Robotech 2021 - XBee

Generated by Doxygen 1.9.3

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 serialib Class Reference

This class is used for communication over a serial device.

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

Public Member Functions

- serialib ()

  Constructor of the class serialib.
- ∼serialib ()

  Destructor of the class serialib. It close the connection.
- char 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 sleep↩Duration_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.

3.1.1 Detailed Description

This class is used for communication over a serial device.

Definition at line 92 of file serialib.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 serialib()

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

Constructor of the class serialib.

Definition at line 29 of file serialib.cpp.

3.1.2.2 ∼serialib()

```
serialib::∼serialib ( )
```

Destructor of the class serialib. It close the connection.

Definition at line 47 of file serialib.cpp.

3.1.3 Member Function Documentation

3.1.3.1 available()

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

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

Returns

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

Definition at line 701 of file serialib.cpp.

3.1.3.2 clearDTR()

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

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

Returns

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

Definition at line 776 of file serialib.cpp.

3.1.3.3 clearRTS()

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

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

Returns

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

Definition at line 846 of file serialib.cpp.

3.1.3.4 closeDevice()

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

Close the connection with the current device.

Definition at line 316 of file serialib.cpp.

3.1.3.5 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.

Parameters

status	= true set DTR status = false unset DTR

Returns

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

Definition at line 736 of file serialib.cpp.

3.1.3.6 flushReceiver()

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

Empty receiver buffer.

Returns

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

Definition at line 682 of file serialib.cpp.

3.1.3.7 isCTS()

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

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

Returns

Return true if CTS is set otherwise false

Definition at line 871 of file serialib.cpp.

3.1.3.8 isDCD()

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

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

Returns

true if DCD is set

false otherwise

Definition at line 919 of file serialib.cpp.

3.1.3.9 isDeviceOpen()

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

Definition at line 303 of file serialib.cpp.

3.1.3.10 isDSR()

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

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

Returns

Return true if DTR is set otherwise false

Definition at line 893 of file serialib.cpp.

3.1.3.11 isDTR()

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

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

Returns

Return true if CTS is set otherwise false

Definition at line 962 of file serialib.cpp.

3.1.3.12 isRI()

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

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

Returns

Return true if RING is set otherwise false

Definition at line 940 of file serialib.cpp.

3.1.3.13 isRTS()

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

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

Returns

Return true if RTS is set otherwise false

Definition at line 983 of file serialib.cpp.

3.1.3.14 openDevice()

```
char 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.

Parameters

Device	: Port name (COM1, COM2, ... for Windows) or (/dev/ttyS0, /dev/ttyACM0, /dev/ttyUSB0 ... for linux)

Parameters

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)`

Returns

1 success

-1 device not found

-2 error while opening the device

-3 error while getting port parameters

-4 Speed (Bauds) not recognized

-5 error while writing port parameters

-6 error while writing timeout parameters

-7 Databits not recognized

-8 Stopbits not recognized

-9 Parity not recognized

Definition at line 128 of file serialib.cpp.

3.1.3.15 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)

Parameters

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

Returns

>=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

Definition at line 614 of file serialib.cpp.

3.1.3.16 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.

Parameters

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)

Returns

1 success

0 Timeout reached

-1 error while setting the Timeout

-2 error while reading the byte

Definition at line 440 of file serialib.cpp.

3.1.3.17 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)

Parameters

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)

Returns

>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

Definition at line 540 of file serialib.cpp.

3.1.3.18 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.

Parameters

status	= true set DTR status = false unset DTR

Returns

false if the function fails

true if the function succeeds

Definition at line 804 of file serialib.cpp.

3.1.3.19 setDTR()

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

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

Returns

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

Definition at line 753 of file serialib.cpp.

3.1.3.20 setRTS()

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

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

Returns

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

Definition at line 821 of file serialib.cpp.

3.1.3.21 writeBytes()

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

Write an array of data on the current serial port.

Parameters

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

Returns

1 success

-1 error while writting data

Definition at line 408 of file serialib.cpp.

3.1.3.22 writeChar()

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

Write a char on the current serial port.

Parameters

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

Returns

1 success

-1 error while writting data

Definition at line 342 of file serialib.cpp.

3.1.3.23 writeString()

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

Write a string on the current serial port.

Parameters

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

Returns

1 success

-1 error while writting data

Definition at line 374 of file serialib.cpp.

The documentation for this class was generated from the following files:

- serialib.h
- serialib.cpp

3.2 timeOut Class Reference

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

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

Public Member Functions

- 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.

3.2.1 Detailed Description

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

Definition at line 245 of file serialib.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 timeOut()

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

Constructor of the class timeOut.

Definition at line 1010 of file serialib.cpp.

3.2.3 Member Function Documentation

3.2.3.1 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.

Returns

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

Definition at line 1038 of file serialib.cpp.

3.2.3.2 initTimer()

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

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

Definition at line 1018 of file serialib.cpp.

The documentation for this class was generated from the following files:

- serialib.h
- serialib.cpp

3.3 Trame Struct Reference

Structure permettant de définir une trame de message reçue et envoyée.

Public Attributes

- int id_exp
- int id_dest
- int code_fct
- int id_trame
- int size
- vector< char > data

3.3.1 Detailed Description

Structure permettant de définir une trame de message reçue et envoyée.

Definition at line 18 of file xbee.cpp.

3.3.2 Member Data Documentation

3.3.2.1 code_fct

```
int Trame::code_fct
```

Definition at line 21 of file xbee.cpp.

3.3.2.2 data

```
vector<char> Trame::data
```

Definition at line 24 of file xbee.cpp.

3.3.2.3 id_dest

```
int Trame::id_dest
```

Definition at line 20 of file xbee.cpp.

3.3.2.4 id_exp

```
int Trame::id_exp
```

Definition at line 19 of file xbee.cpp.

3.3.2.5 id_trame

```
int Trame::id_trame
```

Definition at line 22 of file xbee.cpp.

3.3.2.6 size

```
int Trame::size
```

Definition at line 23 of file xbee.cpp.

The documentation for this struct was generated from the following file:

- xbee.cpp

3.4 xbee Class Reference

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

```
#include <xbee.h>
```

Public Member Functions

- xbee ()
  Constructeur de la classe xbee.
- ∼xbee ()
  Destructeur de la classe xbee.
- int openSerialConnection ()
  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.
- bool enterATMode ()
  Fonction permettant d'entrer dans le mode AT.
- bool exitATMode ()
  Fonction permettant de sortir du mode AT.
- int checkATConfig ()
  Vérification et paramétrage de la bonne configuration pour le module XBee.
- bool readATResponse (const char ∗value=AT_EMPTY_VALUE)
  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)
  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.
- void sendTrame (char ad_dest, char code_fct, char data[ ])
  Fonction permettant d'envoyer une trame de message structurée via UART en XBee.

3.4.1 Detailed Description

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

Definition at line 22 of file xbee.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 xbee()

```
xbee::xbee ( )
```

Constructeur de la classe xbee.

Definition at line 33 of file xbee.cpp.

3.4.2.2 ∼xbee()

```
xbee::∼xbee ( )
```

Destructeur de la classe xbee.

Definition at line 38 of file xbee.cpp.

3.4.3 Member Function Documentation

3.4.3.1 checkATConfig()

```
int xbee::checkATConfig ( )
```

Vérification et paramétrage de la bonne configuration pour le module XBee.

Returns

0 succès

-1 impossible d'entrer dans le mode AT

-2 impossible de configurer le mode API

-3 impossible de configurer le baudrate

-4 impossible de configurer le paramètre de chiffrement AES

-5 impossible de configurer la clé de chiffrement AES

-6 impossible de configurer le canal de découverte réseau

-7 impossible de configurer l'ID du réseau

-8 impossible de configurer le mode coordinateur

-9 impossible de configurer le nombre de bits de parité

-10 impossible de configurer l'addresse source 16bits

-11 impossible de sortir du mode AT

-12 impossible d'écrire les paramètres dans la mémoire flash

Definition at line 91 of file xbee.cpp.

3.4.3.2 closeSerialConnection()

```
void xbee::closeSerialConnection ( )
```

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

Definition at line 67 of file xbee.cpp.

3.4.3.3 enterATMode()

```
bool xbee::enterATMode ( )
```

Fonction permettant d'entrer dans le mode AT.

Returns

true la réponse du module XBee est celle attendue

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

Definition at line 173 of file xbee.cpp.

3.4.3.4 exitATMode()

```
bool xbee::exitATMode ( )
```

Fonction permettant de sortir du mode AT.

Returns

true la réponse du module XBee est celle attendue

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

Definition at line 186 of file xbee.cpp.

3.4.3.5 openSerialConnection()

```
int xbee::openSerialConnection ( )
```

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

Returns

1 succès

-1 port série non trouvé

-2 erreur lors de l'ouverture du port série

-3 erreur lors de la récupération des informations du port série

-4 baudrate non reconnu

-5 erreur lors de l'écriture de la configuration du port série

-6 erreur lors de l'écriture du timeout

-7 databits non reconnus

-8 stopbits non reconnus

-9 parité non reconnue

Definition at line 57 of file xbee.cpp.

3.4.3.6 readATResponse()

```
bool xbee::readATResponse (
             const char * value = AT_EMPTY_VALUE )
```

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

Parameters

value	: la valeur de réponse attendue pour la commande envoyée

Returns

true la réponse du module XBee est celle attendue

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

Definition at line 147 of file xbee.cpp.

3.4.3.7 sendATCommand()

```
bool xbee::sendATCommand (
             const char * command,
             const char * value,
             unsigned int mode )
```

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

Parameters

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)

Returns

true la réponse du module XBee est celle attendue

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

Definition at line 213 of file xbee.cpp.

3.4.3.8 sendTrame()

```
void xbee::sendTrame (
             char ad_dest,
             char code_fct,
             char data[ ] )
```

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

Parameters

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

Definition at line 266 of file xbee.cpp.

3.4.3.9 writeATConfig()

```
bool xbee::writeATConfig ( )
```

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

Returns

true la réponse du module XBee est celle attendue

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

Definition at line 198 of file xbee.cpp.

The documentation for this class was generated from the following files:

- xbee.h
- xbee.cpp

Chapter 4

File Documentation

4.1 define.h File Reference

Macros

- #define SERIAL_PORT "/dev/ttyAMA0"
- #define BAUDRATE 9600
- #define DATABITS SERIAL_DATABITS_8
- #define PARITY SERIAL_PARITY_NONE
- #define STOPBITS SERIAL_STOPBITS_1
- #define BROADCAST 0x0A
- #define ROBOT_01 0x01
- #define ROBOT_02 0x02
- #define CURRENT_ROBOT ROBOT_01
- #define START_SEQ 0x02
- #define END_SEQ 0x04
- #define TEST_ALIVE 0x01
- #define AT_ENTER "+++"
- #define AT_EXIT "ATCN"
- #define AT_END_LINE "\r"
- #define AT_WRITE_CONFIG "ATWR"
- #define AT_GET_API "ATAP"
- #define AT_GET_BAUDRATE "ATBD"
- #define AT_GET_AES "ATEE"
- #define AT_GET_AES_KEY "ATKY"
- #define AT_GET_CHANEL "ATCH"
- #define AT_GET_PAN_ID "ATID"
- #define AT_GET_COORDINATOR "ATCE"
- #define AT_GET_PARITY "ATNB"
- #define AT_GET_16BIT_SOURCE_ADDR "ATMY"
- #define AT_GET_LOW_DEST_ADDR "ATDL"
- #define AT_VALUE_API "1"
- #define AT_VALUE_BAUDRATE "3"
- #define AT_VALUE_AES "1"
- #define AT_VALUE_AES_KEY "32303032"
- #define AT_VALUE_CHANEL "C"
- #define AT_VALUE_PAN_ID "3332"
- #define AT_VALUE_COORDINATOR "0"

- #define AT_VALUE_PARITY "0"
- #define AT_VALUE_16BIT_SOURCE_ADDR "2"
- #define AT_VALUE_LOW_DEST_ADDR "1"
- #define AT_EMPTY_VALUE ""
- #define AT_SUCCESS_VALUE "OK\r"
- #define AT_ERROR_VALUE "ERROR\r"
- #define AT_MODE_GET 1
- #define AT_MODE_SET 2
- #define AT_ERROR_ENTER -1
- #define AT_ERROR_API -2
- #define AT_ERROR_BAUDRATE -3
- #define AT_ERROR_AES -4
- #define AT_ERROR_AES_KEY -13
- #define AT_ERROR_CHANEL -5
- #define AT_ERROR_PAN_ID -6
- #define AT_ERROR_COORDINATOR -7
- #define AT_ERROR_PARITY -8
- #define AT_ERROR_16BIT_SOURCE_ADDR -9
- #define AT_ERROR_LOW_DEST_ADDR -10
- #define AT_ERROR_EXIT -11
- #define AT_ERROR_WRITE_CONFIG -12
- #define AT_ERROR_SUCCESS 0
- #define ERROR_SUCCESS 0

4.1.1 Macro Definition Documentation

4.1.1.1 AT_EMPTY_VALUE

```
#define AT_EMPTY_VALUE ""
```

Definition at line 55 of file define.h.

4.1.1.2 AT_END_LINE

```
#define AT_END_LINE "\r"
```

Definition at line 30 of file define.h.

4.1.1.3 AT_ENTER

```
#define AT_ENTER "+++"
```

Definition at line 28 of file define.h.

4.1.1.4 AT_ERROR_16BIT_SOURCE_ADDR

```
#define AT_ERROR_16BIT_SOURCE_ADDR -9
```

Definition at line 72 of file define.h.

4.1.1.5 AT_ERROR_AES

```
#define AT_ERROR_AES -4
```

Definition at line 66 of file define.h.

4.1.1.6 AT_ERROR_AES_KEY

```
#define AT_ERROR_AES_KEY -13
```

Definition at line 67 of file define.h.

4.1.1.7 AT_ERROR_API

```
#define AT_ERROR_API -2
```

Definition at line 64 of file define.h.

4.1.1.8 AT_ERROR_BAUDRATE

```
#define AT_ERROR_BAUDRATE -3
```

Definition at line 65 of file define.h.

4.1.1.9 AT_ERROR_CHANEL

```
#define AT_ERROR_CHANEL -5
```

Definition at line 68 of file define.h.

4.1.1.10 AT_ERROR_COORDINATOR

```
#define AT_ERROR_COORDINATOR -7
```

Definition at line 70 of file define.h.

4.1.1.11 AT_ERROR_ENTER

```
#define AT_ERROR_ENTER -1
```

Definition at line 63 of file define.h.

4.1.1.12 AT_ERROR_EXIT

```
#define AT_ERROR_EXIT -11
```

Definition at line 74 of file define.h.

4.1.1.13 AT_ERROR_LOW_DEST_ADDR

```
#define AT_ERROR_LOW_DEST_ADDR -10
```

Definition at line 73 of file define.h.

4.1.1.14 AT_ERROR_PAN_ID

```
#define AT_ERROR_PAN_ID -6
```

Definition at line 69 of file define.h.

4.1.1.15 AT_ERROR_PARITY

```
#define AT_ERROR_PARITY -8
```

Definition at line 71 of file define.h.

4.1.1.16 AT_ERROR_SUCCESS

```
#define AT_ERROR_SUCCESS 0
```

Definition at line 76 of file define.h.

4.1.1.17 AT_ERROR_VALUE

```
#define AT_ERROR_VALUE "ERROR\r"
```

Definition at line 57 of file define.h.

4.1.1.18 AT_ERROR_WRITE_CONFIG

```
#define AT_ERROR_WRITE_CONFIG -12
```

Definition at line 75 of file define.h.

4.1.1.19 AT_EXIT

```
#define AT_EXIT "ATCN"
```

Definition at line 29 of file define.h.

4.1.1.20 AT_GET_16BIT_SOURCE_ADDR

```
#define AT_GET_16BIT_SOURCE_ADDR "ATMY"
```

Definition at line 41 of file define.h.

4.1.1.21 AT_GET_AES

```
#define AT_GET_AES "ATEE"
```

Definition at line 35 of file define.h.

4.1.1.22 AT_GET_AES_KEY

```
#define AT_GET_AES_KEY "ATKY"
```

Definition at line 36 of file define.h.

4.1.1.23 AT_GET_API

```
#define AT_GET_API "ATAP"
```

Definition at line 33 of file define.h.

4.1.1.24 AT_GET_BAUDRATE

```
#define AT_GET_BAUDRATE "ATBD"
```

Definition at line 34 of file define.h.

4.1.1.25 AT_GET_CHANEL

```
#define AT_GET_CHANEL "ATCH"
```

Definition at line 37 of file define.h.

4.1.1.26 AT_GET_COORDINATOR

```
#define AT_GET_COORDINATOR "ATCE"
```

Definition at line 39 of file define.h.

4.1.1.27 AT_GET_LOW_DEST_ADDR

```
#define AT_GET_LOW_DEST_ADDR "ATDL"
```

Definition at line 42 of file define.h.

4.1.1.28 AT_GET_PAN_ID

```
#define AT_GET_PAN_ID "ATID"
```

Definition at line 38 of file define.h.

4.1.1.29 AT_GET_PARITY

```
#define AT_GET_PARITY "ATNB"
```

Definition at line 40 of file define.h.

4.1.1.30 AT_MODE_GET

```
#define AT_MODE_GET 1
```

Definition at line 59 of file define.h.

4.1.1.31 AT_MODE_SET

```
#define AT_MODE_SET 2
```

Definition at line 60 of file define.h.

4.1.1.32 AT_SUCCESS_VALUE

```
#define AT_SUCCESS_VALUE "OK\r"
```

Definition at line 56 of file define.h.

4.1.1.33 AT_VALUE_16BIT_SOURCE_ADDR

```
#define AT_VALUE_16BIT_SOURCE_ADDR "2"
```

Definition at line 52 of file define.h.

4.1.1.34 AT_VALUE_AES

```
#define AT_VALUE_AES "1"
```

Definition at line 46 of file define.h.

4.1.1.35 AT_VALUE_AES_KEY

```
#define AT_VALUE_AES_KEY "32303032"
```

Definition at line 47 of file define.h.

4.1.1.36 AT_VALUE_API

```
#define AT_VALUE_API "1"
```

Definition at line 44 of file define.h.

4.1.1.37 AT_VALUE_BAUDRATE

```
#define AT_VALUE_BAUDRATE "3"
```

Definition at line 45 of file define.h.

4.1.1.38 AT_VALUE_CHANEL

```
#define AT_VALUE_CHANEL "C"
```

Definition at line 48 of file define.h.

4.1.1.39 AT_VALUE_COORDINATOR

```
#define AT_VALUE_COORDINATOR "0"
```

Definition at line 50 of file define.h.

4.1.1.40 AT_VALUE_LOW_DEST_ADDR

```
#define AT_VALUE_LOW_DEST_ADDR "1"
```

Definition at line 53 of file define.h.

4.1.1.41 AT_VALUE_PAN_ID

```
#define AT_VALUE_PAN_ID "3332"
```

Definition at line 49 of file define.h.

4.1.1.42 AT_VALUE_PARITY

```
#define AT_VALUE_PARITY "0"
```

Definition at line 51 of file define.h.

4.1.1.43 AT_WRITE_CONFIG

```
#define AT_WRITE_CONFIG "ATWR"
```

Definition at line 31 of file define.h.

4.1.1.44 BAUDRATE

```
#define BAUDRATE 9600
```

Definition at line 6 of file define.h.

4.1.1.45 BROADCAST

```
#define BROADCAST 0x0A
```

Definition at line 12 of file define.h.

4.1.1.46 CURRENT_ROBOT

```
#define CURRENT_ROBOT ROBOT_01
```

Definition at line 16 of file define.h.

4.1.1.47 DATABITS

```
#define DATABITS SERIAL_DATABITS_8
```

Definition at line 7 of file define.h.

4.1.1.48 END_SEQ

```
#define END_SEQ 0x04
```

Definition at line 20 of file define.h.

4.1.1.49 ERROR_SUCCESS

```
#define ERROR_SUCCESS 0
```

Definition at line 79 of file define.h.

4.1.1.50 PARITY

```
#define PARITY SERIAL_PARITY_NONE
```

Definition at line 8 of file define.h.

4.1.1.51 ROBOT_01

```
#define ROBOT_01 0x01
```

Definition at line 13 of file define.h.

4.1.1.52 ROBOT_02

```
#define ROBOT_02 0x02
```

Definition at line 14 of file define.h.

4.1.1.53 SERIAL_PORT

```
#define SERIAL_PORT "/dev/ttyAMA0"
```

Definition at line 5 of file define.h.

4.1.1.54 START_SEQ

```
#define START_SEQ 0x02
```

Definition at line 19 of file define.h.

4.1.1.55 STOPBITS

```
#define STOPBITS SERIAL_STOPBITS_1
```

Definition at line 9 of file define.h.

4.1.1.56 TEST_ALIVE

```
#define TEST_ALIVE 0x01
```

Definition at line 25 of file define.h.

4.2 define.h

Go to the documentation of this file.

```
#ifndef DEFINE_XBEE_H
#define DEFINE_XBEE_H

// Paramètres du port série
#define SERIAL_PORT "/dev/ttyAMA0"
#define BAUDRATE 9600
#define DATABITS SERIAL_DATABITS_8
#define PARITY SERIAL_PARITY_NONE
#define STOPBITS SERIAL_STOPBITS_1

// Addresses des robots
#define BROADCAST 0x0A
#define ROBOT_01 0x01
#define ROBOT_02 0x02

#define CURRENT_ROBOT ROBOT_01

// Paramètres de la trame message
#define START_SEQ 0x02
#define END_SEQ 0x04

static unsigned char ID_TRAME = 0x00;

// Codes fonctions
#define TEST_ALIVE 0x01

// Commandes AT
#define AT_ENTER "+++"
#define AT_EXIT "ATCN"
#define AT_END_LINE "\r"
#define AT_WRITE_CONFIG "ATWR"

#define AT_GET_API "ATAP"
#define AT_GET_BAUDRATE "ATBD"
#define AT_GET_AES "ATEE"
#define AT_GET_AES_KEY "ATKY"
#define AT_GET_CHANEL "ATCH"
#define AT_GET_PAN_ID "ATID"
#define AT_GET_COORDINATOR "ATCE"
#define AT_GET_PARITY "ATNB"
#define AT_GET_16BIT_SOURCE_ADDR "ATMY"
#define AT_GET_LOW_DEST_ADDR "ATDL"

#define AT_VALUE_API "1"
#define AT_VALUE_BAUDRATE "3"
#define AT_VALUE_AES "1"
#define AT_VALUE_AES_KEY "32303032"
#define AT_VALUE_CHANEL "C"
#define AT_VALUE_PAN_ID "3332"
#define AT_VALUE_COORDINATOR "0"
#define AT_VALUE_PARITY "0"
#define AT_VALUE_16BIT_SOURCE_ADDR "2"
#define AT_VALUE_LOW_DEST_ADDR "1"

#define AT_EMPTY_VALUE ""
#define AT_SUCCESS_VALUE "OK\r"
#define AT_ERROR_VALUE "ERROR\r"

#define AT_MODE_GET 1
#define AT_MODE_SET 2

// Codes d'erreurs en mode AT
#define AT_ERROR_ENTER -1
#define AT_ERROR_API -2
#define AT_ERROR_BAUDRATE -3
#define AT_ERROR_AES -4
#define AT_ERROR_AES_KEY -13
#define AT_ERROR_CHANEL -5
#define AT_ERROR_PAN_ID -6
#define AT_ERROR_COORDINATOR -7
#define AT_ERROR_PARITY -8
#define AT_ERROR_16BIT_SOURCE_ADDR -9
#define AT_ERROR_LOW_DEST_ADDR -10
#define AT_ERROR_EXIT -11
#define AT_ERROR_WRITE_CONFIG -12
#define AT_ERROR_SUCCESS 0

// Codes d'erreurs
#define ERROR_SUCCESS 0

#endif
```

4.3 main.cpp File Reference

```
#include "xbee.h"
```

Functions

- int main (int argc, char ∗argv[ ])

4.3.1 Function Documentation

4.3.1.1 main()

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

Definition at line 5 of file main.cpp.

4.4 main.cpp

Go to the documentation of this file.

```
00001 #include "xbee.h"
00002
00003 using namespace std;
00004
00005 int main(int argc, char *argv[]){
00006
00007     xbee xbee;
00008
00009     int error_open_connection = xbee.openSerialConnection();
00010
00011     if(error_open_connection != 1)
00012         cout « ": Erreur de connexion à " « SERIAL_PORT « " [Code erreur : " « error_open_connection «
       "]." « endl;
00013     else
00014         cout « ": Connexion ouverte avec succès sur le port \"" « SERIAL_PORT « "\".\n" « endl;
00015
00016
00017     int error_configuration = xbee.checkATConfig();
00018
00019     if(error_configuration == AT_ERROR_SUCCESS)
00020         cout « "Configuration AT réussie." « endl;
00021     else
00022         cout « "Configuration AT non réussie | [Code erreur : " « error_configuration « "]" « endl;
00023
00024     char msg[1];
00025     msg[0] = 0x02;
00026
00027     //sendTrame(serial, (char) ROBOT_02, (char) TEST_ALIVE, msg);
00028
00029     return EXIT_SUCCESS;
00030 }
```

4.5 serialib.cpp File Reference

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

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

4.5.1 Detailed Description

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

Author

Philippe Lucidarme (University of Angers)

Version

2.0

Date

december the 27th of 2019

Definition in file serialib.cpp.

4.6 serialib.cpp

Go to the documentation of this file.

```
#include "serialib.h"



//_____________________________________
// ::: Constructors and destructors :::


serialib::serialib()
{
#if defined (_WIN32) || defined( _WIN64)
    // Set default value for RTS and DTR (Windows only)
    currentStateRTS=true;
    currentStateDTR=true;
    hSerial = INVALID_HANDLE_VALUE;
#endif
#if defined (__linux__) || defined(__APPLE__)
    fd = -1;
#endif
}
```

```


// Class desctructor
serialib::~serialib()
{
    closeDevice();
}



//_________________________________________
// ::: Configuration and initialization :::



char serialib::openDevice(const char *Device, const unsigned int Bauds,
                          SerialDataBits Databits,
                          SerialParity Parity,
                          SerialStopBits Stopbits) {
#if defined (_WIN32) || defined( _WIN64)
    // Open serial port
    hSerial = CreateFileA(Device,GENERIC_READ |
 GENERIC_WRITE,0,0,OPEN_EXISTING,/*FILE_ATTRIBUTE_NORMAL*/0,0);
    if(hSerial==INVALID_HANDLE_VALUE) {
        if(GetLastError()==ERROR_FILE_NOT_FOUND)
            return -1; // Device not found

        // Error while opening the device
        return -2;
    }

    // Set parameters

    // Structure for the port parameters
    DCB dcbSerialParams;
    dcbSerialParams.DCBlength=sizeof(dcbSerialParams);

    // Get the port parameters
    if (!GetCommState(hSerial, &dcbSerialParams)) return -3;

    // Set the speed (Bauds)
    switch (Bauds)
    {
    case 110  :     dcbSerialParams.BaudRate=CBR_110; break;
    case 300  :     dcbSerialParams.BaudRate=CBR_300; break;
    case 600  :     dcbSerialParams.BaudRate=CBR_600; break;
    case 1200 :     dcbSerialParams.BaudRate=CBR_1200; break;
    case 2400 :     dcbSerialParams.BaudRate=CBR_2400; break;
    case 4800 :     dcbSerialParams.BaudRate=CBR_4800; break;
    case 9600 :     dcbSerialParams.BaudRate=CBR_9600; break;
    case 14400 :    dcbSerialParams.BaudRate=CBR_14400; break;
    case 19200 :    dcbSerialParams.BaudRate=CBR_19200; break;
    case 38400 :    dcbSerialParams.BaudRate=CBR_38400; break;
    case 56000 :    dcbSerialParams.BaudRate=CBR_56000; break;
    case 57600 :    dcbSerialParams.BaudRate=CBR_57600; break;
    case 115200 :   dcbSerialParams.BaudRate=CBR_115200; break;
    case 128000 :   dcbSerialParams.BaudRate=CBR_128000; break;
    case 256000 :   dcbSerialParams.BaudRate=CBR_256000; break;
    default : return -4;
    }
    //select data size
    BYTE bytesize = 0;
    switch(Databits) {
        case SERIAL_DATABITS_5: bytesize = 5; break;
        case SERIAL_DATABITS_6: bytesize = 6; break;
        case SERIAL_DATABITS_7: bytesize = 7; break;
        case SERIAL_DATABITS_8: bytesize = 8; break;
        case SERIAL_DATABITS_16: bytesize = 16; break;
        default: return -7;
    }
    BYTE stopBits = 0;
    switch(Stopbits) {
        case SERIAL_STOPBITS_1: stopBits = ONESTOPBIT; break;
        case SERIAL_STOPBITS_1_5: stopBits = ONE5STOPBITS; break;
        case SERIAL_STOPBITS_2: stopBits = TWOSTOPBITS; break;
        default: return -8;
    }
    BYTE parity = 0;
    switch(Parity) {
        case SERIAL_PARITY_NONE: parity = NOPARITY; break;
        case SERIAL_PARITY_EVEN: parity = EVENPARITY; break;
        case SERIAL_PARITY_ODD: parity = ODDPARITY; break;
        case SERIAL_PARITY_MARK: parity = MARKPARITY; break;
        case SERIAL_PARITY_SPACE: parity = SPACEPARITY; break;
        default: return -9;
    }
    // configure byte size
    dcbSerialParams.ByteSize = bytesize;
    // configure stop bits
    dcbSerialParams.StopBits = stopBits;
    // configure parity
    dcbSerialParams.Parity = parity;

    // Write the parameters
    if(!SetCommState(hSerial, &dcbSerialParams)) return -5;

    // Set TimeOut

    // Set the Timeout parameters
    timeouts.ReadIntervalTimeout=0;
    // No TimeOut
    timeouts.ReadTotalTimeoutConstant=MAXDWORD;
    timeouts.ReadTotalTimeoutMultiplier=0;
    timeouts.WriteTotalTimeoutConstant=MAXDWORD;
    timeouts.WriteTotalTimeoutMultiplier=0;

    // Write the parameters
    if(!SetCommTimeouts(hSerial, &timeouts)) return -6;

    // Opening successfull
    return 1;
#endif
#if defined (__linux__) || defined(__APPLE__)
    // Structure with the device's options
    struct termios options;


    // Open device
    fd = open(Device, O_RDWR | O_NOCTTY | O_NDELAY);
    // If the device is not open, return -1
    if (fd == -1) return -2;
    // Open the device in nonblocking mode
    fcntl(fd, F_SETFL, FNDELAY);


    // Get the current options of the port
    tcgetattr(fd, &options);
    // Clear all the options
    bzero(&options, sizeof(options));

    // Prepare speed (Bauds)
    speed_t         Speed;
    switch (Bauds)
    {
    case 110  :     Speed=B110; break;
    case 300  :     Speed=B300; break;
    case 600  :     Speed=B600; break;
    case 1200 :     Speed=B1200; break;
    case 2400 :     Speed=B2400; break;
    case 4800 :     Speed=B4800; break;
    case 9600 :     Speed=B9600; break;
    case 19200 :    Speed=B19200; break;
    case 38400 :    Speed=B38400; break;
    case 57600 :    Speed=B57600; break;
    case 115200 :   Speed=B115200; break;
    default : return -4;
    }
    int databits_flag = 0;
    switch(Databits) {
        case SERIAL_DATABITS_5: databits_flag = CS5; break;
        case SERIAL_DATABITS_6: databits_flag = CS6; break;
        case SERIAL_DATABITS_7: databits_flag = CS7; break;
        case SERIAL_DATABITS_8: databits_flag = CS8; break;
        //16 bits and everything else not supported
        default: return -7;
    }
    int stopbits_flag = 0;
    switch(Stopbits) {
        case SERIAL_STOPBITS_1: stopbits_flag = 0; break;
        case SERIAL_STOPBITS_2: stopbits_flag = CSTOPB; break;
        //1.5 stopbits and everything else not supported
        default: return -8;
    }
    int parity_flag = 0;
    switch(Parity) {
        case SERIAL_PARITY_NONE: parity_flag = 0; break;
        case SERIAL_PARITY_EVEN: parity_flag = PARENB; break;
        case SERIAL_PARITY_ODD: parity_flag = (PARENB | PARODD); break;
        //mark and space parity not supported
        default: return -9;
    }

    // Set the baud rate
    cfsetispeed(&options, Speed);
    cfsetospeed(&options, Speed);
    // Configure the device : data bits, stop bits, parity, no control flow
    // Ignore modem control lines (CLOCAL) and Enable receiver (CREAD)
    options.c_cflag |= ( CLOCAL | CREAD | databits_flag | parity_flag | stopbits_flag);
    options.c_iflag |= ( IGNPAR | IGNBRK );
    // Timer unused
    options.c_cc[VTIME]=0;
    // At least on character before satisfy reading
    options.c_cc[VMIN]=0;
    // Activate the settings
    tcsetattr(fd, TCSANOW, &options);
    // Success
    return (1);
#endif

}

bool serialib::isDeviceOpen()
{
#if defined (_WIN32) || defined( _WIN64)
    return hSerial != INVALID_HANDLE_VALUE;
#endif
#if defined (__linux__) || defined(__APPLE__)
    return fd >= 0;
#endif
}

void serialib::closeDevice()
{
#if defined (_WIN32) || defined( _WIN64)
    CloseHandle(hSerial);
    hSerial = INVALID_HANDLE_VALUE;
#endif
#if defined (__linux__) || defined(__APPLE__)
    close (fd);
    fd = -1;
#endif
}




//_____________________________________________
// ::: Read/Write operation on characters :::



char serialib::writeChar(const char Byte)
{
#if defined (_WIN32) || defined( _WIN64)
    // Number of bytes written
    DWORD dwBytesWritten;
    // Write the char to the serial device
    // Return -1 if an error occured
    if(!WriteFile(hSerial,&Byte,1,&dwBytesWritten,NULL)) return -1;
    // Write operation successfull
    return 1;
#endif
#if defined (__linux__) || defined(__APPLE__)
    // Write the char
    if (write(fd,&Byte,1)!=1) return -1;

    // Write operation successfull
    return 1;
#endif
}



//_________________________________________
// ::: Read/Write operation on strings :::


char serialib::writeString(const char *receivedString)
{
#if defined (_WIN32) || defined( _WIN64)
    // Number of bytes written
    DWORD dwBytesWritten;
    // Write the string
    if(!WriteFile(hSerial,receivedString,strlen(receivedString),&dwBytesWritten,NULL))
        // Error while writing, return -1
        return -1;
    // Write operation successfull
    return 1;
#endif
#if defined (__linux__) || defined(__APPLE__)
    // Lenght of the string
    int Lenght=strlen(receivedString);
    // Write the string
    if (write(fd,receivedString,Lenght)!=Lenght) return -1;
    // Write operation successfull
    return 1;
#endif
}

// _____________________________________
// ::: Read/Write operation on bytes :::



char serialib::writeBytes(const void *Buffer, const unsigned int NbBytes)
{
#if defined (_WIN32) || defined( _WIN64)
    // Number of bytes written
    DWORD dwBytesWritten;
    // Write data
    if(!WriteFile(hSerial, Buffer, NbBytes, &dwBytesWritten, NULL))
        // Error while writing, return -1
        return -1;
    // Write operation successfull
    return 1;
#endif
#if defined (__linux__) || defined(__APPLE__)
    // Write data
    if (write (fd,Buffer,NbBytes)!=(ssize_t)NbBytes) return -1;
    // Write operation successfull
    return 1;
#endif
}



char serialib::readChar(char *pByte,unsigned int timeOut_ms)
{
#if defined (_WIN32) || defined(_WIN64)
    // Number of bytes read
    DWORD dwBytesRead = 0;

    // Set the TimeOut
    timeouts.ReadTotalTimeoutConstant=timeOut_ms;

    // Write the parameters, return -1 if an error occured
    if(!SetCommTimeouts(hSerial, &timeouts)) return -1;

    // Read the byte, return -2 if an error occured
    if(!ReadFile(hSerial,pByte, 1, &dwBytesRead, NULL)) return -2;

    // Return 0 if the timeout is reached
    if (dwBytesRead==0) return 0;

    // The byte is read
    return 1;
#endif
#if defined (__linux__) || defined(__APPLE__)
    // Timer used for timeout
    timeOut         timer;
    // Initialise the timer
    timer.initTimer();
    // While Timeout is not reached
    while (timer.elapsedTime_ms()<timeOut_ms || timeOut_ms==0)
    {
        // Try to read a byte on the device
        switch (read(fd,pByte,1)) {
        case 1  : return 1; // Read successfull
        case -1 : return -2; // Error while reading
        }
    }
    return 0;
#endif
}



int serialib::readStringNoTimeOut(char *receivedString,char finalChar,unsigned int maxNbBytes)
{
    // Number of characters read
    unsigned int    NbBytes=0;
    // Returned value from Read
    char            charRead;

    // While the buffer is not full
    while (NbBytes<maxNbBytes)
    {
        // Read a character with the restant time
        charRead=readChar(&receivedString[NbBytes]);

        // Check a character has been read
        if (charRead==1)
        {
            // Check if this is the final char
            if (receivedString[NbBytes]==finalChar)
            {
                // This is the final char, add zero (end of string)
                receivedString  [++NbBytes]=0;
                // Return the number of bytes read
                return NbBytes;
            }

            // The character is not the final char, increase the number of bytes read
            NbBytes++;
        }

        // An error occured while reading, return the error number
        if (charRead<0) return charRead;
    }
    // Buffer is full : return -3
    return -3;
}


int serialib::readString(char *receivedString,char finalChar,unsigned int maxNbBytes,unsigned int
 timeOut_ms)
{
    // Check if timeout is requested
    if (timeOut_ms==0) return readStringNoTimeOut(receivedString,finalChar,maxNbBytes);

    // Number of bytes read
    unsigned int    nbBytes=0;
    // Character read on serial device
    char            charRead;
    // Timer used for timeout
    timeOut         timer;
    long int        timeOutParam;

    // Initialize the timer (for timeout)
    timer.initTimer();

    // While the buffer is not full
    while (nbBytes<maxNbBytes)
    {
        // Compute the TimeOut for the next call of ReadChar
        timeOutParam = timeOut_ms-timer.elapsedTime_ms();

        // If there is time remaining
        if (timeOutParam>0)
        {
            // Wait for a byte on the serial link with the remaining time as timeout
            charRead=readChar(&receivedString[nbBytes],timeOutParam);

            // If a byte has been received
            if (charRead==1)
            {
                // Check if the character received is the final one
                if (receivedString[nbBytes]==finalChar)
                {
                    // Final character: add the end character 0
                    receivedString  [++nbBytes]=0;
                    // Return the number of bytes read
                    return nbBytes;
                }
                // This is not the final character, just increase the number of bytes read
                nbBytes++;
            }
            // Check if an error occured during reading char
            // If an error occurend, return the error number
            if (charRead<0) return charRead;
        }
        // Check if timeout is reached
        if (timer.elapsedTime_ms()>timeOut_ms)
        {
            // Add the end caracter
            receivedString[nbBytes]=0;
            // Return 0 (timeout reached)
            return 0;
        }
    }

    // Buffer is full : return -3
    return -3;
}

```

```
int serialib::readBytes (void *buffer,unsigned int maxNbBytes,unsigned int timeOut_ms, unsigned int
 sleepDuration_us)
{
#if defined (_WIN32) || defined(_WIN64)
    // Avoid warning while compiling
    UNUSED(sleepDuration_us);

    // Number of bytes read
    DWORD dwBytesRead = 0;

    // Set the TimeOut
    timeouts.ReadTotalTimeoutConstant=(DWORD)timeOut_ms;

    // Write the parameters and return -1 if an error occrured
    if(!SetCommTimeouts(hSerial, &timeouts)) return -1;


    // Read the bytes from the serial device, return -2 if an error occured
    if(!ReadFile(hSerial,buffer,(DWORD)maxNbBytes,&dwBytesRead, NULL))  return -2;

    // Return the byte read
    return dwBytesRead;
#endif
#if defined (__linux__) || defined(__APPLE__)
    // Timer used for timeout
    timeOut          timer;
    // Initialise the timer
    timer.initTimer();
    unsigned int     NbByteRead=0;
    // While Timeout is not reached
    while (timer.elapsedTime_ms()<timeOut_ms || timeOut_ms==0)
    {
        // Compute the position of the current byte
        unsigned char* Ptr=(unsigned char*)buffer+NbByteRead;
        // Try to read a byte on the device
        int Ret=read(fd,(void*)Ptr,maxNbBytes-NbByteRead);
        // Error while reading
        if (Ret==-1) return -2;

        // One or several byte(s) has been read on the device
        if (Ret>0)
        {
            // Increase the number of read bytes
            NbByteRead+=Ret;
            // Success : bytes has been read
            if (NbByteRead>=maxNbBytes)
                return NbByteRead;
        }
        // Suspend the loop to avoid charging the CPU
        usleep (sleepDuration_us);
    }
    // Timeout reached, return the number of bytes read
    return NbByteRead;
#endif
}




// _________________________
// ::: Special operation :::



char serialib::flushReceiver()
{
#if defined (_WIN32) || defined(_WIN64)
    // Purge receiver
    return PurgeComm (hSerial, PURGE_RXCLEAR);
#endif
#if defined (__linux__) || defined(__APPLE__)
    // Purge receiver
    tcflush(fd,TCIFLUSH);
    return true;
#endif
}



int serialib::available()
{
#if defined (_WIN32) || defined(_WIN64)
    // Device errors
    DWORD commErrors;
    // Device status
    COMSTAT commStatus;
    // Read status
    ClearCommError(hSerial, &commErrors, &commStatus);
    // Return the number of pending bytes
    return commStatus.cbInQue;
#endif
#if defined (__linux__) || defined(__APPLE__)
    int nBytes=0;
    // Return number of pending bytes in the receiver
    ioctl(fd, FIONREAD, &nBytes);
    return nBytes;
#endif

}



// __________________
// ::: I/O Access :::

bool serialib::DTR(bool status)
{
    if (status)
        // Set DTR
        return this->setDTR();
    else
        // Unset DTR
        return this->clearDTR();
}


bool serialib::setDTR()
{
#if defined (_WIN32) || defined(_WIN64)
    // Set DTR
    currentStateDTR=true;
    return EscapeCommFunction(hSerial,SETDTR);
#endif
#if defined (__linux__) || defined(__APPLE__)
    // Set DTR
    int status_DTR=0;
    ioctl(fd, TIOCMGET, &status_DTR);
    status_DTR |= TIOCM_DTR;
    ioctl(fd, TIOCMSET, &status_DTR);
    return true;
#endif
}

bool serialib::clearDTR()
{
#if defined (_WIN32) || defined(_WIN64)
    // Clear DTR
    currentStateDTR=true;
    return EscapeCommFunction(hSerial,CLRDTR);
#endif
#if defined (__linux__) || defined(__APPLE__)
    // Clear DTR
    int status_DTR=0;
    ioctl(fd, TIOCMGET, &status_DTR);
    status_DTR &= ~TIOCM_DTR;
    ioctl(fd, TIOCMSET, &status_DTR);
    return true;
#endif
}



bool serialib::RTS(bool status)
{
    if (status)
        // Set RTS
        return this->setRTS();
    else
        // Unset RTS
        return this->clearRTS();
}


bool serialib::setRTS()
{
#if defined (_WIN32) || defined(_WIN64)
    // Set RTS
    currentStateRTS=false;
    return EscapeCommFunction(hSerial,SETRTS);
#endif
#if defined (__linux__) || defined(__APPLE__)
    // Set RTS
    int status_RTS=0;
    ioctl(fd, TIOCMGET, &status_RTS);
    status_RTS |= TIOCM_RTS;
    ioctl(fd, TIOCMSET, &status_RTS);
    return true;
#endif
}



bool serialib::clearRTS()
{
#if defined (_WIN32) || defined(_WIN64)
    // Clear RTS
    currentStateRTS=false;
    return EscapeCommFunction(hSerial,CLRRTS);
#endif
#if defined (__linux__) || defined(__APPLE__)
    // Clear RTS
    int status_RTS=0;
    ioctl(fd, TIOCMGET, &status_RTS);
    status_RTS &= ~TIOCM_RTS;
    ioctl(fd, TIOCMSET, &status_RTS);
    return true;
#endif
}




bool serialib::isCTS()
{
#if defined (_WIN32) || defined(_WIN64)
    DWORD modemStat;
    GetCommModemStatus(hSerial, &modemStat);
    return modemStat & MS_CTS_ON;
#endif
#if defined (__linux__) || defined(__APPLE__)
    int status=0;
    //Get the current status of the CTS bit
    ioctl(fd, TIOCMGET, &status);
    return status & TIOCM_CTS;
#endif
}



bool serialib::isDSR()
{
#if defined (_WIN32) || defined(_WIN64)
    DWORD modemStat;
    GetCommModemStatus(hSerial, &modemStat);
    return modemStat & MS_DSR_ON;
#endif
#if defined (__linux__) || defined(__APPLE__)
    int status=0;
    //Get the current status of the DSR bit
    ioctl(fd, TIOCMGET, &status);
    return status & TIOCM_DSR;
#endif
}






bool serialib::isDCD()
{
#if defined (_WIN32) || defined(_WIN64)
    DWORD modemStat;
    GetCommModemStatus(hSerial, &modemStat);
    return modemStat & MS_RLSD_ON;
#endif
#if defined (__linux__) || defined(__APPLE__)
    int status=0;
    //Get the current status of the DCD bit
    ioctl(fd, TIOCMGET, &status);
    return status & TIOCM_CAR;
#endif
}


bool serialib::isRI()
{
#if defined (_WIN32) || defined(_WIN64)
    DWORD modemStat;
    GetCommModemStatus(hSerial, &modemStat);
    return modemStat & MS_RING_ON;
#endif
#if defined (__linux__) || defined(__APPLE__)
    int status=0;
    //Get the current status of the RING bit
    ioctl(fd, TIOCMGET, &status);
    return status & TIOCM_RNG;
#endif
}


bool serialib::isDTR()
{
#if defined (_WIN32) || defined( _WIN64)
    return currentStateDTR;
#endif
#if defined (__linux__) || defined(__APPLE__)
    int status=0;
    //Get the current status of the DTR bit
    ioctl(fd, TIOCMGET, &status);
    return status & TIOCM_DTR  ;
#endif
}



bool serialib::isRTS()
{
#if defined (_WIN32) || defined(_WIN64)
    return currentStateRTS;
#endif
#if defined (__linux__) || defined(__APPLE__)
    int status=0;
    //Get the current status of the CTS bit
    ioctl(fd, TIOCMGET, &status);
    return status & TIOCM_RTS;
#endif
}






// ******************************************
//  Class timeOut
// ******************************************


// Constructor
timeOut::timeOut()
{}


//Initialize the timer
void timeOut::initTimer()
{
#if defined (NO_POSIX_TIME)
    LARGE_INTEGER tmp;
    QueryPerformanceFrequency(&tmp);
    counterFrequency = tmp.QuadPart;
    // Used to store the previous time (for computing timeout)
    QueryPerformanceCounter(&tmp);
    previousTime = tmp.QuadPart;
#else
    gettimeofday(&previousTime, NULL);
#endif
}

//Return the elapsed time since initialization
unsigned long int timeOut::elapsedTime_ms()
{
#if defined (NO_POSIX_TIME)
    // Current time
    LARGE_INTEGER CurrentTime;
    // Number of ticks since last call
    int sec;

    // Get current time
    QueryPerformanceCounter(&CurrentTime);

    // Compute the number of ticks elapsed since last call
    sec=CurrentTime.QuadPart-previousTime;

    // Return the elapsed time in milliseconds
    return sec/(counterFrequency/1000);
#else
```

```
    // Current time
    struct timeval CurrentTime;
    // Number of seconds and microseconds since last call
    int sec,usec;

    // Get current time
    gettimeofday(&CurrentTime, NULL);

    // Compute the number of seconds and microseconds elapsed since last call
    sec=CurrentTime.tv_sec-previousTime.tv_sec;
    usec=CurrentTime.tv_usec-previousTime.tv_usec;

    // If the previous usec is higher than the current one
    if (usec<0)
    {
        // Recompute the microseonds and substract one second
        usec=1000000-previousTime.tv_usec+CurrentTime.tv_usec;
        sec--;
    }

    // Return the elapsed time in milliseconds
    return sec*1000+usec/1000;
#endif
}
```

4.7 serialib.h File Reference

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)

Enumerations

- 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 }

4.7.1 Detailed Description

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

Author

Philippe Lucidarme (University of Angers)

Version

2.0

Date

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

Definition in file serialib.h.

4.7.2 Macro Definition Documentation

4.7.2.1 UNUSED

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

To avoid unused parameters

Definition at line 56 of file serialib.h.

4.7.3 Enumeration Type Documentation

4.7.3.1 SerialDataBits

```
enum SerialDataBits
```

number of serial data bits

Enumerator

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

Definition at line 61 of file serialib.h.

4.7.3.2 SerialParity

`enum SerialParity`

type of serial parity bits

Enumerator

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

Definition at line 81 of file serialib.h.

4.7.3.3 SerialStopBits

`enum SerialStopBits`

number of serial stop bits

Enumerator

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

Definition at line 72 of file serialib.h.

4.8 serialib.h

Go to the documentation of this file.

```

#ifndef SERIALIB_H
#define SERIALIB_H

#if defined(__CYGWIN__)
    // This is Cygwin special case
    #include <sys/time.h>
#endif

// Include for windows
#if defined (_WIN32) || defined (_WIN64)
#if defined(__GNUC__)
    // This is MinGW special case
    #include <sys/time.h>
#else
    // sys/time.h does not exist on "actual" Windows
    #define NO_POSIX_TIME
#endif
    // Accessing to the serial port under Windows
    #include <windows.h>
#endif

// Include for Linux
#if defined (__linux__) || defined(__APPLE__)
    #include <stdlib.h>
    #include <sys/types.h>
    #include <sys/shm.h>
    #include <termios.h>
    #include <string.h>
    #include <iostream>
    #include <sys/time.h>
    // File control definitions
    #include <fcntl.h>
    #include <unistd.h>
    #include <sys/ioctl.h>
#endif

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

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
};

class serialib
{
public:

    //_____________________________________
    // ::: Constructors and destructors :::



    // Constructor of the class
    serialib    ();

    // Destructor
    ~serialib   ();



    //_________________________________________
    // ::: Configuration and initialization :::


```

```
    // Open a device
    char openDevice(const char *Device, const unsigned int Bauds,
                    SerialDataBits Databits = SERIAL_DATABITS_8,
                    SerialParity Parity = SERIAL_PARITY_NONE,
                    SerialStopBits Stopbits = SERIAL_STOPBITS_1);

    // Check device opening state
    bool isDeviceOpen();

    // Close the current device
    void    closeDevice();




    //___________________________________________
    // ::: Read/Write operation on characters :::


    // Write a char
    char    writeChar   (char);

    // Read a char (with timeout)
    char    readChar    (char *pByte,const unsigned int timeOut_ms=0);




    //________________________________________
    // ::: Read/Write operation on strings :::


    // Write a string
    char    writeString (const char *String);

    // Read a string (with timeout)
    int     readString  (   char *receivedString,
                            char finalChar,
                            unsigned int maxNbBytes,
                            const unsigned int timeOut_ms=0);



    // _____________________________________
    // ::: Read/Write operation on bytes :::


    // Write an array of bytes
    char    writeBytes  (const void *Buffer, const unsigned int NbBytes);

    // Read an array of byte (with timeout)
    int     readBytes   (void *buffer,unsigned int maxNbBytes,const unsigned int timeOut_ms=0, unsigned int sleepDuration_us=100);




    // _________________________
    // ::: Special operation :::


    // Empty the received buffer
    char    flushReceiver();

    // Return the number of bytes in the received buffer
    int     available();




    // _________________________
    // ::: Access to IO bits :::


    // Set CTR status (Data Terminal Ready, pin 4)
    bool    DTR(bool status);
    bool    setDTR();
    bool    clearDTR();

    // Set RTS status (Request To Send, pin 7)
    bool    RTS(bool status);
    bool    setRTS();
    bool    clearRTS();

    // Get RI status (Ring Indicator, pin 9)
    bool    isRI();

    // Get DCD status (Data Carrier Detect, pin 1)
    bool    isDCD();

    // Get CTS status (Clear To Send, pin 8)
    bool    isCTS();

    // Get DSR status (Data Set Ready, pin 9)
    bool    isDSR();

    // Get RTS status (Request To Send, pin 7)
    bool    isRTS();

    // Get CTR status (Data Terminal Ready, pin 4)
    bool    isDTR();


private:
    // Read a string (no timeout)
    int             readStringNoTimeOut  (char *String,char FinalChar,unsigned int MaxNbBytes);

    // Current DTR and RTS state (can't be read on WIndows)
    bool            currentStateRTS;
    bool            currentStateDTR;





#if defined (_WIN32) || defined( _WIN64)
    // Handle on serial device
    HANDLE          hSerial;
    // For setting serial port timeouts
    COMMTIMEOUTS    timeouts;
#endif
#if defined (__linux__) || defined(__APPLE__)
    int             fd;
#endif

};



// Class timeOut
class timeOut
{
public:

    // Constructor
    timeOut();

    // Init the timer
    void                initTimer();

    // Return the elapsed time since initialization
    unsigned long int   elapsedTime_ms();

private:
#if defined (NO_POSIX_TIME)
    // Used to store the previous time (for computing timeout)
    LONGLONG       counterFrequency;
    LONGLONG       previousTime;
#else
    // Used to store the previous time (for computing timeout)
    struct timeval      previousTime;
#endif
};

#endif // serialib_H
```

4.9 xbee.cpp File Reference

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 "xbee.h"
```

Classes

- struct Trame

  Structure permettant de définir une trame de message reçue et envoyée.

Variables

- serialib serial

4.9.1 Detailed Description

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.

Author

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

Version

1.0

Date

03/02/2022

Definition in file xbee.cpp.

4.9.2 Variable Documentation

4.9.2.1 serial

```
serialib serial
```

Definition at line 12 of file xbee.cpp.

4.10 xbee.cpp

Go to the documentation of this file.

```

#include "xbee.h"

using namespace std;

serialib serial;

struct Trame{
    int id_exp;
    int id_dest;
    int code_fct;
    int id_trame;
    int size;
    vector<char> data;
};

//________________________________________
// ::: Constructeurs et destructeurs :::

xbee::xbee(){ }

xbee::~xbee(){ }


//____________________________________________
// ::: Configuration and initialisation :::

int xbee::openSerialConnection(){
    serial.flushReceiver();
    char errorOpening = serial.openDevice(SERIAL_PORT, BAUDRATE, DATABITS, PARITY, STOPBITS);

    return (int) errorOpening;
}

void xbee::closeSerialConnection(){
    serial.flushReceiver();
    serial.closeDevice();
}

//____________________________________________
// ::: Configuration en mode AT :::

int xbee::checkATConfig(){
    if(!enterATMode())
    return AT_ERROR_ENTER;

    if(!sendATCommand(AT_GET_API, AT_VALUE_API, AT_MODE_SET))
    return AT_ERROR_API;

    if(!sendATCommand(AT_GET_BAUDRATE, AT_VALUE_BAUDRATE, AT_MODE_SET))
    return AT_ERROR_BAUDRATE;

    if(!sendATCommand(AT_GET_AES, AT_VALUE_AES, AT_MODE_SET))
    return AT_ERROR_AES;

    if(!sendATCommand(AT_GET_AES_KEY, AT_VALUE_AES_KEY, AT_MODE_SET))
    return AT_ERROR_AES_KEY;

    if(!sendATCommand(AT_GET_CHANEL, AT_VALUE_CHANEL, AT_MODE_SET))
    return AT_ERROR_CHANEL;

    if(!sendATCommand(AT_GET_PAN_ID, AT_VALUE_PAN_ID, AT_MODE_SET))
    return AT_ERROR_PAN_ID;

    if(!sendATCommand(AT_GET_COORDINATOR, AT_VALUE_COORDINATOR, AT_MODE_SET))
    return AT_ERROR_COORDINATOR;

    if(!sendATCommand(AT_GET_PARITY, AT_VALUE_PARITY, AT_MODE_SET))
    return AT_ERROR_PARITY;

    if(!sendATCommand(AT_GET_16BIT_SOURCE_ADDR, AT_VALUE_16BIT_SOURCE_ADDR, AT_MODE_SET))
    return AT_ERROR_16BIT_SOURCE_ADDR;

    if(!sendATCommand(AT_GET_LOW_DEST_ADDR, AT_VALUE_LOW_DEST_ADDR, AT_MODE_SET))
    return AT_ERROR_LOW_DEST_ADDR;

    if(!writeATConfig())
    return AT_ERROR_WRITE_CONFIG;

    if(!exitATMode())
    return AT_ERROR_EXIT;

    return AT_ERROR_SUCCESS;
}

void xbee::delay(unsigned int time){ usleep(time*1000000); }


bool xbee::readATResponse(const char *value){
    char *reponse(0);
    unsigned int timeout = 100;
    reponse = new char;
    delay(1);
    string rep = "";
    int i = 0;
    while(serial.available() > 0){
        i++;
        serial.readChar(reponse, timeout);
        rep += *reponse;
    }
    delete reponse;
    reponse = 0;

    if(rep == value)
        return true;
    else
        return false;
}

bool xbee::enterATMode(){
    serial.writeString(AT_ENTER);
    //cout « "* Entrée en mode AT..." « endl;
    delay(2);
    serial.writeString(AT_END_LINE);
    return readATResponse(AT_SUCCESS_VALUE);
}

bool xbee::exitATMode(){
    serial.writeString(AT_EXIT);
    serial.writeString(AT_END_LINE);
    //cout « "* Sortie du mode AT..." « endl;
    return readATResponse(AT_SUCCESS_VALUE);
}

bool xbee::writeATConfig(){
    serial.writeString(AT_WRITE_CONFIG);
    serial.writeString(AT_END_LINE);
    //cout « "* Ecriture de la configuration AT..." « endl;
    return readATResponse(AT_SUCCESS_VALUE);
}

bool xbee::sendATCommand(const char *command, const char *value, unsigned int mode){
    serial.writeString(command);
    serial.writeString(value);
    serial.writeString(AT_END_LINE);
    if(mode == AT_MODE_GET){
        //cout « "* Envoi de la commande " « command « "...\n";
        return readATResponse(value);
    }else{
        //cout « "* Envoi de la commande " « command « "=" « value « "...\n";
        return readATResponse(AT_SUCCESS_VALUE);
    }
}

//____________________________________________________________
// ::: Envoi/Réception/Traitement des trames de messages :::

int xbee::crc16(vector<char> trame){
    int crc = 0xFFFF, count = 0;
    unsigned char octet_a_traiter;
    const int POLYNOME = 0xA001;

    octet_a_traiter = trame[0];

    do{
        crc ^= octet_a_traiter;
        for(int i = 0; i < 8; i++){

            if((crc%2)!=0)
            crc = (crc » 1) ^ POLYNOME;

            else
                crc = (crc » 1);

        }
        count++;
        octet_a_traiter = trame[count];

    }while(count < trame.size());
    return crc;
}

void xbee::sendTrame(char ad_dest, char code_fct, char data[]){
    vector<char> trame;
    string convert_data = data;
    uint8_t taille_message = (uint8_t) code_fct + (convert_data.size()) + 0x05;

    uint8_t high = (taille_message » 8) & 0xFF;
    uint8_t low = taille_message & 0xFF;

    //cout « taille_message « endl;
    //cout « (int) high « endl;
    //cout « (int) low « endl;

    //char taille_message_h = (char) high;
    //char taille_message_l = (char) low;

    trame.push_back(START_SEQ);

    trame.push_back(CURRENT_ROBOT);
    trame.push_back(ad_dest);
    trame.push_back(++ID_TRAME);
    trame.push_back((char)taille_message);
    //trame.push_back(taille_message_l);

    trame.push_back(code_fct);

    for(int i=0; i < convert_data.size(); i++)
        trame.push_back(convert_data[i]);

    int crc = crc16(trame);

    trame.push_back((char) crc);

    trame.push_back(END_SEQ);

    for(int i=0; i < trame.size(); i++)
        cout « hex « showbase « setw(4) « static_cast<int>(trame[i]);

    cout « endl;

    char* message = reinterpret_cast<char*>(trame.data());;
    //serial.writeString(message);

}
```

4.11 xbee.h File Reference

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 <string>
#include <vector>
#include <iomanip>
#include <iostream>
```

Classes

- class xbee

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

4.11.1 Detailed Description

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.

Author

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

Version

1.0

Date

03/02/2022

Definition in file xbee.h.

4.12 xbee.h

Go to the documentation of this file.

```
#ifndef XBEE_H
#define XBEE_H

#include "define.h"
#include "serialib.h"
#include <string>
#include <vector>
#include <iomanip>
#include <iostream>

class xbee{

public:

    // Constructeur de la classe
    xbee();

    // Desctructeur de la classe
    ~xbee();

    // Ouverture de la connexion série
    int openSerialConnection();

    // Fermeture de la connexion série
    void closeSerialConnection();

    // Entrée dans le mode de configuration AT
    bool enterATMode();

    // Sortie du mode de configuration AT
    bool exitATMode();

    // Vérification et correction de la configuration AT du module
    int checkATConfig();

    // Lecture de la réponse du module à une commande AT
    bool readATResponse(const char *value = AT_EMPTY_VALUE);

    // Envoi d'une commande AT
    bool sendATCommand(const char *command, const char *value, unsigned int mode);

    // Ecriture de la configuration AT dans la mémoire flash du module
    bool writeATConfig();

    // Création et envoi de la trame de message structurée
    void sendTrame(char ad_dest, char code_fct, char data[]);

private:

    // Calcul du CRC16 Modbus de la trame
    int crc16(std::vector<char> trame);

    // Retard de temporisation dans l’exécution du code
    void delay(unsigned int time);
};

#endif
```

Index