIT);
00287     serial.writeString(XB_AT_V_END_LINE);
00288     logXbee « "sortie du mode AT" « mendl;
00289     return readATResponse(XB_AT_R_SUCCESS);
00290 }
```

4.6.3.10 isCodeFctCorrect()

```
bool XBee::isCodeFctCorrect (
            int code_fct )  [private]
```

Vérifie si le code fonction donné est présent dans le fichier define.h.

Paramètres

code_fct	: le code fonction à vérifier

Renvoie

true : le code fonction est correct
false : le code fonction est incorrect/n'existe pas

Définition à la ligne 538 du fichier xbeelib.cpp.

```
00538                                           {
00539     int size_list_code_fct = sizeof(XB_LIST_CODE_FCT)/sizeof(XB_LIST_CODE_FCT[0]), i = 0;
00540
00541     while(i < size_list_code_fct){
00542         if(XB_LIST_CODE_FCT[i] == code_fct)
00543             return true;
00544         i++;
00545     }
00546
00547     return false;
00548 }
```

4.6.3.11 isCRCCorrect()

```
bool XBee::isCRCCorrect (
              uint8_t crc_low,
              uint8_t crc_high,
              int trame[],
              int trame_size )  [private]
```

Vérifie si le CRC reçu est cohérent avec la trame reçue.

Paramètres

crc_low	: les bits de poids faible du CRC reçu
crc_high	: les bits de poids forts du CRC reçu
trame	: la trame reçue (en enlevant le CRC et le caratère de fin de trame)
trame_size	: la taille de le trame telle qu'entrée dans la fonction

Renvoie

true : la valeur du CRC reçue est bien celle calculée à partir du reste de la trame
false : la valeur du CRC est incohérente ou non calculable

Définition à la ligne 636 du fichier xbeelib.cpp.

```
00636                                                                                   {
00637     int crc = crc16(trame, trame_size);
00638
00639     uint8_t newcrc_low = crc & 0xFF;
00640     uint8_t newcrc_high = (crc » 8) & 0xFF;
00641
00642     if(newcrc_low == crc_low && newcrc_high == crc_high)
00643         return true;
00644
00645     return false;
00646 }
```

4.6.3.12 isDestCorrect()

```
bool XBee::isDestCorrect (
            int dest )  [private]
```

Vérifie si une adresse de destination est correcte.

Paramètres

exp	: l'adresse de destination à vérifier

Renvoie

true : l'adresse est correcte
false : l'adresse est incorrecte

Définition à la ligne 588 du fichier xbeelib.cpp.

```
00588                                   {
00589     int size_list_addr = sizeof(XB_LIST_ADR)/sizeof(XB_LIST_ADR[0]), i = 0;
00590
00591     while(i < size_list_addr){
00592         if(XB_LIST_ADR[i] == dest)
00593             return true;
00594
00595         i++;
00596     }
00597
00598     return false;
00599 }
```

4.6.3.13 isEndSeqCorrect()

```
bool XBee::isEndSeqCorrect (
            int value )  [private]
```

Vérifie si le caractère de fin de la trame correpond à celui attendu.

Paramètres

value	: le caractère à vérifier

Renvoie

true : le caratère est bien celui attendu
false : le caractère est incorrect

Définition à la ligne 620 du fichier xbeelib.cpp.

```
00620                                   {
00621     if(value == XB_V_END)
00622         return true;
00623
00624     return false;
00625 }
```

4.6.3.14 isExpCorrect()

```
bool XBee::isExpCorrect (
            int exp )  [private]
```

Vérifie si une adresse d'expéditeur est correcte.

Paramètres

exp	: l'adresse de l'expéditeur à vérifier

Renvoie

true : l'adresse est correcte
false : l'adresse est incorrecte

Définition à la ligne 569 du fichier xbeelib.cpp.

```
00569                                {
00570     int size_list_addr = sizeof(XB_LIST_ADR)/sizeof(XB_LIST_ADR[0]), i = 0;
00571
00572     while(i < size_list_addr){
00573         if(XB_LIST_ADR[i] == exp)
00574             return true;
00575
00576         i++;
00577     }
00578
00579     return false;
00580 }
```

4.6.3.15 isStartSeqCorrect()

```
bool XBee::isStartSeqCorrect (
            int value )  [private]
```

Vérifie si le caractère de début de la trame correpond à celui attendu.

Paramètres

value	: le caractère à vérifier

Renvoie

true : le caratère est bien celui attendu
false : le caractère est incorrect

Définition à la ligne 607 du fichier xbeelib.cpp.

```
00607                                    {
00608     if(value == XB_V_START)
00609         return true;
00610
00611     return false;
00612 }
```

4.6.3.16 isTrameSizeCorrect()

```
bool XBee::isTrameSizeCorrect (
            std::vector< int > trame )  [private]
```

Vérifie si la taille de la trame est cohérente.

Paramètres

trame	: la trame à vérifier

Renvoie

true : la taille de la trame semble cohérente
false : la taille de la trame est incorrecte, trop petite ou non cohérente

Définition à la ligne 556 du fichier xbeelib.cpp.

```
00556                                           {
00557     if(trame.size() > 10 && trame.size() == trame[4]+5)
00558         return true;
00559
00560     return false;
00561 }
```

4.6.3.17 isXbeeResponding()

```
int XBee::isXbeeResponding ( )
```

Permet de vérifier si un message envoyé a reçu une réponse.

Définition à la ligne 801 du fichier xbeelib.cpp.

```
00801                           {
00802     int size_list_code_fct = sizeof(XB_LIST_CODE_FCT)/sizeof(XB_LIST_CODE_FCT[0]);
00803     while(true){
00804       delay(3);
00805       for(int i = 0; i < size_list_code_fct; i++){
00806           if(trames_envoyees[XB_LIST_CODE_FCT[i]] == 0){
00807               logXbee << "(verif reponse) les trames envoyées portant le code fonction " <<
       XB_LIST_CODE_FCT[i] << " ont toutes reçues une réponse" << endl;
00808           }else{
00809               logXbee << "(verif reponse) /!\\ les trames envoyées portant le code fonction " <<
       XB_LIST_CODE_FCT[i] << " n'ont pas toutes reçues une réponse" << endl;
00810           }
00811       }
00812    }
00813 }
```

4.6.3.18 openSerialConnection()

```
int XBee::openSerialConnection (
            int mode = 0 )
```

Nettoyage du buffer et ouverture de la connexion UART entre la RaspberryPi et le module XBee.

Paramètres

mode	permet de définir la configuration de port à utiliser

Renvoie

500 succès
-501 port série non trouvé
-502 erreur lors de l'ouverture du port série
-503 erreur lors de la récupération des informations du port série
-504 baudrate non reconnu
-505 erreur lors de l'écriture de la configuration du port série
-506 erreur lors de l'écriture du timeout
-507 databits non reconnus
-508 stopbits non reconnus
-509 parité non reconnue

Définition à la ligne 46 du fichier xbeelib.cpp.

```
00046                                       {
00047     int errorOpening;
00048     if(mode == 1){
00049         errorOpening = serial.openDevice(XB_SERIAL_PORT_DEFAULT, XB_BAUDRATE_DEFAULT,
      XB_DATABITS_DEFAULT, XB_PARITY_DEFAULT, XB_STOPBITS_DEFAULT);
00050
00051         if (errorOpening != 1)
00052             logXbee « "(serial) /!\\ erreur " « errorOpening « " : impossible d'ouvrir le port " «
      XB_SERIAL_PORT_DEFAULT  « " - baudrate : " « XB_BAUDRATE_DEFAULT « " - parité : " « XB_PARITY_DEFAULT
      « mendl;
00053         else{
00054             logXbee « "(serial) connexion ouverte avec succès sur le port " « XB_SERIAL_PORT_DEFAULT «
      " - baudrate : " « XB_BAUDRATE_DEFAULT « " - parité : " « XB_PARITY_DEFAULT « mendl;
00055         checkATConfig();
00056     }
00057     } else if(mode == 0) {
00058         errorOpening = serial.openDevice(XB_SERIAL_PORT_PRIMARY, XB_BAUDRATE_PRIMARY,
      XB_DATABITS_PRIMARY, XB_PARITY_PRIMARY, XB_STOPBITS_PRIMARY);
00059
00060         if (errorOpening != 1)
00061             logXbee « "(serial) /!\\ erreur " « errorOpening « " : impossible d'ouvrir le port " «
      XB_SERIAL_PORT_PRIMARY  « " - baudrate : " « XB_BAUDRATE_PRIMARY « " - parités : " «
      XB_PARITY_PRIMARY « mendl;
00062
00063     else{
00064             logXbee « "(serial) connexion ouverte avec succès sur le port " « XB_SERIAL_PORT_PRIMARY «
      " - baudrate : " « XB_BAUDRATE_PRIMARY « " - parité : " « XB_PARITY_PRIMARY « mendl;
00065             checkATConfig();
00066         }
00067     }
00068
00069     return errorOpening;
00070 }
```

4.6.3.19 print()

```
void XBee::print (
            const std::vector< int > & v )
```

Fonction d'affichage des valeurs contenues dans un vecteur d'entiers.

Paramètres

v	: le vecteur dont on souhaite afficher le contenu

Définition à la ligne 865 du fichier xbeelib.cpp.

```
00865                                       {
00866     copy(v.begin(), v.end(),
00867             ostream_iterator<int>(cout « hex, " "));
00868     cout « endl;
00869 }
```

4.6.3.20 processCodeFct()

```
int XBee::processCodeFct (
            int code_fct,
            int exp )  [private]
```

Interprète le code fonction issu d'une trame reçue.

Renvoie

100 succès
-101 code fonction incorrect
-102 code fonction existant mais ne déclenchant aucune action

Définition à la ligne 510 du fichier xbeelib.cpp.

```
00510                                               {
00511     if(!isCodeFctCorrect(code_fct)){
00512         logXbee « "/!\\ (process code fonction) erreur " « XB_FCT_E_NOT_FOUND « " : code fonction
      incorrect " « endl;
00513         return XB_FCT_E_NOT_FOUND;
00514     }
00515     char msg[1];
00516     switch(code_fct){
00517        case XB_FCT_TEST_ALIVE :
00518            msg[0] = {XB_V_ACK};
00519            sendTrame(exp, XB_FCT_TEST_ALIVE, msg);
00520            break;
00521
00522        default :
00523            logXbee « "/!\\ (process code fonction) erreur " « XB_FCT_E_NONE_REACHABLE « " : code
      fonction existant mais ne déclenchant aucune action  " « endl;
00524            return XB_FCT_E_NONE_REACHABLE;
00525     }
00526
00527     trames_envoyees[code_fct] = trames_envoyees[code_fct]-1;
00528     logXbee « "(process code fonction) code fonction n°" « code_fct « " traité avec succès" « endl;
00529     return XB_FCT_E_SUCCESS;
00530 }
```

4.6.3.21 processTrame()

```
int XBee::processTrame (
            std::vector< int > trame )  [private]
```

Découpe une trame reçue en fonction de ses paramètres et interprete son code fonction.

Renvoie

200 succès
-201 taille de la trame incorrecte ou non concordante
-202 premier caractère de la trame incorrect
-203 dernier caractère de la trame incorrect
-204 valeur du CRC incorrecte
-205 adresse de l'expéditeur incorrecte ou inconnue
-206 adresse du destinataire incorrecte ou inconnue

Définition à la ligne 437 du fichier xbeelib.cpp.

```
00437                                                {
00438
00439     if(!isTrameSizeCorrect(trame_recue)){
00440         logXbee « "/!\\ (process trame) erreur " « XB_TRAME_E_SIZE « " : taille de la trame incorrecte
      ou non concordante " « mendl;
00441         return XB_TRAME_E_SIZE;
00442     }
00443     Trame_t trame = {
00444         .start_seq = trame_recue[0],
00445         .adr_emetteur = trame_recue[1],
00446         .adr_dest = trame_recue[2],
00447         .id_trame_low = trame_recue[3]-4,
00448         .id_trame_high = trame_recue[4]-4,
00449         .nb_octets_msg = trame_recue[5]-4,
00450         .code_fct = trame_recue[6],
00451         .crc_low = trame_recue[3+trame_recue[4]],
00452         .crc_high = trame_recue[4+trame_recue[4]],
00453         .end_seq = trame_recue[5+trame_recue[4]]
00454     };
00455
00456     vector<int> data {};
00457
00458     for(uint8_t i = 0; i < trame.nb_octets_msg; i++){
00459        data.push_back(trame_recue[7+i]);
00460     }
00461
00462     trame.param = data;
00463
00464     afficherTrameRecue(trame);
00465
00466     int decoupe_trame[trame_recue[4]+6];
00467
00468     for(uint8_t i = 0; i < trame_recue[4]+3; i++){
00469         decoupe_trame[i] = trame_recue[i];
00470     }
00471
00472     if(!isStartSeqCorrect(trame.start_seq)){
00473         logXbee « "/!\\ (process trame) erreur " « XB_TRAME_E_START « " : premier caractère de la
      trame incorrect " « mendl;
00474         return XB_TRAME_E_START;
00475     }
00476
00477     if(!isEndSeqCorrect(trame.end_seq)){
00478         logXbee « "/!\\ (process trame) erreur " « XB_TRAME_E_END « " : dernier caractère de la trame
      incorrect " « mendl;
00479         return XB_TRAME_E_END;
00480     }
00481
00482     if(!isCRCCorrect(trame.crc_low, trame.crc_high, decoupe_trame, trame_recue[4]+2)){
00483         logXbee « "/!\\ (process trame) erreur " « XB_TRAME_E_CRC « " : valeur du CRC incorrecte " «
      mendl;
00484         return XB_TRAME_E_CRC;
00485     }
00486
00487     if(!isExpCorrect(trame.adr_emetteur)){
00488         logXbee « "/!\\ (process trame) erreur " « XB_TRAME_E_EXP « " : adresse de l'expéditeur
      incorrecte ou inconnue " « mendl;
00489         return XB_TRAME_E_EXP;
00490     }
00491
00492     if(!isDestCorrect(trame.adr_dest)){
00493         logXbee « "/!\\ (process trame) erreur " « XB_TRAME_E_DEST « " : adresse du destinataire
      incorrecte ou inconnue " « mendl;
00494         return XB_TRAME_E_DEST;
00495     }
00496
00497     processCodeFct(trame.code_fct, trame.adr_emetteur);
00498
00499     logXbee « "(process trame) trame n°" « trame.id_trame_high+trame.id_trame_low « "a été traitée
      avec succès " « mendl;
00500
```

```
00501     return XB_TRAME_E_SUCCESS;
00502 }
```

4.6.3.22 readATResponse()

```
bool XBee::readATResponse (
            const char * value = XB_AT_R_EMPTY,
            int mode = 0 )
```

Fonction permettant de lire la réponse à un envoi de commande AT au module XBee.

Paramètres

value	: la valeur de réponse attendue pour la commande envoyée
mode	: le mode de lecture à utiliser

Renvoie

true la réponse du module XBee est celle attendue
false la réponse du module XBee n'est pas celle attendue

Définition à la ligne 254 du fichier xbeelib.cpp.

```
00254                                                   {
00255     string reponse = readString();
00256     logXbee « "(config AT) réponse du Xbee : " « reponse « mendl;
00257
00258     if(mode == 0)
00259         if(reponse == value) return true;
00260
00261     else if(mode == 1)
00262         if(reponse != value) return true;
00263
00264     return false;
00265 }
```

4.6.3.23 readBuffer()

```
vector< int > XBee::readBuffer ( )  [private]
```

Permet de lire l'intégralité du buffer Rx de la RaspberryPi.

Renvoie

rep : la valeur du buffer sous forme d'un vecteur d'entiers signés sur 32 bits

Définition à la ligne 652 du fichier xbeelib.cpp.

```
00652                                {
00653     char *reponse(0);
00654     unsigned int timeout = 100;
00655     reponse = new char;
00656     vector<int> rep;
00657     delay(1);
00658     int i = 0;
00659     while(serial.available() > 0){
00660         i++;
00661         serial.readChar(reponse, timeout);
00662         rep.push_back(*reponse);
00663     }
00664     delete reponse;
00665     reponse = 0;
00666
00667     return rep;
00668 }
```

4.6.3.24 readString()

```
string XBee::readString ( )  [private]
```

Permet de lire l'intégralité du contenu du buffer Rx de la RaspberryPi et de le renvoyer sous forme d'objet string.

Renvoie

rep : la valeur du buffer concaténée sous forme d'objet string

Définition à la ligne 675 du fichier xbeelib.cpp.

```
00675                          {
00676      char *reponse(0);
00677      unsigned int timeout = 100;
00678      reponse = new char;
00679      string rep;
00680      delay(1);
00681      int i = 0;
00682
00683      while(serial.available() > 0){
00684     i++;
00685     serial.readChar(reponse, timeout);
00686     rep += *reponse;
00687      }
00688
00689       delete reponse;
00690       reponse = 0;
00691       return rep;
00692
00693 }
```

4.6.3.25 sendATCommand()

```
bool XBee::sendATCommand (
              const char * command,
              const char * value,
              unsigned int mode = XB_AT_M_SET )
```

Fonction permettant d'envoyer en UART via le port série une commmande AT.

Paramètres

command	: le paramètre AT a envoyer au module
value	: la valeur de réponse attendue
mode	: le mode de transmission de la commande AT (mode lecture ou écriture)

Renvoie

true la réponse du module XBee est celle attendue
false la réponse du module XBee n'est pas celle attendue

Définition à la ligne 325 du fichier xbeelib.cpp.

```
00325                                                                                {
00326     if(mode == XB_AT_M_GET){
00327         serial.writeString(command);
00328         serial.writeString(XB_AT_V_END_LINE);
00329         logXbee « "(config AT) envoi de la commande AT : " « command « endl;
00330         return readATResponse(value);
00331     }else{
00332         serial.writeString(command);
```

```
00333         serial.writeString(value);
00334         logXbee « "(config AT) envoi de la commande AT : " « command « "=" « value « mendl;
00335         if(command == XB_AT_CMD_DISCOVER_NETWORK)
00336             return readATResponse(XB_AT_R_SUCCESS);
00337         else
00338             return readATResponse(XB_AT_V_DISCOVER_NETWORK, 1);
00339     }
00340 }
```

4.6.3.26 sendHeartbeat()

```
void XBee::sendHeartbeat ( )
```

Permet d'envoyer des demandes de battements de coeur au second robot afin de savoir s'il est toujours opérationnel.

Définition à la ligne 788 du fichier xbeelib.cpp.

```
00788                           {
00789     char* msg;
00790     msg[0] = XB_V_ACK;
00791
00792     while(true){
00793       delay(3);
00794       sendTrame(XB_ADR_ROBOT_02, XB_FCT_TEST_ALIVE, msg);
00795     }
00796 }
```

4.6.3.27 sendMsg()

```
void XBee::sendMsg (
            std::string msg )
```

Permet d'envoyer un mesage ASCII sans format particulier via XBee.

Paramètres

msg	: le message à envoyer

Définition à la ligne 819 du fichier xbeelib.cpp.

```
00819                               {
00820     serial.writeString(stringToChar(msg));
00821     logXbee « "(envoi message) message : " « msg « " envoyé avec succès" « mendl;
00822 }
```

4.6.3.28 sendTrame()

```
int XBee::sendTrame (
            uint8_t ad_dest,
            uint8_t code_fct,
            char * data = 0x00 )
```

Fonction permettant d'envoyer une trame de message structurée via UART en XBee.

Paramètres

ad_dest	: l'adresse du destinataire du message
code_fct	: le code de la fonction concernée par le message
data	: les valeurs des paramètres demandées par le code fonction

Renvoie

{XB_TRAME_E_SUCCESS} succès

Définition à la ligne 384 du fichier xbeelib.cpp.

```
00384                                                                   {
00385
00386     cout « hex « showbase;
00387
00388     uint8_t length_trame = strlen(data)+10;
00389     uint8_t trame[length_trame];
00390     int trame_int[length_trame];
00391     int id_trame = ++ID_TRAME;
00392     uint8_t id_trame_low = id_trame & 0xFF;
00393     uint8_t id_trame_high = (id_trame » 8) & 0xFF;
00394
00395     trame[0] = XB_V_START;
00396     trame[1] = XB_ADR_CURRENT_ROBOT;
00397     trame[2] = ad_dest;
00398     trame[3] = id_trame_low+4;
00399     trame[4] = id_trame_high+4;
00400     trame[5] = strlen(data)+4;
00401     trame[6] = code_fct;
00402
00403     for(size_t i = 0; i < strlen(data); i++){
00404         trame[i+7] = data[i];
00405     }
00406
00407
00408     for(int i=0; i < length_trame; i++){
00409         trame_int[i] = int(trame[i]);
00410     }
00411     int crc = crc16(trame_int, strlen(data)+6);
00412     uint8_t crc_low = crc & 0xFF;
00413     uint8_t crc_high = (crc » 8) & 0xFF;
00414
00415     trame[strlen(data)+7] = crc_low;
00416     trame[strlen(data)+8] = crc_high;
00417     trame[strlen(data)+9] = XB_V_END;
00418
00419     serial.writeBytes(trame, length_trame);
00420     logXbee « "(sendTrame) envoi de la trame n°" « dec « id_trame_low+id_trame_high  « " effectué avec
      succès" « mendl;
00421
00422     trames_envoyees[code_fct] = trames_envoyees[code_fct]+1;
00423
00424     return XB_TRAME_E_SUCCESS;
00425 }
```

4.6.3.29 slice()

```
vector< int > XBee::slice (
            const std::vector< int > & v,
            int a,
            int b )  [private]
```

Fonction de traitement permettant d'extraire un sous-vecteur d'entiers d'un vecteur d'entiers.

Paramètres

v	: le vecteur à découper
a	: l'indice de la première valeur à découper
b	: l'indice de la dernière valeur à découper

Renvoie

vec : le sous-vecteur d'entiers découpé

Définition à la ligne 878 du fichier xbeelib.cpp.

```
00878                                                       {
00879     auto first = v.cbegin() + a;
00880     auto last = v.cbegin() + b + 1;
00881
00882     vector<int> vec(first, last);
00883     return vec;
00884 }
```

4.6.3.30 stringToChar()

```
char * XBee::stringToChar (
            std::string chaine )
```

Permet de convertir un objet de type string en chaine de caractère standard C.

Paramètres

chaine	: l'objet string à convertir

Renvoie

message : la chaine de caractère convertie

Définition à la ligne 829 du fichier xbeelib.cpp.

```
00829                                   {
00830     char* message = strcpy(new char[chaine.size() + 1], chaine.c_str());
00831     return message;
00832 }
```

4.6.3.31 subTrame()

```
int XBee::subTrame (
            std::vector< int > msg_recu )  [private]
```

Découpe le résultat de la lecture du buffer en différentes trames avant le traitement.

Paramètres

msg_recu	: le résultat de la lecture du buffer

Renvoie

300 succès

-301 la position des trames dans le message reçu est incorrecte : les caractères de début et de fin de trame ne sont pas au même nombre

-302 la position des trames dans le message reçu est incorrecte : certains caractères de début de trame sont placés après des caractères de fin de trame

-303 la position des trames dans le message reçu est incorrecte : des caractères inconnus sont placés entre deux trames

-304 le premier caractère lu dans le buffer n'est pas celui d'un début de trame

-305 le dernier caractère lu dans le buffer n'est pas celui d'une fin de trame

-306 aucun caractère de début et/ou de fin n'est présent dans le message reçu

Définition à la ligne 725 du fichier xbeelib.cpp.

```
00725                                       {
00726
00727     vector<int> list_start_seq {};
00728     vector<int> list_end_seq {};
00729     vector<int> decoupe {};
00730     int decoupe_retour;
00731
00732     for(uint8_t i = 0; i < msg_recu.size(); i++){
00733         if(msg_recu[i] == XB_V_START)
00734             list_start_seq.push_back(i);
00735
00736         if(msg_recu[i] == XB_V_END)
00737             list_end_seq.push_back(i);
00738     }
00739
00740     if(list_start_seq.size() == 0 || list_end_seq.size() == 0){
00741         logXbee « "/!\\ (découpe trame) erreur " « XB_SUB_TRAME_E_NULL « " : aucun caractère de début
    et/ou de fin n'est présent dans le message reçu " « mendl;
00742         return XB_SUB_TRAME_E_NULL;
00743     }
00744
00745     if(list_start_seq.size() != list_end_seq.size()){
00746         logXbee « "/!\\ (découpe trame) erreur " « XB_SUB_TRAME_E_SIZE « " : les caractères de début
    et de fin de trame ne sont pas au même nombre " « mendl;
00747         return XB_SUB_TRAME_E_SIZE;
00748     }
00749
00750     for(uint8_t i = 0; i < list_start_seq.size(); i++){
00751         if(list_start_seq[i] > list_end_seq[i]){
00752             logXbee « "/!\\ (découpe trame) erreur " « XB_SUB_TRAME_E_REPARTITION « " : certains
    caractères de début de trame sont placés après des caractères de fin de trame " « mendl;
00753             return XB_SUB_TRAME_E_REPARTITION;
00754         }
00755
00756         if(i != 0){
00757             if(list_start_seq[i] != list_end_seq[i-1]-1){
00758                 logXbee « "/!\\ (découpe trame) erreur " « XB_SUB_TRAME_E_DECOUPAGE « " : des
    caractères inconnus sont placés entre deux trames " « mendl;
00759                 return XB_SUB_TRAME_E_DECOUPAGE;
00760             }
00761         }
00762     }
00763
00764
00765     if(list_start_seq[0] != 0){
00766         logXbee « "/!\\ (découpe trame) erreur " « XB_SUB_TRAME_E_START « " : le premier caractère lu
    dans le buffer n'est pas celui d'un début de trame " « mendl;
00767         return XB_SUB_TRAME_E_START;
00768     }
00769
00770     if(list_end_seq[list_end_seq.size()-1] != msg_recu.size()-1){
00771         logXbee « "/!\\ (découpe trame) erreur " « XB_SUB_TRAME_E_END « " : le dernier caractère lu
    dans le buffer n'est pas celui d'une fin de trame " « mendl;
00772         return XB_SUB_TRAME_E_END;
00773     }
00774
00775     for(uint8_t i = 0; i < list_start_seq.size(); i++){
00776        decoupe.clear();
00777        decoupe = slice(msg_recu, list_start_seq[i], list_end_seq[i]);
00778        decoupe_retour = processTrame(decoupe);
00779     }
00780
00781     logXbee « "(découpe trame) découpage des trames effectué avec succès" « mendl;
00782     return XB_SUB_TRAME_E_SUCCESS;
00783 }
```

4.6.3.32 waitForATrame()

```
void XBee::waitForATrame ( )
```

Permet l'attente et la vérification régulée d'une trame en entrée dans le buffer du port Rx de la RaspberryPi et d'appeler la fonction de découpe des trames.

Définition à la ligne 698 du fichier xbeelib.cpp.

```
00698                           {
00699     vector<int> rep;
00700
00701     while(true){
00702       rep.clear();
00703
00704       delay(1/100);
00705
00706       if(serial.available() > 0){
00707         rep = readBuffer();
00708         subTrame(rep);
00709       }
00710     }
00711 }
```

4.6.3.33 writeATConfig()

```
bool XBee::writeATConfig ( )
```

Fonction permettant d'écrire dans la mémoire flash du module XBee, les paramètres AT définis.

Renvoie

true la réponse du module XBee est celle attendue
false la réponse du module XBee n'est pas celle attendue

Définition à la ligne 309 du fichier xbeelib.cpp.

```
00309                           {
00310     serial.writeString(XB_AT_CMD_WRITE_CONFIG);
00311     serial.writeString(XB_AT_V_END_LINE);
00312     //cout « "* Ecriture de la configuration AT..." « endl;
00313     logXbee « "écriture des paramètres AT dans la mémoire" « mendl;
00314     return readATResponse(XB_AT_R_SUCCESS);
00315 }
```

4.6.4 Documentation des données membres

4.6.4.1 ID_TRAME

```
int XBee::ID_TRAME = 0  [private]
```

Définition à la ligne 156 du fichier xbeelib.h.

4.6.4.2 MODE

```
int XBee::MODE = 0  [private]
```

Définition à la ligne 159 du fichier xbeelib.h.

4.6.4.3 trames_envoyees

```
std::vector<int> XBee::trames_envoyees = {}  [private]
```

Définition à la ligne 162 du fichier xbeelib.h.

La documentation de cette classe a été générée à partir du fichier suivant :

- xbeelib.h
- xbeelib.cpp

Chapitre 5

Documentation des fichiers

5.1 Référence du fichier define.h

Fichier contenant l'ensemble des constantes utilisées dans la librairie XBee.

Macros

- #define XB_SERIAL_PORT_PRIMARY "/dev/ttyAMA0"
- #define XB_BAUDRATE_PRIMARY 9600
- #define XB_DATABITS_PRIMARY SERIAL_DATABITS_8
- #define XB_PARITY_PRIMARY SERIAL_PARITY_EVEN
- #define XB_STOPBITS_PRIMARY SERIAL_STOPBITS_1
- #define XB_SERIAL_PORT_DEFAULT "/dev/ttyAMA0"
- #define XB_BAUDRATE_DEFAULT 9600
- #define XB_DATABITS_DEFAULT SERIAL_DATABITS_8
- #define XB_PARITY_DEFAULT SERIAL_PARITY_NONE
- #define XB_STOPBITS_DEFAULT SERIAL_STOPBITS_1
- #define XB_V_ACK 0x06
- #define XB_V_NACK 0x15
- #define XB_ADR_BROADCAST 0x11
- #define XB_ADR_ROBOT_01 0x12
- #define XB_ADR_ROBOT_02 0x13
- #define XB_LIST_ADR (int[ ]){XB_ADR_BROADCAST, XB_ADR_ROBOT_01, XB_ADR_ROBOT_02}
- #define XB_ADR_CURRENT_ROBOT XB_ADR_ROBOT_01
- #define XB_V_START 0x02
- #define XB_V_END 0x03
- #define XB_FCT_TEST_ALIVE 0x1E
- #define XB_LIST_CODE_FCT (int[ ]){XB_FCT_TEST_ALIVE}
- #define XB_E_SUCCESS 000
- #define XB_FCT_E_SUCCESS 100
- #define XB_FCT_E_NOT_FOUND -101
- #define XB_FCT_E_NONE_REACHABLE -102
- #define XB_TRAME_E_SUCCESS 200
- #define XB_TRAME_E_SIZE -201
- #define XB_TRAME_E_START -202
- #define XB_TRAME_E_END -203
- #define XB_TRAME_E_CRC -204
- #define XB_TRAME_E_EXP -205
- #define XB_TRAME_E_DEST -206
- #define XB_SUB_TRAME_E_SUCCESS 300
- #define XB_SUB_TRAME_E_SIZE -301
- #define XB_SUB_TRAME_E_REPARTITION -302
- #define XB_SUB_TRAME_E_DECOUPAGE -303

- #define XB_SUB_TRAME_E_START -304
- #define XB_SUB_TRAME_E_END -305
- #define XB_SUB_TRAME_E_NULL -306
- #define XB_AT_E_SUCCESS 400
- #define XB_AT_E_ENTER -401
- #define XB_AT_E_API -402
- #define XB_AT_E_BAUDRATE -403
- #define XB_AT_E_AES -404
- #define XB_AT_E_AES_KEY -405
- #define XB_AT_E_CHANEL -406
- #define XB_AT_E_PAN_ID -407
- #define XB_AT_E_COORDINATOR -408
- #define XB_AT_E_PARITY -409
- #define XB_AT_E_16BIT_SOURCE_ADDR -410
- #define XB_AT_E_LOW_DEST_ADDR -411
- #define XB_AT_E_EXIT -412
- #define XB_AT_E_WRITE_CONFIG -413
- #define XB_AT_E_DISCOVER_NETWORK -414
- #define XB_SER_E_SUCCESS 500
- #define XB_SER_E_NOT_FOUND -501
- #define XB_SER_E_OPEN -502
- #define XB_SER_E_PARAM -503
- #define XB_SER_E_UKN_BAUDRATE -504
- #define XB_SER_E_CONFIG -505
- #define XB_SER_E_TIMOUT -506
- #define XB_SER_E_UKN_DATABITS -507
- #define XB_SER_E_UKN_STOPBITS -508
- #define XB_SER_E_UKN_PARITY -509
- #define XB_AT_CMD_ENTER "+++"
- #define XB_AT_CMD_EXIT "ATCN"
- #define XB_AT_CMD_WRITE_CONFIG "ATWR"
- #define XB_AT_CMD_API "ATAP"
- #define XB_AT_CMD_BAUDRATE "ATBD"
- #define XB_AT_CMD_AES "ATEE"
- #define XB_AT_CMD_AES_KEY "ATKY"
- #define XB_AT_CMD_CHANEL "ATCH"
- #define XB_AT_CMD_PAN_ID "ATID"
- #define XB_AT_CMD_COORDINATOR "ATCE"
- #define XB_AT_CMD_PARITY "ATNB"
- #define XB_AT_CMD_16BIT_SOURCE_ADDR "ATMY"
- #define XB_AT_CMD_LOW_DEST_ADDR "ATDL"
- #define XB_AT_CMD_DISCOVER_NETWORK "ATND"
- #define XB_AT_V_END_LINE "\r"
- #define XB_AT_V_API "0\r"
- #define XB_AT_V_BAUDRATE "3\r"
- #define XB_AT_V_AES "1\r"
- #define XB_AT_V_AES_KEY "32303032\r"
- #define XB_AT_V_CHANEL "C\r"
- #define XB_AT_V_PAN_ID "3332\r"
- #define XB_AT_V_COORDINATOR "0\r"
- #define XB_AT_V_PARITY "1\r"
- #define XB_AT_V_16BIT_SOURCE_ADDR "1\r"
- #define XB_AT_V_LOW_DEST_ADDR "2\r"
- #define XB_AT_V_DISCOVER_NETWORK ""
- #define XB_AT_R_EMPTY ""
- #define XB_AT_R_SUCCESS "OK\r"
- #define XB_AT_R_ERROR "ERROR\r"
- #define XB_AT_M_GET 1
- #define XB_AT_M_SET 2

5.1.1 Description détaillée

Fichier contenant l'ensemble des constantes utilisées dans la librairie XBee.

Auteur

Samuel-Charles DITTE-DESTREE (samuelditte destree@protonmail.com)

Version

3.0

Date

10/03/2022

Définition dans le fichier define.h.

5.1.2 Documentation des macros

5.1.2.1 XB_ADR_BROADCAST

```
#define XB_ADR_BROADCAST 0x11
```

Définition à la ligne 31 du fichier define.h.

5.1.2.2 XB_ADR_CURRENT_ROBOT

```
#define XB_ADR_CURRENT_ROBOT XB_ADR_ROBOT_01
```

Définition à la ligne 37 du fichier define.h.

5.1.2.3 XB_ADR_ROBOT_01

```
#define XB_ADR_ROBOT_01 0x12
```

Définition à la ligne 32 du fichier define.h.

5.1.2.4 XB_ADR_ROBOT_02

```
#define XB_ADR_ROBOT_02 0x13
```

Définition à la ligne 33 du fichier define.h.

5.1.2.5 XB_AT_CMD_16BIT_SOURCE_ADDR

```
#define XB_AT_CMD_16BIT_SOURCE_ADDR "ATMY"
```

Définition à la ligne 115 du fichier define.h.

5.1.2.6 XB_AT_CMD_AES

```
#define XB_AT_CMD_AES "ATEE"
```

Définition à la ligne 109 du fichier define.h.

5.1.2.7 XB_AT_CMD_AES_KEY

```
#define XB_AT_CMD_AES_KEY "ATKY"
```

Définition à la ligne 110 du fichier define.h.

5.1.2.8 XB_AT_CMD_API

```
#define XB_AT_CMD_API "ATAP"
```

Définition à la ligne 107 du fichier define.h.

5.1.2.9 XB_AT_CMD_BAUDRATE

```
#define XB_AT_CMD_BAUDRATE "ATBD"
```

Définition à la ligne 108 du fichier define.h.

5.1.2.10 XB_AT_CMD_CHANEL

```
#define XB_AT_CMD_CHANEL "ATCH"
```

Définition à la ligne 111 du fichier define.h.

5.1.2.11 XB_AT_CMD_COORDINATOR

```
#define XB_AT_CMD_COORDINATOR "ATCE"
```

Définition à la ligne 113 du fichier define.h.

5.1.2.12 XB_AT_CMD_DISCOVER_NETWORK

```
#define XB_AT_CMD_DISCOVER_NETWORK "ATND"
```

Définition à la ligne 117 du fichier define.h.

5.1.2.13 XB_AT_CMD_ENTER

```
#define XB_AT_CMD_ENTER "+++"
```

Définition à la ligne 104 du fichier define.h.

5.1.2.14 XB_AT_CMD_EXIT

```
#define XB_AT_CMD_EXIT "ATCN"
```

Définition à la ligne 105 du fichier define.h.

5.1.2.15 XB_AT_CMD_LOW_DEST_ADDR

```
#define XB_AT_CMD_LOW_DEST_ADDR "ATDL"
```

Définition à la ligne 116 du fichier define.h.

5.1.2.16 XB_AT_CMD_PAN_ID

```
#define XB_AT_CMD_PAN_ID "ATID"
```

Définition à la ligne 112 du fichier define.h.

5.1.2.17 XB_AT_CMD_PARITY

```
#define XB_AT_CMD_PARITY "ATNB"
```

Définition à la ligne 114 du fichier define.h.

5.1.2.18 XB_AT_CMD_WRITE_CONFIG

```
#define XB_AT_CMD_WRITE_CONFIG "ATWR"
```

Définition à la ligne 106 du fichier define.h.

5.1.2.19 XB_AT_E_16BIT_SOURCE_ADDR

```
#define XB_AT_E_16BIT_SOURCE_ADDR -410
```

Définition à la ligne 85 du fichier define.h.

5.1.2.20 XB_AT_E_AES

```
#define XB_AT_E_AES -404
```

Définition à la ligne 79 du fichier define.h.

5.1.2.21 XB_AT_E_AES_KEY

```
#define XB_AT_E_AES_KEY -405
```

Définition à la ligne 80 du fichier define.h.

5.1.2.22 XB_AT_E_API

```
#define XB_AT_E_API -402
```

Définition à la ligne 77 du fichier define.h.

5.1.2.23 XB_AT_E_BAUDRATE

```
#define XB_AT_E_BAUDRATE -403
```

Définition à la ligne 78 du fichier define.h.

5.1.2.24 XB_AT_E_CHANEL

```
#define XB_AT_E_CHANEL -406
```

Définition à la ligne 81 du fichier define.h.

5.1.2.25 XB_AT_E_COORDINATOR

```
#define XB_AT_E_COORDINATOR -408
```

Définition à la ligne 83 du fichier define.h.

5.1.2.26 XB_AT_E_DISCOVER_NETWORK

```
#define XB_AT_E_DISCOVER_NETWORK -414
```

Définition à la ligne 89 du fichier define.h.

5.1.2.27 XB_AT_E_ENTER

```
#define XB_AT_E_ENTER -401
```

Définition à la ligne 76 du fichier define.h.

5.1.2.28 XB_AT_E_EXIT

```
#define XB_AT_E_EXIT -412
```

Définition à la ligne 87 du fichier define.h.

5.1.2.29 XB_AT_E_LOW_DEST_ADDR

```
#define XB_AT_E_LOW_DEST_ADDR -411
```

Définition à la ligne 86 du fichier define.h.

5.1.2.30 XB_AT_E_PAN_ID

```
#define XB_AT_E_PAN_ID -407
```

Définition à la ligne 82 du fichier define.h.

5.1.2.31 XB_AT_E_PARITY

```
#define XB_AT_E_PARITY -409
```

Définition à la ligne 84 du fichier define.h.

5.1.2.32 XB_AT_E_SUCCESS

```
#define XB_AT_E_SUCCESS 400
```

Définition à la ligne 75 du fichier define.h.

5.1.2.33 XB_AT_E_WRITE_CONFIG

```
#define XB_AT_E_WRITE_CONFIG -413
```

Définition à la ligne 88 du fichier define.h.

5.1.2.34 XB_AT_M_GET

```
#define XB_AT_M_GET 1
```

Définition à la ligne 139 du fichier define.h.

5.1.2.35 XB_AT_M_SET

```
#define XB_AT_M_SET 2
```

Définition à la ligne 140 du fichier define.h.

5.1.2.36 XB_AT_R_EMPTY

```
#define XB_AT_R_EMPTY ""
```

Définition à la ligne 134 du fichier define.h.

5.1.2.37 XB_AT_R_ERROR

```
#define XB_AT_R_ERROR "ERROR\r"
```

Définition à la ligne 136 du fichier define.h.

5.1.2.38 XB_AT_R_SUCCESS

```
#define XB_AT_R_SUCCESS "OK\r"
```

Définition à la ligne 135 du fichier define.h.

5.1.2.39 XB_AT_V_16BIT_SOURCE_ADDR

```
#define XB_AT_V_16BIT_SOURCE_ADDR "1\r"
```

Définition à la ligne 129 du fichier define.h.

5.1.2.40 XB_AT_V_AES

```
#define XB_AT_V_AES "1\r"
```

Définition à la ligne 123 du fichier define.h.

5.1.2.41 XB_AT_V_AES_KEY

```
#define XB_AT_V_AES_KEY "32303032\r"
```

Définition à la ligne 124 du fichier define.h.

5.1.2.42 XB_AT_V_API

```
#define XB_AT_V_API "0\r"
```

Définition à la ligne 121 du fichier define.h.

5.1.2.43 XB_AT_V_BAUDRATE

```
#define XB_AT_V_BAUDRATE "3\r"
```

Définition à la ligne 122 du fichier define.h.

5.1.2.44 XB_AT_V_CHANEL

```
#define XB_AT_V_CHANEL "C\r"
```

Définition à la ligne 125 du fichier define.h.

5.1.2.45 XB_AT_V_COORDINATOR

```
#define XB_AT_V_COORDINATOR "0\r"
```

Définition à la ligne 127 du fichier define.h.

5.1.2.46 XB_AT_V_DISCOVER_NETWORK

```
#define XB_AT_V_DISCOVER_NETWORK ""
```

Définition à la ligne 131 du fichier define.h.

5.1.2.47 XB_AT_V_END_LINE

```
#define XB_AT_V_END_LINE "\r"
```

Définition à la ligne 120 du fichier define.h.

5.1.2.48 XB_AT_V_LOW_DEST_ADDR

```
#define XB_AT_V_LOW_DEST_ADDR "2\r"
```

Définition à la ligne 130 du fichier define.h.

5.1.2.49 XB_AT_V_PAN_ID

```
#define XB_AT_V_PAN_ID "3332\r"
```

Définition à la ligne 126 du fichier define.h.

5.1.2.50 XB_AT_V_PARITY

```
#define XB_AT_V_PARITY "1\r"
```

Définition à la ligne 128 du fichier define.h.

5.1.2.51 XB_BAUDRATE_DEFAULT

```
#define XB_BAUDRATE_DEFAULT 9600
```

Définition à la ligne 22 du fichier define.h.

5.1.2.52 XB_BAUDRATE_PRIMARY

```
#define XB_BAUDRATE_PRIMARY 9600
```

Définition à la ligne 14 du fichier define.h.

5.1.2.53 XB_DATABITS_DEFAULT

```
#define XB_DATABITS_DEFAULT SERIAL_DATABITS_8
```

Définition à la ligne 23 du fichier define.h.

5.1.2.54 XB_DATABITS_PRIMARY

```
#define XB_DATABITS_PRIMARY SERIAL_DATABITS_8
```

Définition à la ligne 15 du fichier define.h.

5.1.2.55 XB_E_SUCCESS

```
#define XB_E_SUCCESS 000
```

Définition à la ligne 49 du fichier define.h.

5.1.2.56 XB_FCT_E_NONE_REACHABLE

```
#define XB_FCT_E_NONE_REACHABLE -102
```

Définition à la ligne 54 du fichier define.h.

5.1.2.57 XB_FCT_E_NOT_FOUND

```
#define XB_FCT_E_NOT_FOUND -101
```

Définition à la ligne 53 du fichier define.h.

5.1.2.58 XB_FCT_E_SUCCESS

```
#define XB_FCT_E_SUCCESS 100
```

Définition à la ligne 52 du fichier define.h.

5.1.2.59 XB_FCT_TEST_ALIVE

```
#define XB_FCT_TEST_ALIVE 0x1E
```

Définition à la ligne 44 du fichier define.h.

5.1.2.60 XB_LIST_ADR

```
#define XB_LIST_ADR (int[]){XB_ADR_BROADCAST, XB_ADR_ROBOT_01, XB_ADR_ROBOT_02}
```

Définition à la ligne 35 du fichier define.h.

5.1.2.61 XB_LIST_CODE_FCT

```
#define XB_LIST_CODE_FCT (int[]){XB_FCT_TEST_ALIVE}
```

Définition à la ligne 46 du fichier define.h.

5.1.2.62 XB_PARITY_DEFAULT

```
#define XB_PARITY_DEFAULT SERIAL_PARITY_NONE
```

Définition à la ligne 24 du fichier define.h.

5.1.2.63 XB_PARITY_PRIMARY

```
#define XB_PARITY_PRIMARY SERIAL_PARITY_EVEN
```

Définition à la ligne 16 du fichier define.h.

5.1.2.64 XB_SER_E_CONFIG

```
#define XB_SER_E_CONFIG -505
```

Définition à la ligne 97 du fichier define.h.

5.1.2.65 XB_SER_E_NOT_FOUND

```
#define XB_SER_E_NOT_FOUND -501
```

Définition à la ligne 93 du fichier define.h.

5.1.2.66 XB_SER_E_OPEN

```
#define XB_SER_E_OPEN -502
```

Définition à la ligne 94 du fichier define.h.

5.1.2.67 XB_SER_E_PARAM

```
#define XB_SER_E_PARAM -503
```

Définition à la ligne 95 du fichier define.h.

5.1.2.68 XB_SER_E_SUCCESS

```
#define XB_SER_E_SUCCESS 500
```

Définition à la ligne 92 du fichier define.h.

5.1.2.69 XB_SER_E_TIMOUT

```
#define XB_SER_E_TIMOUT -506
```

Définition à la ligne 98 du fichier define.h.

5.1.2.70 XB_SER_E_UKN_BAUDRATE

```
#define XB_SER_E_UKN_BAUDRATE -504
```

Définition à la ligne 96 du fichier define.h.

5.1.2.71 XB_SER_E_UKN_DATABITS

```
#define XB_SER_E_UKN_DATABITS -507
```

Définition à la ligne 99 du fichier define.h.

5.1.2.72 XB_SER_E_UKN_PARITY

```
#define XB_SER_E_UKN_PARITY -509
```

Définition à la ligne 101 du fichier define.h.

5.1.2.73 XB_SER_E_UKN_STOPBITS

```
#define XB_SER_E_UKN_STOPBITS -508
```

Définition à la ligne 100 du fichier define.h.

5.1.2.74 XB_SERIAL_PORT_DEFAULT

```
#define XB_SERIAL_PORT_DEFAULT "/dev/ttyAMA0"
```

Définition à la ligne 21 du fichier define.h.

5.1.2.75 XB_SERIAL_PORT_PRIMARY

```
#define XB_SERIAL_PORT_PRIMARY "/dev/ttyAMA0"
```

Définition à la ligne 13 du fichier define.h.

5.1.2.76 XB_STOPBITS_DEFAULT

```
#define XB_STOPBITS_DEFAULT SERIAL_STOPBITS_1
```

Définition à la ligne 25 du fichier define.h.

5.1.2.77 XB_STOPBITS_PRIMARY

```
#define XB_STOPBITS_PRIMARY SERIAL_STOPBITS_1
```

Définition à la ligne 17 du fichier define.h.

5.1.2.78 XB_SUB_TRAME_E_DECOUPAGE

```
#define XB_SUB_TRAME_E_DECOUPAGE -303
```

Définition à la ligne 69 du fichier define.h.

5.1.2.79 XB_SUB_TRAME_E_END

```
#define XB_SUB_TRAME_E_END -305
```

Définition à la ligne 71 du fichier define.h.

5.1.2.80 XB_SUB_TRAME_E_NULL

```
#define XB_SUB_TRAME_E_NULL -306
```

Définition à la ligne 72 du fichier define.h.

5.1.2.81 XB_SUB_TRAME_E_REPARTITION

```
#define XB_SUB_TRAME_E_REPARTITION -302
```

Définition à la ligne 68 du fichier define.h.

5.1.2.82 XB_SUB_TRAME_E_SIZE

```
#define XB_SUB_TRAME_E_SIZE -301
```

Définition à la ligne 67 du fichier define.h.

5.1.2.83 XB_SUB_TRAME_E_START

```
#define XB_SUB_TRAME_E_START -304
```

Définition à la ligne 70 du fichier define.h.

5.1.2.84 XB_SUB_TRAME_E_SUCCESS

```
#define XB_SUB_TRAME_E_SUCCESS 300
```

Définition à la ligne 66 du fichier define.h.

5.1.2.85 XB_TRAME_E_CRC

```
#define XB_TRAME_E_CRC -204
```

Définition à la ligne 61 du fichier define.h.

5.1.2.86 XB_TRAME_E_DEST

```
#define XB_TRAME_E_DEST -206
```

Définition à la ligne 63 du fichier define.h.

5.1.2.87 XB_TRAME_E_END

```
#define XB_TRAME_E_END -203
```

Définition à la ligne 60 du fichier define.h.

5.1.2.88 XB_TRAME_E_EXP

```
#define XB_TRAME_E_EXP -205
```

Définition à la ligne 62 du fichier define.h.

5.1.2.89 XB_TRAME_E_SIZE

```
#define XB_TRAME_E_SIZE -201
```

Définition à la ligne 58 du fichier define.h.

5.1.2.90 XB_TRAME_E_START

```
#define XB_TRAME_E_START -202
```

Définition à la ligne 59 du fichier define.h.

5.1.2.91 XB_TRAME_E_SUCCESS

```
#define XB_TRAME_E_SUCCESS 200
```

Définition à la ligne 57 du fichier define.h.

5.1.2.92 XB_V_ACK

```
#define XB_V_ACK 0x06
```

Définition à la ligne 27 du fichier define.h.

5.1.2.93 XB_V_END

```
#define XB_V_END 0x03
```

Définition à la ligne 41 du fichier define.h.

5.1.2.94 XB_V_NACK

```
#define XB_V_NACK 0x15
```

Définition à la ligne 28 du fichier define.h.

5.1.2.95 XB_V_START

```
#define XB_V_START 0x02
```

Définition à la ligne 40 du fichier define.h.

5.2 define.h

Aller à la documentation de ce fichier.

```
00001
00008 #ifndef DEFINE_XBEE_H
00009 #define DEFINE_XBEE_H
00010
00011 // Paramètres du port série
00012 // Configuration des modules XBee pour la compétition
00013 #define XB_SERIAL_PORT_PRIMARY "/dev/ttyAMA0"
00014 #define XB_BAUDRATE_PRIMARY 9600
00015 #define XB_DATABITS_PRIMARY SERIAL_DATABITS_8
00016 #define XB_PARITY_PRIMARY SERIAL_PARITY_EVEN
00017 #define XB_STOPBITS_PRIMARY SERIAL_STOPBITS_1
00018
00019
00020 // Configuration d'usine par défaut des modules XBee neufs
00021 #define XB_SERIAL_PORT_DEFAULT "/dev/ttyAMA0"
00022 #define XB_BAUDRATE_DEFAULT 9600
00023 #define XB_DATABITS_DEFAULT SERIAL_DATABITS_8
00024 #define XB_PARITY_DEFAULT SERIAL_PARITY_NONE
00025 #define XB_STOPBITS_DEFAULT SERIAL_STOPBITS_1
00026
00027 #define XB_V_ACK 0x06
00028 #define XB_V_NACK 0x15
00029
00030 // Addresses des robots
00031 #define XB_ADR_BROADCAST 0x11
00032 #define XB_ADR_ROBOT_01 0x12
00033 #define XB_ADR_ROBOT_02 0x13
00034
00035 #define XB_LIST_ADR (int[]){XB_ADR_BROADCAST, XB_ADR_ROBOT_01, XB_ADR_ROBOT_02}
00036
00037 #define XB_ADR_CURRENT_ROBOT XB_ADR_ROBOT_01
00038
00039 // Paramètres de la trame message
00040 #define XB_V_START 0x02
00041 #define XB_V_END 0x03
00042
00043 // Codes fonctions
00044 #define XB_FCT_TEST_ALIVE 0x1E
00045
00046 #define XB_LIST_CODE_FCT (int[]){XB_FCT_TEST_ALIVE}
00047
00048 // Code erreurs généraux
00049 #define XB_E_SUCCESS 000
00050
00051 // Codes erreurs des codes fonctions
00052 #define XB_FCT_E_SUCCESS 100
00053 #define XB_FCT_E_NOT_FOUND -101
00054 #define XB_FCT_E_NONE_REACHABLE -102
00055
00056 // Codes erreurs traitement de trame
00057 #define XB_TRAME_E_SUCCESS 200
00058 #define XB_TRAME_E_SIZE -201
00059 #define XB_TRAME_E_START -202
00060 #define XB_TRAME_E_END -203
00061 #define XB_TRAME_E_CRC -204
```

```
00062 #define XB_TRAME_E_EXP -205
00063 #define XB_TRAME_E_DEST -206
00064
00065 // Codes erreurs découpage de trame
00066 #define XB_SUB_TRAME_E_SUCCESS 300
00067 #define XB_SUB_TRAME_E_SIZE -301
00068 #define XB_SUB_TRAME_E_REPARTITION -302
00069 #define XB_SUB_TRAME_E_DECOUPAGE -303
00070 #define XB_SUB_TRAME_E_START -304
00071 #define XB_SUB_TRAME_E_END -305
00072 #define XB_SUB_TRAME_E_NULL -306
00073
00074 // Codes d'erreurs AT
00075 #define XB_AT_E_SUCCESS 400
00076 #define XB_AT_E_ENTER -401
00077 #define XB_AT_E_API -402
00078 #define XB_AT_E_BAUDRATE -403
00079 #define XB_AT_E_AES -404
00080 #define XB_AT_E_AES_KEY -405
00081 #define XB_AT_E_CHANEL -406
00082 #define XB_AT_E_PAN_ID -407
00083 #define XB_AT_E_COORDINATOR -408
00084 #define XB_AT_E_PARITY -409
00085 #define XB_AT_E_16BIT_SOURCE_ADDR -410
00086 #define XB_AT_E_LOW_DEST_ADDR -411
00087 #define XB_AT_E_EXIT -412
00088 #define XB_AT_E_WRITE_CONFIG -413
00089 #define XB_AT_E_DISCOVER_NETWORK -414
00090
00091 // Codes d'erreurs ouverture connexion série
00092 #define XB_SER_E_SUCCESS 500
00093 #define XB_SER_E_NOT_FOUND -501
00094 #define XB_SER_E_OPEN -502
00095 #define XB_SER_E_PARAM -503
00096 #define XB_SER_E_UKN_BAUDRATE -504
00097 #define XB_SER_E_CONFIG -505
00098 #define XB_SER_E_TIMOUT -506
00099 #define XB_SER_E_UKN_DATABITS -507
00100 #define XB_SER_E_UKN_STOPBITS -508
00101 #define XB_SER_E_UKN_PARITY -509
00102
00103 // Commandes AT
00104 #define XB_AT_CMD_ENTER "+++"
00105 #define XB_AT_CMD_EXIT "ATCN"
00106 #define XB_AT_CMD_WRITE_CONFIG "ATWR"
00107 #define XB_AT_CMD_API "ATAP"
00108 #define XB_AT_CMD_BAUDRATE "ATBD"
00109 #define XB_AT_CMD_AES "ATEE"
00110 #define XB_AT_CMD_AES_KEY "ATKY"
00111 #define XB_AT_CMD_CHANEL "ATCH"
00112 #define XB_AT_CMD_PAN_ID "ATID"
00113 #define XB_AT_CMD_COORDINATOR "ATCE"
00114 #define XB_AT_CMD_PARITY "ATNB"
00115 #define XB_AT_CMD_16BIT_SOURCE_ADDR "ATMY"
00116 #define XB_AT_CMD_LOW_DEST_ADDR "ATDL"
00117 #define XB_AT_CMD_DISCOVER_NETWORK "ATND"
00118
00119 // Valeurs AT
00120 #define XB_AT_V_END_LINE "\r"
00121 #define XB_AT_V_API "0\r"
00122 #define XB_AT_V_BAUDRATE "3\r"
00123 #define XB_AT_V_AES "1\r"
00124 #define XB_AT_V_AES_KEY "32303032\r"
00125 #define XB_AT_V_CHANEL "C\r"
00126 #define XB_AT_V_PAN_ID "3332\r"
00127 #define XB_AT_V_COORDINATOR "0\r"
00128 #define XB_AT_V_PARITY "1\r"
00129 #define XB_AT_V_16BIT_SOURCE_ADDR "1\r"
00130 #define XB_AT_V_LOW_DEST_ADDR "2\r"
00131 #define XB_AT_V_DISCOVER_NETWORK ""
00132
00133 // Réponses AT
00134 #define XB_AT_R_EMPTY ""
00135 #define XB_AT_R_SUCCESS "OK\r"
00136 #define XB_AT_R_ERROR "ERROR\r"
00137
00138 // Mode AT
00139 #define XB_AT_M_GET 1
00140 #define XB_AT_M_SET 2
00141 #endif
```

5.3 Référence du fichier loglib.cpp

```
#include "loglib.h"
#include <iostream>
#include <sstream>
#include <string>
```

Fonctions

— char * stringToChar (std::string chaine)
— Log & operator<< (Log &log, Mendl const &data)

5.3.1 Documentation des fonctions

5.3.1.1 operator<<()

```
Log & operator<< (
            Log & log,
            Mendl const & data )
```

Définition à la ligne 20 du fichier loglib.cpp.

```
00020                                           {
00021     time_t now = time(0);
00022     tm *ltm = localtime(&now);
00023     cout « endl;
00024     stringstream cmd;
00025     cmd « "echo \"" « "["«ltm->tm_hour « ":" « ltm->tm_min « ":" « ltm->tm_sec « " - "« log.name «"] "
      «log.ss.str()«"\" » log.log";
00026     log.ss.str("");
00027     system(stringToChar(cmd.str()));
00028
00029     return log;
00030
00031 }
```

5.3.1.2 stringToChar()

```
char * stringToChar (
            std::string chaine )
```

Définition à la ligne 8 du fichier loglib.cpp.

```
00008                                   {
00009     char* message = strcpy(new char[chaine.size() + 1], chaine.c_str());
00010     return message;
00011 }
```

5.4 loglib.cpp

Aller à la documentation de ce fichier.

```
00001 #include "loglib.h"
00002 #include <iostream>
00003 #include <sstream>
00004 #include <string>
00005
00006 using namespace std;
00007
00008 char* stringToChar(std::string chaine){
00009     char* message = strcpy(new char[chaine.size() + 1], chaine.c_str());
00010     return message;
00011 }
00012
00013
00014 Log::Log(string nom){
00015     name = nom;
00016     stringstream ss;
00017 }
00018
00019
00020 Log& operator«(Log &log, Mendl const& data){
00021     time_t now = time(0);
00022     tm *ltm = localtime(&now);
00023     cout « endl;
00024     stringstream cmd;
00025     cmd « "echo \"" « "["«ltm->tm_hour « ":" « ltm->tm_min « ":" « ltm->tm_sec « " - "« log.name «"] "
      «log.ss.str()«"\" » log.log";
00026     log.ss.str("");
00027     system(stringToChar(cmd.str()));
00028
00029     return log;
00030
00031 }
00032
```

5.5 Référence du fichier loglib.h

```
#include <ostream>
#include <iostream>
#include <string>
#include <sstream>
#include <cstring>
```

Classes

- struct Mendl
- class Log

Fonctions

- char * stringToChar (std::string chaine)
- template<typename T >
  Log & operator<< (Log &log, T const &data)

Variables

- const Mendl mendl

5.5.1 Documentation des fonctions

5.5.1.1 operator<<()

```
template<typename T >
Log & operator<< (
            Log & log,
            T const & data )
```

Définition à la ligne 34 du fichier loglib.h.

```
00035 {
00036     log.ss « data;
00037     std::cout « data;
00038     return log;
00039 }
```

5.5.1.2 stringToChar()

```
char * stringToChar (
            std::string chaine )
```

Définition à la ligne 8 du fichier loglib.cpp.

```
00008                                   {
00009     char* message = strcpy(new char[chaine.size() + 1], chaine.c_str());
00010     return message;
00011 }
```

5.5.2 Documentation des variables

5.5.2.1 mendl

```
const Mendl mendl
```

Définition à la ligne 13 du fichier loglib.h.

5.6 loglib.h

Aller à la documentation de ce fichier.

```
00001 #ifndef LOGLIB_H
00002 #define LOGLIB_H
00003
00004 #include <ostream>
00005 #include <iostream>
00006 #include <string>
00007 #include <sstream>
00008 #include <cstring>
00009
00010 struct Mendl{
00011 };
00012
00013 const Mendl mendl;
00014
00015 char* stringToChar(std::string chaine);
00016
00017
00018 class Log : public std::ostream{
00019     private:
00020         std::stringstream ss;
00021
00022     public:
00023         std::string name;
00024         Log(std::string nom);
00025         int save(int data);
00026         template<typename T>
00027         friend Log& operator«(Log& log, const T &classObj);
00028         friend Log& operator«(Log& log, const Mendl& data);
00029
00030 };
00031
00032
00033 template <typename T>
00034 Log& operator«(Log &log, T const &data)
00035 {
00036     log.ss « data;
00037     std::cout « data;
00038     return log;
00039 }
00040
00041 #endif
```

5.7 Référence du fichier main.cpp

```
#include "xbeelib.h"
```

Fonctions

— int main (int argc, char ∗argv[ ])

5.7.1 Documentation des fonctions

5.7.1.1 main()

```
int main (
              int argc,
              char * argv[] )
```

Définition à la ligne 5 du fichier main.cpp.

```
00005                                 {
00006
00007     XBee xbee;
00008     int status = xbee.openSerialConnection();
00009
00010     if(status != 1)
00011         return 0;
00012
00013     thread heartbeat(&XBee::sendHeartbeat, xbee);
00014     thread t(&XBee::waitForATrame, xbee);
00015     while(true){}
00016
00017     xbee.closeSerialConnection();
00018     return XB_E_SUCCESS;
00019 }
```

5.8 main.cpp

Aller à la documentation de ce fichier.

```
00001 #include "xbeelib.h"
00002
00003 using namespace std;
00004
00005 int main(int argc, char *argv[]){
00006
00007     XBee xbee;
00008     int status = xbee.openSerialConnection();
00009
00010     if(status != 1)
00011         return 0;
00012
00013     thread heartbeat(&XBee::sendHeartbeat, xbee);
00014     thread t(&XBee::waitForATrame, xbee);
00015     while(true){}
00016
00017     xbee.closeSerialConnection();
00018     return XB_E_SUCCESS;
00019 }
```

5.9 Référence du fichier serialib.cpp

Source file of the class serialib. This class is used for communication over a serial device.

```
#include "serialib.h"
#include "define.h"
```

5.9.1 Description détaillée

Source file of the class serialib. This class is used for communication over a serial device.

Auteur

Philippe Lucidarme (University of Angers)

Version

2.0

Date

december the 27th of 2019

Définition dans le fichier serialib.cpp.

5.10 serialib.cpp

Aller à la documentation de ce fichier.

```
00001
00018 #include "serialib.h"
00019 #include "define.h"
00020
00021
00022
00023 //_____________________________________
00024 // ::: Constructors and destructors :::
00025
00026
00030 serialib::serialib()
00031 {
00032 #if defined (_WIN32) || defined( _WIN64)
00033     // Set default value for RTS and DTR (Windows only)
00034     currentStateRTS=true;
00035     currentStateDTR=true;
00036     hSerial = INVALID_HANDLE_VALUE;
00037 #endif
00038 #if defined (__linux__) || defined(__APPLE__)
00039     fd = -1;
00040 #endif
00041 }
00042
00043
00047 // Class desctructor
00048 serialib::~serialib()
00049 {
00050     closeDevice();
00051 }
00052
00053
00054
00055 //_____________________________________________
00056 // ::: Configuration and initialization :::
00057
00058
00059
00129 int serialib::openDevice(const char *Device, const unsigned int Bauds,
00130                          SerialDataBits Databits,
00131                          SerialParity Parity,
00132                          SerialStopBits Stopbits) {
00133 #if defined (_WIN32) || defined( _WIN64)
00134     // Open serial port
00135     hSerial = CreateFileA(Device,GENERIC_READ |
       GENERIC_WRITE,0,0,OPEN_EXISTING,/*FILE_ATTRIBUTE_NORMAL*/0,0);
00136     if(hSerial==INVALID_HANDLE_VALUE) {
00137         if(GetLastError()==ERROR_FILE_NOT_FOUND)
00138             return XB_SER_E_NOT_FOUND; // Device not found
00139
00140         // Error while opening the device
00141         return XB_SER_E_OPEN;
```

```
00142     }
00143
00144     // Set parameters
00145
00146     // Structure for the port parameters
00147     DCB dcbSerialParams;
00148     dcbSerialParams.DCBlength=sizeof(dcbSerialParams);
00149
00150     // Get the port parameters
00151     if (!GetCommState(hSerial, &dcbSerialParams)) return XB_SER_E_PARAM;
00152
00153     // Set the speed (Bauds)
00154     switch (Bauds)
00155     {
00156     case 110  :     dcbSerialParams.BaudRate=CBR_110; break;
00157     case 300  :     dcbSerialParams.BaudRate=CBR_300; break;
00158     case 600  :     dcbSerialParams.BaudRate=CBR_600; break;
00159     case 1200 :     dcbSerialParams.BaudRate=CBR_1200; break;
00160     case 2400 :     dcbSerialParams.BaudRate=CBR_2400; break;
00161     case 4800 :     dcbSerialParams.BaudRate=CBR_4800; break;
00162     case 9600 :     dcbSerialParams.BaudRate=CBR_9600; break;
00163     case 14400 :    dcbSerialParams.BaudRate=CBR_14400; break;
00164     case 19200 :    dcbSerialParams.BaudRate=CBR_19200; break;
00165     case 38400 :    dcbSerialParams.BaudRate=CBR_38400; break;
00166     case 56000 :    dcbSerialParams.BaudRate=CBR_56000; break;
00167     case 57600 :    dcbSerialParams.BaudRate=CBR_57600; break;
00168     case 115200 :   dcbSerialParams.BaudRate=CBR_115200; break;
00169     case 128000 :   dcbSerialParams.BaudRate=CBR_128000; break;
00170     case 256000 :   dcbSerialParams.BaudRate=CBR_256000; break;
00171     default : return XB_SER_E_UKN_BAUDRATE;
00172     }
00173     //select data size
00174     BYTE bytesize = 0;
00175     switch(Databits) {
00176         case SERIAL_DATABITS_5: bytesize = 5; break;
00177         case SERIAL_DATABITS_6: bytesize = 6; break;
00178         case SERIAL_DATABITS_7: bytesize = 7; break;
00179         case SERIAL_DATABITS_8: bytesize = 8; break;
00180         case SERIAL_DATABITS_16: bytesize = 16; break;
00181         default: return XB_SER_E_UKN_DATABITS;
00182     }
00183     BYTE stopBits = 0;
00184     switch(Stopbits) {
00185         case SERIAL_STOPBITS_1: stopBits = ONESTOPBIT; break;
00186         case SERIAL_STOPBITS_1_5: stopBits = ONE5STOPBITS; break;
00187         case SERIAL_STOPBITS_2: stopBits = TWOSTOPBITS; break;
00188         default: return XB_SER_E_UKN_STOPBITS;
00189     }
00190     BYTE parity = 0;
00191     switch(Parity) {
00192         case SERIAL_PARITY_NONE: parity = NOPARITY; break;
00193         case SERIAL_PARITY_EVEN: parity = EVENPARITY; break;
00194         case SERIAL_PARITY_ODD: parity = ODDPARITY; break;
00195         case SERIAL_PARITY_MARK: parity = MARKPARITY; break;
00196         case SERIAL_PARITY_SPACE: parity = SPACEPARITY; break;
00197         default: return XB_SER_E_UKN_PARITY;
00198     }
00199     // configure byte size
00200     dcbSerialParams.ByteSize = bytesize;
00201     // configure stop bits
00202     dcbSerialParams.StopBits = stopBits;
00203     // configure parity
00204     dcbSerialParams.Parity = parity;
00205
00206     // Write the parameters
00207     if(!SetCommState(hSerial, &dcbSerialParams)) return XB_SER_E_CONFIG;
00208
00209     // Set TimeOut
00210
00211     // Set the Timeout parameters
00212     timeouts.ReadIntervalTimeout=0;
00213     // No TimeOut
00214     timeouts.ReadTotalTimeoutConstant=MAXDWORD;
00215     timeouts.ReadTotalTimeoutMultiplier=0;
00216     timeouts.WriteTotalTimeoutConstant=MAXDWORD;
00217     timeouts.WriteTotalTimeoutMultiplier=0;
00218
00219     // Write the parameters
00220     if(!SetCommTimeouts(hSerial, &timeouts)) return XB_SER_E_TIMOUT;
00221
00222     // Opening successfull
00223     return 1;
00224 #endif
00225 #if defined (__linux__) || defined(__APPLE__)
00226     // Structure with the device's options
00227     struct termios options;
00228
```

```
00229
00230     // Open device
00231     fd = open(Device, O_RDWR | O_NOCTTY | O_NDELAY);
00232     // If the device is not open, return -1
00233     if (fd == -1) return XB_SER_E_OPEN;
00234     // Open the device in nonblocking mode
00235     fcntl(fd, F_SETFL, FNDELAY);
00236
00237
00238     // Get the current options of the port
00239     tcgetattr(fd, &options);
00240     // Clear all the options
00241     bzero(&options, sizeof(options));
00242
00243     // Prepare speed (Bauds)
00244     speed_t         Speed;
00245     switch (Bauds)
00246     {
00247     case 110  :     Speed=B110; break;
00248     case 300  :     Speed=B300; break;
00249     case 600  :     Speed=B600; break;
00250     case 1200 :     Speed=B1200; break;
00251     case 2400 :     Speed=B2400; break;
00252     case 4800 :     Speed=B4800; break;
00253     case 9600 :     Speed=B9600; break;
00254     case 19200 :    Speed=B19200; break;
00255     case 38400 :    Speed=B38400; break;
00256     case 57600 :    Speed=B57600; break;
00257     case 115200 :   Speed=B115200; break;
00258     default : return XB_SER_E_UKN_BAUDRATE;
00259     }
00260     int databits_flag = 0;
00261     switch(Databits) {
00262         case SERIAL_DATABITS_5: databits_flag = CS5; break;
00263         case SERIAL_DATABITS_6: databits_flag = CS6; break;
00264         case SERIAL_DATABITS_7: databits_flag = CS7; break;
00265         case SERIAL_DATABITS_8: databits_flag = CS8; break;
00266         //16 bits and everything else not supported
00267         default: return XB_SER_E_UKN_DATABITS;
00268     }
00269     int stopbits_flag = 0;
00270     switch(Stopbits) {
00271         case SERIAL_STOPBITS_1: stopbits_flag = 0; break;
00272         case SERIAL_STOPBITS_2: stopbits_flag = CSTOPB; break;
00273         //1.5 stopbits and everything else not supported
00274         default: return XB_SER_E_UKN_STOPBITS;
00275     }
00276     int parity_flag = 0;
00277     switch(Parity) {
00278         case SERIAL_PARITY_NONE: parity_flag = 0; break;
00279         case SERIAL_PARITY_EVEN: parity_flag = PARENB; break;
00280         case SERIAL_PARITY_ODD: parity_flag = (PARENB | PARODD); break;
00281         //mark and space parity not supported
00282         default: return XB_SER_E_UKN_PARITY;
00283     }
00284
00285     // Set the baud rate
00286     cfsetispeed(&options, Speed);
00287     cfsetospeed(&options, Speed);
00288     // Configure the device : data bits, stop bits, parity, no control flow
00289     // Ignore modem control lines (CLOCAL) and Enable receiver (CREAD)
00290     options.c_cflag |= ( CLOCAL | CREAD | databits_flag | parity_flag | stopbits_flag);
00291     options.c_iflag |= ( IGNPAR | IGNBRK );
00292     // Timer unused
00293     options.c_cc[VTIME]=0;
00294     // At least on character before satisfy reading
00295     options.c_cc[VMIN]=0;
00296     // Activate the settings
00297     tcsetattr(fd, TCSANOW, &options);
00298     // Success
00299     return (XB_SER_E_SUCCESS);
00300 #endif
00301
00302 }
00303
00304 bool serialib::isDeviceOpen()
00305 {
00306 #if defined (_WIN32) || defined( _WIN64)
00307     return hSerial != INVALID_HANDLE_VALUE;
00308 #endif
00309 #if defined (__linux__) || defined(__APPLE__)
00310     return fd >= 0;
00311 #endif
00312 }
00313
00317 void serialib::closeDevice()
00318 {
```

```
00319 #if defined (_WIN32) || defined( _WIN64)
00320     CloseHandle(hSerial);
00321     hSerial = INVALID_HANDLE_VALUE;
00322 #endif
00323 #if defined (__linux__) || defined(__APPLE__)
00324     close (fd);
00325     fd = -1;
00326 #endif
00327 }
00328
00329
00330
00331
00332 //_____________________________________________
00333 // ::: Read/Write operation on characters :::
00334
00335
00336
00343 char serialib::writeChar(const char Byte)
00344 {
00345 #if defined (_WIN32) || defined( _WIN64)
00346     // Number of bytes written
00347     DWORD dwBytesWritten;
00348     // Write the char to the serial device
00349     // Return -1 if an error occured
00350     if(!WriteFile(hSerial,&Byte,1,&dwBytesWritten,NULL)) return -1;
00351     // Write operation successfull
00352     return 1;
00353 #endif
00354 #if defined (__linux__) || defined(__APPLE__)
00355     // Write the char
00356     if (write(fd,&Byte,1)!=1) return -1;
00357
00358     // Write operation successfull
00359     return 1;
00360 #endif
00361 }
00362
00363
00364
00365 //_____________________________________________
00366 // ::: Read/Write operation on strings :::
00367
00368
00375 char serialib::writeString(const char *receivedString)
00376 {
00377 #if defined (_WIN32) || defined( _WIN64)
00378     // Number of bytes written
00379     DWORD dwBytesWritten;
00380     // Write the string
00381     if(!WriteFile(hSerial,receivedString,strlen(receivedString),&dwBytesWritten,NULL))
00382         // Error while writing, return -1
00383         return -1;
00384     // Write operation successfull
00385     return 1;
00386 #endif
00387 #if defined (__linux__) || defined(__APPLE__)
00388     // Lenght of the string
00389     int Lenght=strlen(receivedString);
00390     // Write the string
00391     if (write(fd,receivedString,Lenght)!=Lenght) return -1;
00392     // Write operation successfull
00393     return 1;
00394 #endif
00395 }
00396
00397 // _____________________________________
00398 // ::: Read/Write operation on bytes :::
00399
00400
00401
00409 char serialib::writeBytes(const void *Buffer, const unsigned int NbBytes)
00410 {
00411 #if defined (_WIN32) || defined( _WIN64)
00412     // Number of bytes written
00413     DWORD dwBytesWritten;
00414     // Write data
00415     if(!WriteFile(hSerial, Buffer, NbBytes, &dwBytesWritten, NULL))
00416         // Error while writing, return -1
00417         return -1;
00418     // Write operation successfull
00419     return 1;
00420 #endif
00421 #if defined (__linux__) || defined(__APPLE__)
00422     // Write data
00423     if (write (fd,Buffer,NbBytes)!=(ssize_t)NbBytes) return -1;
00424     // Write operation successfull
```

```
00425     return 1;
00426 #endif
00427 }
00428
00429
00430
00441 char serialib::readChar(char *pByte,unsigned int timeOut_ms)
00442 {
00443 #if defined (_WIN32) || defined(_WIN64)
00444     // Number of bytes read
00445     DWORD dwBytesRead = 0;
00446
00447     // Set the TimeOut
00448     timeouts.ReadTotalTimeoutConstant=timeOut_ms;
00449
00450     // Write the parameters, return -1 if an error occured
00451     if(!SetCommTimeouts(hSerial, &timeouts)) return -1;
00452
00453     // Read the byte, return -2 if an error occured
00454     if(!ReadFile(hSerial,pByte, 1, &dwBytesRead, NULL)) return -2;
00455
00456     // Return 0 if the timeout is reached
00457     if (dwBytesRead==0) return 0;
00458
00459     // The byte is read
00460     return 1;
00461 #endif
00462 #if defined (__linux__) || defined(__APPLE__)
00463     // Timer used for timeout
00464     timeOut         timer;
00465     // Initialise the timer
00466     timer.initTimer();
00467     // While Timeout is not reached
00468     while (timer.elapsedTime_ms()<timeOut_ms || timeOut_ms==0)
00469     {
00470         // Try to read a byte on the device
00471         switch (read(fd,pByte,1)) {
00472         case 1  : return 1; // Read successfull
00473         case -1 : return -2; // Error while reading
00474         }
00475     }
00476     return 0;
00477 #endif
00478 }
00479
00480
00481
00492 int serialib::readStringNoTimeOut(char *receivedString,char finalChar,unsigned int maxNbBytes)
00493 {
00494     // Number of characters read
00495     unsigned int    NbBytes=0;
00496     // Returned value from Read
00497     char            charRead;
00498
00499     // While the buffer is not full
00500     while (NbBytes<maxNbBytes)
00501     {
00502         // Read a character with the restant time
00503         charRead=readChar(&receivedString[NbBytes]);
00504
00505         // Check a character has been read
00506         if (charRead==1)
00507         {
00508             // Check if this is the final char
00509             if (receivedString[NbBytes]==finalChar)
00510             {
00511                 // This is the final char, add zero (end of string)
00512                 receivedString  [++NbBytes]=0;
00513                 // Return the number of bytes read
00514                 return NbBytes;
00515             }
00516
00517             // The character is not the final char, increase the number of bytes read
00518             NbBytes++;
00519         }
00520
00521         // An error occured while reading, return the error number
00522         if (charRead<0) return charRead;
00523     }
00524     // Buffer is full : return -3
00525     return -3;
00526 }
00527
00528
00541 int serialib::readString(char *receivedString,char finalChar,unsigned int maxNbBytes,unsigned int
       timeOut_ms)
00542 {
```

```
00543     // Check if timeout is requested
00544     if (timeOut_ms==0) return readStringNoTimeOut(receivedString,finalChar,maxNbBytes);
00545
00546     // Number of bytes read
00547     unsigned int    nbBytes=0;
00548     // Character read on serial device
00549     char            charRead;
00550     // Timer used for timeout
00551     timeOut         timer;
00552     long int        timeOutParam;
00553
00554     // Initialize the timer (for timeout)
00555     timer.initTimer();
00556
00557     // While the buffer is not full
00558     while (nbBytes<maxNbBytes)
00559     {
00560         // Compute the TimeOut for the next call of ReadChar
00561         timeOutParam = timeOut_ms-timer.elapsedTime_ms();
00562
00563         // If there is time remaining
00564         if (timeOutParam>0)
00565         {
00566             // Wait for a byte on the serial link with the remaining time as timeout
00567             charRead=readChar(&receivedString[nbBytes],timeOutParam);
00568
00569             // If a byte has been received
00570             if (charRead==1)
00571             {
00572                 // Check if the character received is the final one
00573                 if (receivedString[nbBytes]==finalChar)
00574                 {
00575                     // Final character: add the end character 0
00576                     receivedString  [++nbBytes]=0;
00577                     // Return the number of bytes read
00578                     return nbBytes;
00579                 }
00580                 // This is not the final character, just increase the number of bytes read
00581                 nbBytes++;
00582             }
00583             // Check if an error occured during reading char
00584             // If an error occurend, return the error number
00585             if (charRead<0) return charRead;
00586         }
00587         // Check if timeout is reached
00588         if (timer.elapsedTime_ms()>timeOut_ms)
00589         {
00590             // Add the end caracter
00591             receivedString[nbBytes]=0;
00592             // Return 0 (timeout reached)
00593             return 0;
00594         }
00595     }
00596
00597     // Buffer is full : return -3
00598     return -3;
00599 }
00600
00601
00615 int serialib::readBytes (void *buffer,unsigned int maxNbBytes,unsigned int timeOut_ms, unsigned int
       sleepDuration_us)
00616 {
00617 #if defined (_WIN32) || defined(_WIN64)
00618     // Avoid warning while compiling
00619     UNUSED(sleepDuration_us);
00620
00621     // Number of bytes read
00622     DWORD dwBytesRead = 0;
00623
00624     // Set the TimeOut
00625     timeouts.ReadTotalTimeoutConstant=(DWORD)timeOut_ms;
00626
00627     // Write the parameters and return -1 if an error occrured
00628     if(!SetCommTimeouts(hSerial, &timeouts)) return -1;
00629
00630
00631     // Read the bytes from the serial device, return -2 if an error occured
00632     if(!ReadFile(hSerial,buffer,(DWORD)maxNbBytes,&dwBytesRead, NULL))  return -2;
00633
00634     // Return the byte read
00635     return dwBytesRead;
00636 #endif
00637 #if defined (__linux__) || defined(__APPLE__)
00638     // Timer used for timeout
00639     timeOut          timer;
00640     // Initialise the timer
00641     timer.initTimer();
```

```
00642     unsigned int     NbByteRead=0;
00643     // While Timeout is not reached
00644     while (timer.elapsedTime_ms()<timeOut_ms || timeOut_ms==0)
00645     {
00646         // Compute the position of the current byte
00647         unsigned char* Ptr=(unsigned char*)buffer+NbByteRead;
00648         // Try to read a byte on the device
00649         int Ret=read(fd,(void*)Ptr,maxNbBytes-NbByteRead);
00650         // Error while reading
00651         if (Ret==-1) return -2;
00652
00653         // One or several byte(s) has been read on the device
00654         if (Ret>0)
00655         {
00656             // Increase the number of read bytes
00657             NbByteRead+=Ret;
00658             // Success : bytes has been read
00659             if (NbByteRead>=maxNbBytes)
00660                 return NbByteRead;
00661         }
00662         // Suspend the loop to avoid charging the CPU
00663         usleep (sleepDuration_us);
00664     }
00665     // Timeout reached, return the number of bytes read
00666     return NbByteRead;
00667 #endif
00668 }
00669
00670
00671
00672
00673 // ____________________________
00674 // ::: Special operation :::
00675
00676
00677
00683 char serialib::flushReceiver()
00684 {
00685 #if defined (_WIN32) || defined(_WIN64)
00686     // Purge receiver
00687     return PurgeComm (hSerial, PURGE_RXCLEAR);
00688 #endif
00689 #if defined (__linux__) || defined(__APPLE__)
00690     // Purge receiver
00691     tcflush(fd,TCIFLUSH);
00692     return true;
00693 #endif
00694 }
00695
00696
00697
00702 int serialib::available()
00703 {
00704 #if defined (_WIN32) || defined(_WIN64)
00705     // Device errors
00706     DWORD commErrors;
00707     // Device status
00708     COMSTAT commStatus;
00709     // Read status
00710     ClearCommError(hSerial, &commErrors, &commStatus);
00711     // Return the number of pending bytes
00712     return commStatus.cbInQue;
00713 #endif
00714 #if defined (__linux__) || defined(__APPLE__)
00715     int nBytes=0;
00716     // Return number of pending bytes in the receiver
00717     ioctl(fd, FIONREAD, &nBytes);
00718     return nBytes;
00719 #endif
00720
00721 }
00722
00723
00724
00725 // ____________________
00726 // ::: I/O Access :::
00727
00737 bool serialib::DTR(bool status)
00738 {
00739     if (status)
00740         // Set DTR
00741         return this->setDTR();
00742     else
00743         // Unset DTR
00744         return this->clearDTR();
00745 }
00746
```

```
00747
00754 bool serialib::setDTR()
00755 {
00756 #if defined (_WIN32) || defined(_WIN64)
00757     // Set DTR
00758     currentStateDTR=true;
00759     return EscapeCommFunction(hSerial,SETDTR);
00760 #endif
00761 #if defined (__linux__) || defined(__APPLE__)
00762     // Set DTR
00763     int status_DTR=0;
00764     ioctl(fd, TIOCMGET, &status_DTR);
00765     status_DTR |= TIOCM_DTR;
00766     ioctl(fd, TIOCMSET, &status_DTR);
00767     return true;
00768 #endif
00769 }
00770
00777 bool serialib::clearDTR()
00778 {
00779 #if defined (_WIN32) || defined(_WIN64)
00780     // Clear DTR
00781     currentStateDTR=true;
00782     return EscapeCommFunction(hSerial,CLRDTR);
00783 #endif
00784 #if defined (__linux__) || defined(__APPLE__)
00785     // Clear DTR
00786     int status_DTR=0;
00787     ioctl(fd, TIOCMGET, &status_DTR);
00788     status_DTR &= ~TIOCM_DTR;
00789     ioctl(fd, TIOCMSET, &status_DTR);
00790     return true;
00791 #endif
00792 }
00793
00794
00795
00805 bool serialib::RTS(bool status)
00806 {
00807     if (status)
00808         // Set RTS
00809         return this->setRTS();
00810     else
00811         // Unset RTS
00812         return this->clearRTS();
00813 }
00814
00815
00822 bool serialib::setRTS()
00823 {
00824 #if defined (_WIN32) || defined(_WIN64)
00825     // Set RTS
00826     currentStateRTS=false;
00827     return EscapeCommFunction(hSerial,SETRTS);
00828 #endif
00829 #if defined (__linux__) || defined(__APPLE__)
00830     // Set RTS
00831     int status_RTS=0;
00832     ioctl(fd, TIOCMGET, &status_RTS);
00833     status_RTS |= TIOCM_RTS;
00834     ioctl(fd, TIOCMSET, &status_RTS);
00835     return true;
00836 #endif
00837 }
00838
00839
00840
00847 bool serialib::clearRTS()
00848 {
00849 #if defined (_WIN32) || defined(_WIN64)
00850     // Clear RTS
00851     currentStateRTS=false;
00852     return EscapeCommFunction(hSerial,CLRRTS);
00853 #endif
00854 #if defined (__linux__) || defined(__APPLE__)
00855     // Clear RTS
00856     int status_RTS=0;
00857     ioctl(fd, TIOCMGET, &status_RTS);
00858     status_RTS &= ~TIOCM_RTS;
00859     ioctl(fd, TIOCMSET, &status_RTS);
00860     return true;
00861 #endif
00862 }
00863
00864
00865
00866
```

```
00872 bool serialib::isCTS()
00873 {
00874 #if defined (_WIN32) || defined(_WIN64)
00875     DWORD modemStat;
00876     GetCommModemStatus(hSerial, &modemStat);
00877     return modemStat & MS_CTS_ON;
00878 #endif
00879 #if defined (__linux__) || defined(__APPLE__)
00880     int status=0;
00881     //Get the current status of the CTS bit
00882     ioctl(fd, TIOCMGET, &status);
00883     return status & TIOCM_CTS;
00884 #endif
00885 }
00886
00887
00888
00894 bool serialib::isDSR()
00895 {
00896 #if defined (_WIN32) || defined(_WIN64)
00897     DWORD modemStat;
00898     GetCommModemStatus(hSerial, &modemStat);
00899     return modemStat & MS_DSR_ON;
00900 #endif
00901 #if defined (__linux__) || defined(__APPLE__)
00902     int status=0;
00903     //Get the current status of the DSR bit
00904     ioctl(fd, TIOCMGET, &status);
00905     return status & TIOCM_DSR;
00906 #endif
00907 }
00908
00909
00910
00911
00912
00913
00920 bool serialib::isDCD()
00921 {
00922 #if defined (_WIN32) || defined(_WIN64)
00923     DWORD modemStat;
00924     GetCommModemStatus(hSerial, &modemStat);
00925     return modemStat & MS_RLSD_ON;
00926 #endif
00927 #if defined (__linux__) || defined(__APPLE__)
00928     int status=0;
00929     //Get the current status of the DCD bit
00930     ioctl(fd, TIOCMGET, &status);
00931     return status & TIOCM_CAR;
00932 #endif
00933 }
00934
00935
00941 bool serialib::isRI()
00942 {
00943 #if defined (_WIN32) || defined(_WIN64)
00944     DWORD modemStat;
00945     GetCommModemStatus(hSerial, &modemStat);
00946     return modemStat & MS_RING_ON;
00947 #endif
00948 #if defined (__linux__) || defined(__APPLE__)
00949     int status=0;
00950     //Get the current status of the RING bit
00951     ioctl(fd, TIOCMGET, &status);
00952     return status & TIOCM_RNG;
00953 #endif
00954 }
00955
00956
00963 bool serialib::isDTR()
00964 {
00965 #if defined (_WIN32) || defined( _WIN64)
00966     return currentStateDTR;
00967 #endif
00968 #if defined (__linux__) || defined(__APPLE__)
00969     int status=0;
00970     //Get the current status of the DTR bit
00971     ioctl(fd, TIOCMGET, &status);
00972     return status & TIOCM_DTR  ;
00973 #endif
00974 }
00975
00976
00977
00984 bool serialib::isRTS()
00985 {
00986 #if defined (_WIN32) || defined(_WIN64)
```

```
00987     return currentStateRTS;
00988 #endif
00989 #if defined (__linux__) || defined(__APPLE__)
00990     int status=0;
00991     //Get the current status of the CTS bit
00992     ioctl(fd, TIOCMGET, &status);
00993     return status & TIOCM_RTS;
00994 #endif
00995 }
00996
00997
00998
00999
01000
01001
01002 // ****************************************
01003 //  Class timeOut
01004 // ****************************************
01005
01006
01010 // Constructor
01011 timeOut::timeOut()
01012 {}
01013
01014
01018 //Initialize the timer
01019 void timeOut::initTimer()
01020 {
01021 #if defined (NO_POSIX_TIME)
01022     LARGE_INTEGER tmp;
01023     QueryPerformanceFrequency(&tmp);
01024     counterFrequency = tmp.QuadPart;
01025     // Used to store the previous time (for computing timeout)
01026     QueryPerformanceCounter(&tmp);
01027     previousTime = tmp.QuadPart;
01028 #else
01029     gettimeofday(&previousTime, NULL);
01030 #endif
01031 }
01032
01038 //Return the elapsed time since initialization
01039 unsigned long int timeOut::elapsedTime_ms()
01040 {
01041 #if defined (NO_POSIX_TIME)
01042     // Current time
01043     LARGE_INTEGER CurrentTime;
01044     // Number of ticks since last call
01045     int sec;
01046
01047     // Get current time
01048     QueryPerformanceCounter(&CurrentTime);
01049
01050     // Compute the number of ticks elapsed since last call
01051     sec=CurrentTime.QuadPart-previousTime;
01052
01053     // Return the elapsed time in milliseconds
01054     return sec/(counterFrequency/1000);
01055 #else
01056     // Current time
01057     struct timeval CurrentTime;
01058     // Number of seconds and microseconds since last call
01059     int sec,usec;
01060
01061     // Get current time
01062     gettimeofday(&CurrentTime, NULL);
01063
01064     // Compute the number of seconds and microseconds elapsed since last call
01065     sec=CurrentTime.tv_sec-previousTime.tv_sec;
01066     usec=CurrentTime.tv_usec-previousTime.tv_usec;
01067
01068     // If the previous usec is higher than the current one
01069     if (usec<0)
01070     {
01071         // Recompute the microseonds and substract one second
01072         usec=1000000-previousTime.tv_usec+CurrentTime.tv_usec;
01073         sec--;
01074     }
01075
01076     // Return the elapsed time in milliseconds
01077     return sec*1000+usec/1000;
01078 #endif
01079 }
```

5.11 Référence du fichier serialib.h

Header file of the class serialib. This class is used for communication over a serial device.

Classes

— class serialib
This class is used for communication over a serial device.
— class timeOut
This class can manage a timer which is used as a timeout.

Macros

— #define UNUSED(x) (void)(x)

Énumérations

— enum SerialDataBits {
SERIAL_DATABITS_5 , SERIAL_DATABITS_6 , SERIAL_DATABITS_7 , SERIAL_DATABITS_8 ,
SERIAL_DATABITS_16 }
— enum SerialStopBits { SERIAL_STOPBITS_1 , SERIAL_STOPBITS_1_5 , SERIAL_STOPBITS_2 }
— enum SerialParity {
SERIAL_PARITY_NONE , SERIAL_PARITY_EVEN , SERIAL_PARITY_ODD , SERIAL_PARITY_MARK ,
SERIAL_PARITY_SPACE }

5.11.1 Description détaillée

Header file of the class serialib. This class is used for communication over a serial device.

Auteur

Philippe Lucidarme (University of Angers)

Version

2.0

Date

december the 27th of 2019 This Serial library is used to communicate through serial port.

Définition dans le fichier serialib.h.

5.11.2 Documentation des macros

5.11.2.1 UNUSED

```
#define UNUSED(
              x ) (void)(x)
```

To avoid unused parameters

Définition à la ligne 56 du fichier serialib.h.

5.11.3 Documentation du type de l'énumération

5.11.3.1 SerialDataBits

```
enum SerialDataBits
```

number of serial data bits

Valeurs énumérées

SERIAL_DATABITS_5	5 databits
SERIAL_DATABITS_6	6 databits
SERIAL_DATABITS_7	7 databits
SERIAL_DATABITS_8	8 databits
SERIAL_DATABITS_16	16 databits

Définition à la ligne 61 du fichier serialib.h.

```
00061                     {
00062     SERIAL_DATABITS_5,
00063     SERIAL_DATABITS_6,
00064     SERIAL_DATABITS_7,
00065     SERIAL_DATABITS_8,
00066     SERIAL_DATABITS_16,
00067 };
```

5.11.3.2 SerialParity

```
enum SerialParity
```

type of serial parity bits

Valeurs énumérées

SERIAL_PARITY_NONE	no parity bit
SERIAL_PARITY_EVEN	even parity bit
SERIAL_PARITY_ODD	odd parity bit
SERIAL_PARITY_MARK	mark parity
SERIAL_PARITY_SPACE	space bit

Définition à la ligne 81 du fichier serialib.h.

```
00081                   {
00082     SERIAL_PARITY_NONE,
00083     SERIAL_PARITY_EVEN,
00084     SERIAL_PARITY_ODD,
00085     SERIAL_PARITY_MARK,
00086     SERIAL_PARITY_SPACE
00087 };
```

5.11.3.3 SerialStopBits

```
enum SerialStopBits
```

number of serial stop bits

Valeurs énumérées

SERIAL_STOPBITS_1	1 stop bit
SERIAL_STOPBITS_1↩ _5	1.5 stop bits
SERIAL_STOPBITS_2	2 stop bits

Définition à la ligne 72 du fichier serialib.h.

```
00072                     {
00073     SERIAL_STOPBITS_1,
00074     SERIAL_STOPBITS_1_5,
00075     SERIAL_STOPBITS_2,
00076 };
```

5.12 serialib.h

Aller à la documentation de ce fichier.

```
00001
00019 #ifndef SERIALIB_H
00020 #define SERIALIB_H
00021
00022 #if defined(__CYGWIN__)
00023     // This is Cygwin special case
00024     #include <sys/time.h>
00025 #endif
00026
00027 // Include for windows
00028 #if defined (_WIN32) || defined (_WIN64)
00029 #if defined(__GNUC__)
00030     // This is MinGW special case
00031     #include <sys/time.h>
00032 #else
00033     // sys/time.h does not exist on "actual" Windows
00034     #define NO_POSIX_TIME
```

```
00035 #endif
00036     // Accessing to the serial port under Windows
00037     #include <windows.h>
00038 #endif
00039
00040 // Include for Linux
00041 #if defined (__linux__) || defined(__APPLE__)
00042     #include <stdlib.h>
00043     #include <sys/types.h>
00044     #include <sys/shm.h>
00045     #include <termios.h>
00046     #include <string.h>
00047     #include <iostream>
00048     #include <sys/time.h>
00049     // File control definitions
00050     #include <fcntl.h>
00051     #include <unistd.h>
00052     #include <sys/ioctl.h>
00053 #endif
00054
00056 #define UNUSED(x) (void)(x)
00057
00061 enum SerialDataBits {
00062     SERIAL_DATABITS_5,
00063     SERIAL_DATABITS_6,
00064     SERIAL_DATABITS_7,
00065     SERIAL_DATABITS_8,
00066     SERIAL_DATABITS_16,
00067 };
00068
00072 enum SerialStopBits {
00073     SERIAL_STOPBITS_1,
00074     SERIAL_STOPBITS_1_5,
00075     SERIAL_STOPBITS_2,
00076 };
00077
00081 enum SerialParity {
00082     SERIAL_PARITY_NONE,
00083     SERIAL_PARITY_EVEN,
00084     SERIAL_PARITY_ODD,
00085     SERIAL_PARITY_MARK,
00086     SERIAL_PARITY_SPACE
00087 };
00088
00092 class serialib
00093 {
00094 public:
00095
00096     //_____________________________________
00097     // ::: Constructors and destructors :::
00098
00099
00100
00101     // Constructor of the class
00102     serialib    ();
00103
00104     // Destructor
00105     ~serialib   ();
00106
00107
00108
00109     //_____________________________________
00110     // ::: Configuration and initialization :::
00111
00112
00113     // Open a device
00114     int openDevice(const char *Device, const unsigned int Bauds,
00115                    SerialDataBits Databits = SERIAL_DATABITS_8,
00116                    SerialParity Parity = SERIAL_PARITY_NONE,
00117                    SerialStopBits Stopbits = SERIAL_STOPBITS_1);
00118
00119     // Check device opening state
00120     bool isDeviceOpen();
00121
00122     // Close the current device
00123     void    closeDevice();
00124
00125
00126
00127
00128     //_____________________________________
00129     // ::: Read/Write operation on characters :::
00130
00131
00132     // Write a char
00133     char    writeChar   (char);
00134
```

```
00135     // Read a char (with timeout)
00136     char    readChar    (char *pByte,const unsigned int timeOut_ms=0);
00137
00138
00139
00140
00141     //_____________________________________
00142     // ::: Read/Write operation on strings :::
00143
00144
00145     // Write a string
00146     char    writeString (const char *String);
00147
00148     // Read a string (with timeout)
00149     int     readString  (   char *receivedString,
00150                             char finalChar,
00151                             unsigned int maxNbBytes,
00152                             const unsigned int timeOut_ms=0);
00153
00154
00155
00156     // ____________________________________
00157     // ::: Read/Write operation on bytes :::
00158
00159
00160     // Write an array of bytes
00161     char    writeBytes  (const void *Buffer, const unsigned int NbBytes);
00162
00163     // Read an array of byte (with timeout)
00164     int     readBytes   (void *buffer,unsigned int maxNbBytes,const unsigned int timeOut_ms=0,
      unsigned int sleepDuration_us=100);
00165
00166
00167
00168
00169     // ____________________________
00170     // ::: Special operation :::
00171
00172
00173     // Empty the received buffer
00174     char    flushReceiver();
00175
00176     // Return the number of bytes in the received buffer
00177     int     available();
00178
00179
00180
00181
00182     // ____________________________
00183     // ::: Access to IO bits :::
00184
00185
00186     // Set CTR status (Data Terminal Ready, pin 4)
00187     bool    DTR(bool status);
00188     bool    setDTR();
00189     bool    clearDTR();
00190
00191     // Set RTS status (Request To Send, pin 7)
00192     bool    RTS(bool status);
00193     bool    setRTS();
00194     bool    clearRTS();
00195
00196     // Get RI status (Ring Indicator, pin 9)
00197     bool    isRI();
00198
00199     // Get DCD status (Data Carrier Detect, pin 1)
00200     bool    isDCD();
00201
00202     // Get CTS status (Clear To Send, pin 8)
00203     bool    isCTS();
00204
00205     // Get DSR status (Data Set Ready, pin 9)
00206     bool    isDSR();
00207
00208     // Get RTS status (Request To Send, pin 7)
00209     bool    isRTS();
00210
00211     // Get CTR status (Data Terminal Ready, pin 4)
00212     bool    isDTR();
00213
00214
00215 private:
00216     // Read a string (no timeout)
00217     int             readStringNoTimeOut  (char *String,char FinalChar,unsigned int MaxNbBytes);
00218
00219     // Current DTR and RTS state (can't be read on WIndows)
00220     bool            currentStateRTS;
```

```
00221     bool            currentStateDTR;
00222
00223
00224
00225
00226
00227 #if defined (_WIN32) || defined( _WIN64)
00228     // Handle on serial device
00229     HANDLE          hSerial;
00230     // For setting serial port timeouts
00231     COMMTIMEOUTS    timeouts;
00232 #endif
00233 #if defined (__linux__) || defined(__APPLE__)
00234     int             fd;
00235 #endif
00236
00237 };
00238
00239
00240
00244 // Class timeOut
00245 class timeOut
00246 {
00247 public:
00248
00249     // Constructor
00250     timeOut();
00251
00252     // Init the timer
00253     void                initTimer();
00254
00255     // Return the elapsed time since initialization
00256     unsigned long int   elapsedTime_ms();
00257
00258 private:
00259 #if defined (NO_POSIX_TIME)
00260     // Used to store the previous time (for computing timeout)
00261     LONGLONG       counterFrequency;
00262     LONGLONG       previousTime;
00263 #else
00264     // Used to store the previous time (for computing timeout)
00265     struct timeval      previousTime;
00266 #endif
00267 };
00268
00269 #endif // serialib_H
```

5.13 Référence du fichier xbeelib.cpp

Fichier source de la classe XBee. Cette classe est utilisée afin de programmer les modules XBee en UART et de mettre en place des communications entre différents modules XBee.

```
#include "xbeelib.h"
```

Variables

— serialib serial
— Log logXbee ("xbee")

5.13.1 Description détaillée

Fichier source de la classe XBee. Cette classe est utilisée afin de programmer les modules XBee en UART et de mettre en place des communications entre différents modules XBee.

Auteur

Samuel-Charles DITTE-DESTREE (samuelditteDestree@protonmail.com)

Version

3.0

Date

10/03/2022

Définition dans le fichier xbeelib.cpp.

5.13.2 Documentation des variables

5.13.2.1 logXbee

```
Log logXbee("xbee") (
             "xbee"  )
```

5.13.2.2 serial

```
serialib serial
```

Définition à la ligne 12 du fichier xbeelib.cpp.

5.14 xbeelib.cpp

Aller à la documentation de ce fichier.

```
00001
00008 #include "xbeelib.h"
00009
00010 using namespace std;
00011
00012 serialib serial;
00013 Log logXbee("xbee");
00014
00015 //_____________________________________
00016 // ::: Constructeurs et destructeurs :::
00017
00021 XBee::XBee(){ }
00022
00026 XBee::~XBee(){ }
00027
00028
00029 //_____________________________________________
00030 // ::: Configuration and initialisation :::
00031
00046 int XBee::openSerialConnection(int mode){
00047     int errorOpening;
00048     if(mode == 1){
00049         errorOpening = serial.openDevice(XB_SERIAL_PORT_DEFAULT, XB_BAUDRATE_DEFAULT,
      XB_DATABITS_DEFAULT, XB_PARITY_DEFAULT, XB_STOPBITS_DEFAULT);
00050
00051         if (errorOpening != 1)
00052             logXbee « "(serial) /!\\ erreur " « errorOpening « " : impossible d'ouvrir le port " «
      XB_SERIAL_PORT_DEFAULT  « " - baudrate : " « XB_BAUDRATE_DEFAULT « " - parité : " « XB_PARITY_DEFAULT
      « mendl;
00053         else{
```

```
00054             logXbee « "(serial) connexion ouverte avec succès sur le port " « XB_SERIAL_PORT_DEFAULT « " - baudrate : " « XB_BAUDRATE_DEFAULT « " - parité : " « XB_PARITY_DEFAULT « mendl;
00055         checkATConfig();
00056     }
00057     } else if(mode == 0) {
00058         errorOpening = serial.openDevice(XB_SERIAL_PORT_PRIMARY, XB_BAUDRATE_PRIMARY, XB_DATABITS_PRIMARY, XB_PARITY_PRIMARY, XB_STOPBITS_PRIMARY);
00059
00060         if (errorOpening != 1)
00061             logXbee « "(serial) /!\\ erreur " « errorOpening « " : impossible d'ouvrir le port " « XB_SERIAL_PORT_PRIMARY  « " - baudrate : " « XB_BAUDRATE_PRIMARY « " - parités : " « XB_PARITY_PRIMARY « mendl;
00062
00063     else{
00064             logXbee « "(serial) connexion ouverte avec succès sur le port " « XB_SERIAL_PORT_PRIMARY « " - baudrate : " « XB_BAUDRATE_PRIMARY « " - parité : " « XB_PARITY_PRIMARY « mendl;
00065             checkATConfig();
00066         }
00067     }
00068
00069     return errorOpening;
00070 }
00071
00075 void XBee::closeSerialConnection(){
00076     serial.flushReceiver();
00077     logXbee « "(serial) buffer Rx nettoyé avec succès" « mendl;
00078
00079     serial.closeDevice();
00080     logXbee « "(serial) connexion série fermée avec succès" « mendl;
00081 }
00082
00083 //_____________________________________
00084 // ::: Configuration en mode AT :::
00085
00104 int XBee::checkATConfig(){
00105     if(!enterATMode()){
00106         logXbee « "/!\\ (config AT) erreur " « XB_AT_E_ENTER « " : impossible d'entrer dans le mode AT" « mendl;
00107         closeSerialConnection();
00108         if(MODE == 0){
00109             MODE = 1;
00110             openSerialConnection(1);
00111         }else{
00112             MODE = 0;
00113             openSerialConnection();
00114         }
00115         return XB_AT_E_ENTER;
00116     }
00117     else logXbee « "(config AT) entrée dans le mode AT" « mendl;
00118
00119     if(!sendATCommand(XB_AT_CMD_BAUDRATE, XB_AT_V_BAUDRATE, XB_AT_M_GET)){
00120         if(!sendATCommand(XB_AT_CMD_BAUDRATE, XB_AT_V_BAUDRATE)){
00121             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_BAUDRATE « " : impossible de configurer le baudrate" « mendl;
00122             return XB_AT_E_BAUDRATE;
00123         }
00124         logXbee « "(config AT) baudrate configuré avec succès" « mendl;
00125     }
00126     else logXbee « "(config AT) baudrate vérifié avec succès" « mendl;
00127
00128     if(!sendATCommand(XB_AT_CMD_PARITY, XB_AT_V_PARITY, XB_AT_M_GET)){
00129         if(!sendATCommand(XB_AT_CMD_PARITY, XB_AT_V_PARITY)){
00130             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_PARITY « " : impossible de configurer le nombre de bits de parité" « mendl;
00131             return XB_AT_E_PARITY;
00132         }
00133         logXbee « "(config AT) nombre de bits de parité configuré avec succès" « mendl;
00134
00135         if(!writeATConfig()){
00136             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_WRITE_CONFIG « " : impossible d'écrire les paramètres dans la mémoire flash" « mendl;
00137             return XB_AT_E_WRITE_CONFIG;
00138         }
00139
00140         closeSerialConnection();
00141         if(MODE == 0){
00142             MODE = 1;
00143             openSerialConnection(1);
00144         }else{
00145             MODE = 0;
00146             openSerialConnection();
00147         }
00148     }
00149     else logXbee « "(config AT) nombre de bits de parité vérifié avec succès" « mendl;
00150
00151     if(!sendATCommand(XB_AT_CMD_API, XB_AT_V_API, XB_AT_M_GET)){
00152         if(!sendATCommand(XB_AT_CMD_API, XB_AT_V_API)){
```

```
00153             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_API « " : impossible de configurer le mode
      API" « mendl;
00154             return XB_AT_E_API;
00155         }
00156         logXbee « "(config AT) mode API configuré avec succès" « mendl;
00157     }
00158     else logXbee « "(config AT) mode API vérifié avec succès" « mendl;
00159 
00160     if(!sendATCommand(XB_AT_CMD_AES, XB_AT_V_AES, XB_AT_M_GET)){
00161         if(!sendATCommand(XB_AT_CMD_AES, XB_AT_V_AES)){
00162             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_AES « " : impossible de configurer le
      paramètre de chiffrement AES" « mendl;
00163             return XB_AT_E_AES;
00164         }
00165         logXbee « "(config AT) chiffrement AES configuré avec succès" « mendl;
00166     }
00167     else logXbee « "(config AT) chiffrement AES vérifié avec succès" « mendl;
00168 
00169     if(!sendATCommand(XB_AT_CMD_AES_KEY, XB_AT_V_AES_KEY)){
00170         logXbee « "/!\\ (config AT) erreur " « XB_AT_E_AES_KEY « " : impossible de configurer la clé
      de chiffrement AES" « mendl;
00171         return XB_AT_E_AES_KEY;
00172     }
00173 
00174     if(!sendATCommand(XB_AT_CMD_CHANEL, XB_AT_V_CHANEL, XB_AT_M_GET)){
00175         if(!sendATCommand(XB_AT_CMD_CHANEL, XB_AT_V_CHANEL)){
00176             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_CHANEL « " : impossible de configurer le
      canal de découverte réseau" « mendl;
00177             return XB_AT_E_CHANEL;
00178         }
00179         logXbee « "(config AT) canal de découverte réseau configuré avec succès" « mendl;
00180     }
00181     else logXbee « "(config AT) canal de découverte réseau vérifié avec succès" « mendl;
00182 
00183     if(!sendATCommand(XB_AT_CMD_PAN_ID, XB_AT_V_PAN_ID, XB_AT_M_GET)){
00184         if(!sendATCommand(XB_AT_CMD_PAN_ID, XB_AT_V_PAN_ID)){
00185             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_PAN_ID « " : impossible de configurer l'ID
      du réseau" « mendl;
00186             return XB_AT_E_PAN_ID;
00187         }
00188         logXbee « "(config AT) ID du réseau configuré avec succès" « mendl;
00189     }
00190     else logXbee « "(config AT) ID du réseau vérifié avec succès" « mendl;
00191 
00192     if(!sendATCommand(XB_AT_CMD_COORDINATOR, XB_AT_V_COORDINATOR, XB_AT_M_GET)){
00193         if(!sendATCommand(XB_AT_CMD_COORDINATOR, XB_AT_V_COORDINATOR)){
00194             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_COORDINATOR « " : impossible de configurer
      le mode coordinateur" « mendl;
00195             return XB_AT_E_COORDINATOR;
00196         }
00197         logXbee « "(config AT) mode coordinateur configuré avec succès" « mendl;
00198     }
00199     else logXbee « "(config AT) mode coordinateur vérifié avec succès" « mendl;
00200 
00201     if(!sendATCommand(XB_AT_CMD_16BIT_SOURCE_ADDR, XB_AT_V_16BIT_SOURCE_ADDR, XB_AT_M_GET)){
00202         if(!sendATCommand(XB_AT_CMD_16BIT_SOURCE_ADDR, XB_AT_V_16BIT_SOURCE_ADDR)){
00203             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_16BIT_SOURCE_ADDR « " : impossible de
      configurer l'adresse source 16bits" « mendl;
00204             return XB_AT_E_16BIT_SOURCE_ADDR;
00205         }
00206         logXbee « "(config AT) adresse source 16bits configurée avec succès" « mendl;
00207     }
00208     else logXbee « "(config AT) adresse source 16bits vérifiée avec succès" « mendl;
00209 
00210     if(!sendATCommand(XB_AT_CMD_LOW_DEST_ADDR, XB_AT_V_LOW_DEST_ADDR, XB_AT_M_GET)){
00211         if(!sendATCommand(XB_AT_CMD_LOW_DEST_ADDR, XB_AT_V_LOW_DEST_ADDR)){
00212             logXbee « "/!\\ (config AT) erreur " « XB_AT_E_LOW_DEST_ADDR « " : impossible de
      configurer l'adresse de destination" « mendl;
00213             return XB_AT_E_LOW_DEST_ADDR;
00214         }
00215         logXbee « "(config AT) adresse de destination configurée avec succès" « mendl;
00216     }
00217     else logXbee « "(config AT) adresse de destination vérifiée avec succès" « mendl;
00218 
00219     if(!writeATConfig()){
00220         logXbee « "/!\\ (config AT) erreur " « XB_AT_E_WRITE_CONFIG « " : impossible d'écrire les
      paramètres dans la mémoire flash" « mendl;
00221         return XB_AT_E_WRITE_CONFIG;
00222     }
00223     else logXbee « "(config AT) configuration AT enregistrée dans la mémoire du module" « mendl;
00224 
00225     if(!discoverXbeeNetwork()){
00226         logXbee « "/!\\ (config AT) erreur " « XB_AT_E_DISCOVER_NETWORK « " : impossible d'établir une
      connexion avec le module XBee distant" « mendl;
00227         return XB_AT_E_DISCOVER_NETWORK;
00228     }
00229     else logXbee « "(config AT) connexion XBee établie avec succès avec le module distant" « mendl;
```

```
00230
00231     if(!exitATMode()){
00232         logXbee « "/!\\ (config AT) erreur " « XB_AT_E_EXIT « " : impossible de sortir du mode AT" «
      mendl;
00233         return XB_AT_E_EXIT;
00234     }
00235
00236     logXbee « "(config AT) configuration AT réalisée avec succès" « mendl;
00237     return XB_AT_E_SUCCESS;
00238 }
00239
00244 void XBee::delay(unsigned int time){
       std::this_thread::sleep_for(std::chrono::milliseconds(time*1000)); }
00245
00246
00254 bool XBee::readATResponse(const char *value, int mode){
00255     string reponse = readString();
00256     logXbee « "(config AT) réponse du Xbee : " « reponse « mendl;
00257
00258     if(mode == 0)
00259         if(reponse == value) return true;
00260
00261     else if(mode == 1)
00262         if(reponse != value) return true;
00263
00264     return false;
00265 }
00266
00272 bool XBee::enterATMode(){
00273     serial.writeString(XB_AT_CMD_ENTER);
00274     delay(3);
00275     serial.writeString(XB_AT_V_END_LINE);
00276     logXbee « "entrée en mode AT" « mendl;
00277     return readATResponse(XB_AT_R_SUCCESS);
00278 }
00279
00285 bool XBee::exitATMode(){
00286     serial.writeString(XB_AT_CMD_EXIT);
00287     serial.writeString(XB_AT_V_END_LINE);
00288     logXbee « "sortie du mode AT" « mendl;
00289     return readATResponse(XB_AT_R_SUCCESS);
00290 }
00291
00297 bool XBee::discoverXbeeNetwork(){
00298     serial.writeString(XB_AT_CMD_DISCOVER_NETWORK);
00299     serial.writeString(XB_AT_V_END_LINE);
00300     logXbee « "lancement de la découverte réseau XBee" « mendl;
00301     return readATResponse(XB_AT_V_DISCOVER_NETWORK);
00302 }
00303
00309 bool XBee::writeATConfig(){
00310     serial.writeString(XB_AT_CMD_WRITE_CONFIG);
00311     serial.writeString(XB_AT_V_END_LINE);
00312     //cout « "* Ecriture de la configuration AT..." « endl;
00313     logXbee « "écriture des paramètres AT dans la mémoire" « mendl;
00314     return readATResponse(XB_AT_R_SUCCESS);
00315 }
00316
00325 bool XBee::sendATCommand(const char *command, const char *value, unsigned int mode){
00326     if(mode == XB_AT_M_GET){
00327         serial.writeString(command);
00328         serial.writeString(XB_AT_V_END_LINE);
00329         logXbee « "(config AT) envoi de la commande AT : " « command « mendl;
00330         return readATResponse(value);
00331     }else{
00332         serial.writeString(command);
00333         serial.writeString(value);
00334         logXbee « "(config AT) envoi de la commande AT : " « command « "=" « value « mendl;
00335         if(command == XB_AT_CMD_DISCOVER_NETWORK)
00336             return readATResponse(XB_AT_R_SUCCESS);
00337         else
00338             return readATResponse(XB_AT_V_DISCOVER_NETWORK, 1);
00339     }
00340 }
00341
00342 //_____________________________________________________
00343 // ::: Envoi/Réception/Traitement des trames de messages :::
00344
00351 int XBee::crc16(int trame[], uint8_t taille){
00352     int crc = 0xFFFF, count = 0;
00353     int octet_a_traiter;
00354     const int POLYNOME = 0xA001;
00355
00356     octet_a_traiter = trame[0];
00357
00358     do{
00359         crc ^= octet_a_traiter;
```

```
00360         for(uint8_t i = 0; i < 8; i++){
00361
00362             if((crc%2)!=0)
00363             crc = (crc » 1) ^ POLYNOME;
00364
00365             else
00366                 crc = (crc » 1);
00367
00368         }
00369         count++;
00370         octet_a_traiter = trame[count];
00371
00372     }while(count < taille);
00373
00374     return crc;
00375 }
00376
00384 int XBee::sendTrame(uint8_t ad_dest, uint8_t code_fct, char* data){
00385
00386     cout « hex « showbase;
00387
00388     uint8_t length_trame = strlen(data)+10;
00389     uint8_t trame[length_trame];
00390     int trame_int[length_trame];
00391     int id_trame = ++ID_TRAME;
00392     uint8_t id_trame_low = id_trame & 0xFF;
00393     uint8_t id_trame_high = (id_trame » 8) & 0xFF;
00394
00395     trame[0] = XB_V_START;
00396     trame[1] = XB_ADR_CURRENT_ROBOT;
00397     trame[2] = ad_dest;
00398     trame[3] = id_trame_low+4;
00399     trame[4] = id_trame_high+4;
00400     trame[5] = strlen(data)+4;
00401     trame[6] = code_fct;
00402
00403     for(size_t i = 0; i < strlen(data); i++){
00404         trame[i+7] = data[i];
00405     }
00406
00407
00408     for(int i=0; i < length_trame; i++){
00409         trame_int[i] = int(trame[i]);
00410     }
00411     int crc = crc16(trame_int, strlen(data)+6);
00412     uint8_t crc_low = crc & 0xFF;
00413     uint8_t crc_high = (crc » 8) & 0xFF;
00414
00415     trame[strlen(data)+7] = crc_low;
00416     trame[strlen(data)+8] = crc_high;
00417     trame[strlen(data)+9] = XB_V_END;
00418
00419     serial.writeBytes(trame, length_trame);
00420     logXbee « "(sendTrame) envoi de la trame n°" « dec « id_trame_low+id_trame_high  « " effectué avec
      succès" « mendl;
00421
00422     trames_envoyees[code_fct] = trames_envoyees[code_fct]+1;
00423
00424     return XB_TRAME_E_SUCCESS;
00425 }
00426
00437 int XBee::processTrame(vector<int> trame_recue){
00438
00439     if(!isTrameSizeCorrect(trame_recue)){
00440         logXbee « "/!\\ (process trame) erreur " « XB_TRAME_E_SIZE « " : taille de la trame incorrecte
      ou non concordante " « mendl;
00441         return XB_TRAME_E_SIZE;
00442     }
00443     Trame_t trame = {
00444         .start_seq = trame_recue[0],
00445         .adr_emetteur = trame_recue[1],
00446         .adr_dest = trame_recue[2],
00447         .id_trame_low = trame_recue[3]-4,
00448         .id_trame_high = trame_recue[4]-4,
00449         .nb_octets_msg = trame_recue[5]-4,
00450         .code_fct = trame_recue[6],
00451         .crc_low = trame_recue[3+trame_recue[4]],
00452         .crc_high = trame_recue[4+trame_recue[4]],
00453         .end_seq = trame_recue[5+trame_recue[4]]
00454     };
00455
00456     vector<int> data {};
00457
00458     for(uint8_t i = 0; i < trame.nb_octets_msg; i++){
00459        data.push_back(trame_recue[7+i]);
00460     }
00461
```

```
00462     trame.param = data;
00463 
00464     afficherTrameRecue(trame);
00465 
00466     int decoupe_trame[trame_recue[4]+6];
00467 
00468     for(uint8_t i = 0; i < trame_recue[4]+3; i++){
00469         decoupe_trame[i] = trame_recue[i];
00470     }
00471 
00472     if(!isStartSeqCorrect(trame.start_seq)){
00473         logXbee « "/!\\ (process trame) erreur " « XB_TRAME_E_START « " : premier caractère de la
      trame incorrect " « mendl;
00474         return XB_TRAME_E_START;
00475     }
00476 
00477     if(!isEndSeqCorrect(trame.end_seq)){
00478         logXbee « "/!\\ (process trame) erreur " « XB_TRAME_E_END « " : dernier caractère de la trame
      incorrect " « mendl;
00479         return XB_TRAME_E_END;
00480     }
00481 
00482     if(!isCRCCorrect(trame.crc_low, trame.crc_high, decoupe_trame, trame_recue[4]+2)){
00483         logXbee « "/!\\ (process trame) erreur " « XB_TRAME_E_CRC « " : valeur du CRC incorrecte " «
      mendl;
00484         return XB_TRAME_E_CRC;
00485     }
00486 
00487     if(!isExpCorrect(trame.adr_emetteur)){
00488         logXbee « "/!\\ (process trame) erreur " « XB_TRAME_E_EXP « " : adresse de l'expéditeur
      incorrecte ou inconnue " « mendl;
00489         return XB_TRAME_E_EXP;
00490     }
00491 
00492     if(!isDestCorrect(trame.adr_dest)){
00493         logXbee « "/!\\ (process trame) erreur " « XB_TRAME_E_DEST « " : adresse du destinataire
      incorrecte ou inconnue " « mendl;
00494         return XB_TRAME_E_DEST;
00495     }
00496 
00497     processCodeFct(trame.code_fct, trame.adr_emetteur);
00498 
00499     logXbee « "(process trame) trame n°" « trame.id_trame_high+trame.id_trame_low « "a été traitée
      avec succès " « mendl;
00500 
00501     return XB_TRAME_E_SUCCESS;
00502 }
00503 
00510 int XBee::processCodeFct(int code_fct, int exp){
00511     if(!isCodeFctCorrect(code_fct)){
00512         logXbee « "/!\\ (process code fonction) erreur " « XB_FCT_E_NOT_FOUND « " : code fonction
      incorrect " « mendl;
00513         return XB_FCT_E_NOT_FOUND;
00514     }
00515     char msg[1];
00516     switch(code_fct){
00517        case XB_FCT_TEST_ALIVE :
00518            msg[0] = {XB_V_ACK};
00519            sendTrame(exp, XB_FCT_TEST_ALIVE, msg);
00520            break;
00521 
00522        default :
00523            logXbee « "/!\\ (process code fonction) erreur " « XB_FCT_E_NONE_REACHABLE « " : code
      fonction existant mais ne déclenchant aucune action  " « mendl;
00524            return XB_FCT_E_NONE_REACHABLE;
00525     }
00526 
00527     trames_envoyees[code_fct] = trames_envoyees[code_fct]-1;
00528     logXbee « "(process code fonction) code fonction n°" « code_fct « " traité avec succès" « mendl;
00529     return XB_FCT_E_SUCCESS;
00530 }
00531 
00538 bool XBee::isCodeFctCorrect(int code_fct){
00539     int size_list_code_fct = sizeof(XB_LIST_CODE_FCT)/sizeof(XB_LIST_CODE_FCT[0]), i = 0;
00540 
00541     while(i < size_list_code_fct){
00542         if(XB_LIST_CODE_FCT[i] == code_fct)
00543             return true;
00544         i++;
00545     }
00546 
00547     return false;
00548 }
00549 
00556 bool XBee::isTrameSizeCorrect(vector<int> trame){
00557     if(trame.size() > 10 && trame.size() == trame[4]+5)
00558         return true;
```

```
00559
00560     return false;
00561 }
00562
00569 bool XBee::isExpCorrect(int exp){
00570     int size_list_addr = sizeof(XB_LIST_ADR)/sizeof(XB_LIST_ADR[0]), i = 0;
00571
00572     while(i < size_list_addr){
00573         if(XB_LIST_ADR[i] == exp)
00574             return true;
00575
00576         i++;
00577     }
00578
00579     return false;
00580 }
00581
00588 bool XBee::isDestCorrect(int dest){
00589     int size_list_addr = sizeof(XB_LIST_ADR)/sizeof(XB_LIST_ADR[0]), i = 0;
00590
00591     while(i < size_list_addr){
00592         if(XB_LIST_ADR[i] == dest)
00593             return true;
00594
00595         i++;
00596     }
00597
00598     return false;
00599 }
00600
00607 bool XBee::isStartSeqCorrect(int value){
00608     if(value == XB_V_START)
00609         return true;
00610
00611     return false;
00612 }
00613
00620 bool XBee::isEndSeqCorrect(int value){
00621     if(value == XB_V_END)
00622         return true;
00623
00624     return false;
00625 }
00626
00636 bool XBee::isCRCCorrect(uint8_t crc_low, uint8_t crc_high, int trame[], int trame_size){
00637     int crc = crc16(trame, trame_size);
00638
00639     uint8_t newcrc_low = crc & 0xFF;
00640     uint8_t newcrc_high = (crc » 8) & 0xFF;
00641
00642     if(newcrc_low == crc_low && newcrc_high == crc_high)
00643         return true;
00644
00645     return false;
00646 }
00647
00652 vector<int> XBee::readBuffer(){
00653     char *reponse(0);
00654     unsigned int timeout = 100;
00655     reponse = new char;
00656     vector<int> rep;
00657     delay(1);
00658     int i = 0;
00659     while(serial.available() > 0){
00660         i++;
00661         serial.readChar(reponse, timeout);
00662         rep.push_back(*reponse);
00663     }
00664     delete reponse;
00665     reponse = 0;
00666
00667     return rep;
00668 }
00669
00670
00675 string XBee::readString() {
00676      char *reponse(0);
00677      unsigned int timeout = 100;
00678      reponse = new char;
00679      string rep;
00680      delay(1);
00681      int i = 0;
00682
00683      while(serial.available() > 0){
00684     i++;
00685     serial.readChar(reponse, timeout);
00686     rep += *reponse;
```

```
00687       }
00688
00689        delete reponse;
00690        reponse = 0;
00691        return rep;
00692
00693 }
00694
00698 void XBee::waitForATrame(){
00699    vector<int> rep;
00700
00701    while(true){
00702      rep.clear();
00703
00704      delay(1/100);
00705
00706      if(serial.available() > 0){
00707        rep = readBuffer();
00708        subTrame(rep);
00709      }
00710    }
00711 }
00712
00725 int XBee::subTrame(vector<int> msg_recu){
00726
00727     vector<int> list_start_seq {};
00728     vector<int> list_end_seq {};
00729     vector<int> decoupe {};
00730     int decoupe_retour;
00731
00732     for(uint8_t i = 0; i < msg_recu.size(); i++){
00733         if(msg_recu[i] == XB_V_START)
00734             list_start_seq.push_back(i);
00735
00736         if(msg_recu[i] == XB_V_END)
00737             list_end_seq.push_back(i);
00738     }
00739
00740     if(list_start_seq.size() == 0 || list_end_seq.size() == 0){
00741         logXbee « "/!\\ (découpe trame) erreur " « XB_SUB_TRAME_E_NULL « " : aucun caractère de début
      et/ou de fin n'est présent dans le message reçu " « mendl;
00742         return XB_SUB_TRAME_E_NULL;
00743     }
00744
00745     if(list_start_seq.size() != list_end_seq.size()){
00746         logXbee « "/!\\ (découpe trame) erreur " « XB_SUB_TRAME_E_SIZE « " : les caractères de début
      et de fin de trame ne sont pas au même nombre " « mendl;
00747         return XB_SUB_TRAME_E_SIZE;
00748     }
00749
00750     for(uint8_t i = 0; i < list_start_seq.size(); i++){
00751         if(list_start_seq[i] > list_end_seq[i]){
00752             logXbee « "/!\\ (découpe trame) erreur " « XB_SUB_TRAME_E_REPARTITION « " : certains
      caractères de début de trame sont placés après des caractères de fin de trame " « mendl;
00753             return XB_SUB_TRAME_E_REPARTITION;
00754         }
00755
00756         if(i != 0){
00757             if(list_start_seq[i] != list_end_seq[i-1]-1){
00758                 logXbee « "/!\\ (découpe trame) erreur " « XB_SUB_TRAME_E_DECOUPAGE « " : des
      caractères inconnus sont placés entre deux trames " « mendl;
00759                 return XB_SUB_TRAME_E_DECOUPAGE;
00760             }
00761         }
00762     }
00763
00764
00765     if(list_start_seq[0] != 0){
00766         logXbee « "/!\\ (découpe trame) erreur " « XB_SUB_TRAME_E_START « " : le premier caractère lu
      dans le buffer n'est pas celui d'un début de trame " « mendl;
00767         return XB_SUB_TRAME_E_START;
00768     }
00769
00770     if(list_end_seq[list_end_seq.size()-1] != msg_recu.size()-1){
00771         logXbee « "/!\\ (découpe trame) erreur " « XB_SUB_TRAME_E_END « " : le dernier caractère lu
      dans le buffer n'est pas celui d'une fin de trame " « mendl;
00772         return XB_SUB_TRAME_E_END;
00773     }
00774
00775     for(uint8_t i = 0; i < list_start_seq.size(); i++){
00776        decoupe.clear();
00777        decoupe = slice(msg_recu, list_start_seq[i], list_end_seq[i]);
00778        decoupe_retour = processTrame(decoupe);
00779     }
00780
00781     logXbee « "(découpe trame) découpage des trames effectué avec succès" « mendl;
00782     return XB_SUB_TRAME_E_SUCCESS;
```

```
00783 }
00784
00788 void XBee::sendHeartbeat(){
00789     char* msg;
00790     msg[0] = XB_V_ACK;
00791
00792     while(true){
00793       delay(3);
00794       sendTrame(XB_ADR_ROBOT_02, XB_FCT_TEST_ALIVE, msg);
00795    }
00796 }
00797
00801 int XBee::isXbeeResponding(){
00802     int size_list_code_fct = sizeof(XB_LIST_CODE_FCT)/sizeof(XB_LIST_CODE_FCT[0]);
00803     while(true){
00804       delay(3);
00805       for(int i = 0; i < size_list_code_fct; i++){
00806           if(trames_envoyees[XB_LIST_CODE_FCT[i]] == 0){
00807               logXbee « "(verif reponse) les trames envoyées portant le code fonction " «
       XB_LIST_CODE_FCT[i] « " ont toutes reçues une réponse" « mendl;
00808           }else{
00809               logXbee « "(verif reponse) /!\\ les trames envoyées portant le code fonction " «
       XB_LIST_CODE_FCT[i] « " n'ont pas toutes reçues une réponse" « mendl;
00810           }
00811       }
00812    }
00813 }
00814
00819 void XBee::sendMsg(string msg){
00820     serial.writeString(stringToChar(msg));
00821     logXbee « "(envoi message) message : " « msg « " envoyé avec succès" « mendl;
00822 }
00823
00829 char* XBee::stringToChar(string chaine){
00830     char* message = strcpy(new char[chaine.size() + 1], chaine.c_str());
00831     return message;
00832 }
00833
00839 string XBee::charToString(char* message){
00840     string chaine = string(message);
00841     return chaine;
00842 }
00843
00847 void XBee::afficherTrameRecue(Trame_t trame){
00848     cout « hex « showbase;
00849     cout « "\t-> Start seq : " « trame.start_seq « endl;
00850     cout « "\t-> Emetteur : " « trame.adr_emetteur « endl;
00851     cout « "\t-> Destinataire : " « trame.adr_dest « endl;
00852     cout « "\t-> Id trame  : " « trame.id_trame_low « " " « trame.crc_high « endl;
00853     cout « "\t-> Taille msg : " « trame.nb_octets_msg - 3 « endl;
00854     cout « "\t-> Code fct : " « trame.code_fct « endl;
00855     cout « "\t-> Data : ";
00856     print(trame.param);
00857     cout « "\t-> CRC : " « trame.crc_low « " " « trame.crc_high « endl;
00858     cout « "\t-> End seq : " « trame.end_seq « endl;
00859 }
00860
00865 void XBee::print(const vector<int> &v){
00866     copy(v.begin(), v.end(),
00867             ostream_iterator<int>(cout « hex, " "));
00868     cout « endl;
00869 }
00870
00878 vector<int> XBee::slice(const vector<int> &v, int a, int b){
00879     auto first = v.cbegin() + a;
00880     auto last = v.cbegin() + b + 1;
00881
00882     vector<int> vec(first, last);
00883     return vec;
00884 }
```

5.15 Référence du fichier xbeelib.h

Fichier d'en-tête de la classe XBee. Cette classe est utilisée afin de programmer les modules XBee en UART et de mettre en place des communications entre différents modules XBee.

```
#include "define.h"
#include "serialib.h"
#include "loglib.h"
```

```
#include <string>
#include <vector>
#include <iomanip>
#include <iostream>
#include <chrono>
#include <thread>
#include <iterator>
```

Classes

— class XBee

Cette classe est utilisée pour la communication entre un module XBee et une RaspberryPi et entre plusieurs modules XBee.

— struct XBee::Trame_t

5.15.1 Description détaillée

Fichier d'en-tête de la classe XBee. Cette classe est utilisée afin de programmer les modules XBee en UART et de mettre en place des communications entre différents modules XBee.

Auteur

Samuel-Charles DITTE-DESTREE (samueldittedestree@protonmail.com)

Version

3.0

Date

10/03/2022

Définition dans le fichier xbeelib.h.

5.16 xbeelib.h

Aller à la documentation de ce fichier.

```
00001
00008 #ifndef XBEE_H
00009 #define XBEE_H
00010
00011 #include "define.h"
00012 #include "serialib.h"
00013 #include "loglib.h"
00014 #include <string>
00015 #include <vector>
00016 #include <iomanip>
00017 #include <iostream>
00018 #include <chrono>
00019 #include <thread>
00020 #include <iterator>
00021
00025 class XBee{
00026
00027 public:
00028
00029     // Constructeur de la classe
00030     XBee();
00031
```

```
00032     // Desctructeur de la classe
00033     ~XBee();
00034
00035     // Ouverture de la connexion série
00036     int openSerialConnection(int mode = 0);
00037
00038     // Fermeture de la connexion série
00039     void closeSerialConnection();
00040
00041     // Vérfication et paramétrage de la configuration AT par défaut du module
00042     int checkATConfig();
00043
00044     // Lecture de la réponse du module à une commande AT
00045     bool readATResponse(const char *value = XB_AT_R_EMPTY, int mode = 0);
00046
00047     // Envoi d'une commande AT
00048     bool sendATCommand(const char *command, const char *value, unsigned int mode = XB_AT_M_SET);
00049
00050     // Ecriture de la configuration AT dans la mémoire flash du module
00051     bool writeATConfig();
00052
00053     // Création et envoi de la trame de message structurée
00054     int sendTrame(uint8_t ad_dest, uint8_t code_fct, char* data = 0x00);
00055
00056     // Envoi d'un objet string quelconque
00057     void sendMsg(std::string msg);
00058
00059     // Attente de trames dans le buffer Rx et appel des fonctions de traitement
00060     void waitForATrame();
00061
00062     // Envoi de la demande de battement de coeur
00063     void sendHeartbeat();
00064
00065     // Permet de vérifier si un message envoyé a reçu une réponse
00066     int isXbeeResponding();
00067
00068     // Conversion char* en string
00069     std::string charToString(char* message);
00070
00071     // Conversion string en char*
00072     char* stringToChar(std::string chaine);
00073
00074     // Affichage du contenu d'un vecteur d'entiers
00075     void print(const std::vector<int> &v);
00076
00077 private:
00078
00083     typedef struct{
00084       int start_seq;
00085       int adr_emetteur;
00086       int adr_dest;
00087       int id_trame_low;
00088       int id_trame_high;
00089       int nb_octets_msg;
00090       int code_fct;
00091       std::vector<int> param;
00092       int crc_low;
00093       int crc_high;
00094       int end_seq;
00096     } Trame_t;
00097
00098     // Entrée dans le mode de configuration AT
00099     bool enterATMode();
00100
00101     // Sortie du mode de configuration AT
00102     bool exitATMode();
00103
00104     // Lance une découverte réseau des modules Xbee
00105     bool discoverXbeeNetwork();
00106
00107     // Vérifie si l'adresse de l'expéditeur existe
00108     bool isExpCorrect(int exp);
00109
00110     // Vérifie si l'adresse de destination existe
00111     bool isDestCorrect(int dest);
00112
00113     // Vérifie si le code fonction existe
00114     bool isCodeFctCorrect(int code_fct);
00115
00116     // Vérifie si la taille de la trame est cohérente
00117     bool isTrameSizeCorrect(std::vector<int> trame);
00118
00119     // Découpe un ensemble de trames en trames uniques
00120     int subTrame(std::vector<int> msg_recu);
00121
00122     // Interprète le code fonction et exécute les fonctions associées
00123     int processCodeFct(int code_fct, int exp);
```

```
00124
00125     // Affichage des différents paramètres d'une structure Trame_t
00126     void afficherTrameRecue(Trame_t trame);
00127
00128     // Interprète et contrôle l'intégrité d'une trame reçue
00129     int processTrame(std::vector<int> trame);
00130
00131     // Lis le contenu du buffer Rx de la RaspberryPi
00132     std::vector<int> readBuffer();
00133
00134     // Lis le contenu du buffer Rx de la RaspberryPi
00135     std::string readString();
00136
00137     // Calcul du CRC16 Modbus de la trame
00138     int crc16(int trame[], uint8_t taille);
00139
00140     // Vérifie si le caractère de début de trame est correct
00141     bool isStartSeqCorrect(int value);
00142
00143     // Vérifie si le caractère de fin de trame est correct
00144     bool isEndSeqCorrect(int value);
00145
00146     // Vérifie si le CRC de la trame est correct
00147     bool isCRCCorrect(uint8_t crc_low, uint8_t crc_high, int trame[], int trame_size);
00148
00149     // Retard de temporisation dans l'exécution du code
00150     void delay(unsigned int time);
00151
00152     // Découpe un vecteur d'entiers en un sous vecteur
00153     std::vector<int> slice(const std::vector<int> &v, int a, int b);
00154
00155     // Variable calculant l'ID de la trame
00156     int ID_TRAME = 0;
00157
00158     // Variable permettant de définir la configuration série à utiliser
00159     int MODE = 0;
00160
00161     //vecteur contenant la liste des trames envoyées classées par destinataire et code fonction
00162     std::vector<int> trames_envoyees = {};
00163 };
00164
00165 #endif // XBEE_H
```

Index