

Robotech 2021 - XBee

Généré par Doxygen 1.9.3

1 Index des classes	1
1.1 Liste des classes	1
2 Index des fichiers	3
2.1 Liste des fichiers	3
3 Documentation des classes	5
3.1 Référence de la classe serialib	5
3.1.1 Description détaillée	6
3.1.2 Documentation des constructeurs et destructeur	6
3.1.2.1 serialib()	6
3.1.2.2 ~serialib()	6
3.1.3 Documentation des fonctions membres	7
3.1.3.1 available()	7
3.1.3.2 clearDTR()	7
3.1.3.3 clearRTS()	7
3.1.3.4 closeDevice()	7
3.1.3.5 DTR()	7
3.1.3.6 flushReceiver()	8
3.1.3.7 isCTS()	8
3.1.3.8 isDCD()	8
3.1.3.9 isDeviceOpen()	9
3.1.3.10 isDSR()	9
3.1.3.11 isDTR()	9
3.1.3.12 isRI()	9
3.1.3.13 isRTS()	10
3.1.3.14 openDevice()	10
3.1.3.15 readBytes()	11
3.1.3.16 readChar()	12
3.1.3.17 readString()	12
3.1.3.18 RTS()	13
3.1.3.19 setDTR()	13
3.1.3.20 setRTS()	14
3.1.3.21 writeBytes()	14
3.1.3.22 writeChar()	14
3.1.3.23 writeString()	15
3.2 Référence de la classe timeOut	15
3.2.1 Description détaillée	15
3.2.2 Documentation des constructeurs et destructeur	16
3.2.2.1 timeOut()	16
3.2.3 Documentation des fonctions membres	16
3.2.3.1 elapsedTime_ms()	16
3.2.3.2 initTimer()	16

3.3 Référence de la structure Trame	16
3.3.1 Description détaillée	17
3.3.2 Documentation des données membres	17
3.3.2.1 code_fct	17
3.3.2.2 data	17
3.3.2.3 id_dest	17
3.3.2.4 id_exp	17
3.3.2.5 id_trame	18
3.3.2.6 size	18
3.4 Référence de la classe xbee	18
3.4.1 Description détaillée	19
3.4.2 Documentation des constructeurs et destructeur	19
3.4.2.1 xbee()	19
3.4.2.2 ~xbee()	19
3.4.3 Documentation des fonctions membres	19
3.4.3.1 checkATConfig()	19
3.4.3.2 closeSerialConnection()	20
3.4.3.3 enterATMode()	20
3.4.3.4 exitATMode()	20
3.4.3.5 openSerialConnection()	20
3.4.3.6 readATResponse()	21
3.4.3.7 sendATCommand()	21
3.4.3.8 sendTrame()	21
3.4.3.9 writeATConfig()	22
4 Documentation des fichiers	23
4.1 Référence du fichier define.h	23
4.1.1 Documentation des macros	24
4.1.1.1 AT_EMPTY_VALUE	24
4.1.1.2 AT_END_LINE	24
4.1.1.3 AT_ENTER	24
4.1.1.4 AT_ERROR_16BIT_SOURCE_ADDR	24
4.1.1.5 AT_ERROR_AES	25
4.1.1.6 AT_ERROR_AES_KEY	25
4.1.1.7 AT_ERROR_API	25
4.1.1.8 AT_ERROR_BAUDRATE	25
4.1.1.9 AT_ERROR_CHANEL	25
4.1.1.10 AT_ERROR_COORDINATOR	25
4.1.1.11 AT_ERROR_ENTER	26
4.1.1.12 AT_ERROR_EXIT	26
4.1.1.13 AT_ERROR_LOW_DEST_ADDR	26
4.1.1.14 AT_ERROR_PAN_ID	26

4.1.1.15 AT_ERROR_PARITY	26
4.1.1.16 AT_ERROR_SUCCESS	26
4.1.1.17 AT_ERROR_VALUE	27
4.1.1.18 AT_ERROR_WRITE_CONFIG	27
4.1.1.19 AT_EXIT	27
4.1.1.20 AT_GET_16BIT_SOURCE_ADDR	27
4.1.1.21 AT_GET_AES	27
4.1.1.22 AT_GET_AES_KEY	27
4.1.1.23 AT_GET_API	28
4.1.1.24 AT_GET_BAUDRATE	28
4.1.1.25 AT_GET_CHANEL	28
4.1.1.26 AT_GET_COORDINATOR	28
4.1.1.27 AT_GET_LOW_DEST_ADDR	28
4.1.1.28 AT_GET_PAN_ID	28
4.1.1.29 AT_GET_PARITY	29
4.1.1.30 AT_MODE_GET	29
4.1.1.31 AT_MODE_SET	29
4.1.1.32 AT_SUCCESS_VALUE	29
4.1.1.33 AT_VALUE_16BIT_SOURCE_ADDR	29
4.1.1.34 AT_VALUE_AES	29
4.1.1.35 AT_VALUE_AES_KEY	30
4.1.1.36 AT_VALUE_API	30
4.1.1.37 AT_VALUE_BAUDRATE	30
4.1.1.38 AT_VALUE_CHANEL	30
4.1.1.39 AT_VALUE_COORDINATOR	30
4.1.1.40 AT_VALUE_LOW_DEST_ADDR	30
4.1.1.41 AT_VALUE_PAN_ID	31
4.1.1.42 AT_VALUE_PARITY	31
4.1.1.43 AT_WRITE_CONFIG	31
4.1.1.44 BAUDRATE	31
4.1.1.45 BROADCAST	31
4.1.1.46 CURRENT_ROBOT	31
4.1.1.47 DATABITS	32
4.1.1.48 END_SEQ	32
4.1.1.49 ERROR_SUCCESS	32
4.1.1.50 PARITY	32
4.1.1.51 ROBOT_01	32
4.1.1.52 ROBOT_02	32
4.1.1.53 SERIAL_PORT	33
4.1.1.54 START_SEQ	33
4.1.1.55 STOPBITS	33
4.1.1.56 TEST_ALIVE	33

4.2 define.h	33
4.3 Référence du fichier main.cpp	34
4.3.1 Documentation des fonctions	34
4.3.1.1 main()	35
4.4 main.cpp	35
4.5 Référence du fichier serialib.cpp	35
4.5.1 Description détaillée	35
4.6 serialib.cpp	36
4.7 Référence du fichier serialib.h	45
4.7.1 Description détaillée	46
4.7.2 Documentation des macros	46
4.7.2.1 UNUSED	47
4.7.3 Documentation du type de l'énumération	47
4.7.3.1 SerialDataBits	47
4.7.3.2 SerialParity	47
4.7.3.3 SerialStopBits	48
4.8 serialib.h	48
4.9 Référence du fichier xbee.cpp	51
4.9.1 Description détaillée	51
4.9.2 Documentation des variables	51
4.9.2.1 serial	52
4.10 xbee.cpp	52
4.11 Référence du fichier xbee.h	54
4.11.1 Description détaillée	55
4.12 xbee.h	55
Index	57

Chapitre 1

Index des classes

1.1 Liste des classes

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

serialib	This class is used for communication over a serial device	5
timeOut	This class can manage a timer which is used as a timeout	15
Trame	Structure permettant de définir une trame de message reçue et envoyée	16
xbee	Cette classe est utilisée pour la communication entre un module XBee et une RaspberryPi et entre plusieurs modules XBee	18

Chapitre 2

Index des fichiers

2.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

define.h	23
main.cpp	34
serialib.cpp	Source file of the class serialib. This class is used for communication over a serial device . . .	35
serialib.h	Header file of the class serialib. This class is used for communication over a serial device . . .	45
xbec.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	51
xbec.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 . . .	54

Chapitre 3

Documentation des classes

3.1 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.
- [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 Terminal 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 Description détaillée

This class is used for communication over a serial device.

Définition à la ligne 92 du fichier [serialib.h](#).

3.1.2 Documentation des constructeurs et destructeur

3.1.2.1 [serialib\(\)](#)

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

Constructor of the class serialib.

Définition à la ligne 29 du fichier [serialib.cpp](#).

3.1.2.2 [~serialib\(\)](#)

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

Destructor of the class serialib. It close the connection.

Définition à la ligne 47 du fichier [serialib.cpp](#).

3.1.3 Documentation des fonctions membres

3.1.3.1 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 701 du fichier [serialib.cpp](#).

3.1.3.2 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 776 du fichier [serialib.cpp](#).

3.1.3.3 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 846 du fichier [serialib.cpp](#).

3.1.3.4 closeDevice()

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

Close the connection with the current device.

Définition à la ligne 316 du fichier [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.

Paramètres

<i>status</i>	= 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 [736](#) du fichier [serialib.cpp](#).

3.1.3.6 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 [682](#) du fichier [serialib.cpp](#).

3.1.3.7 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 [871](#) du fichier [serialib.cpp](#).

3.1.3.8 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 [919](#) du fichier [serialib.cpp](#).

3.1.3.9 isDeviceOpen()

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

Définition à la ligne 303 du fichier [serialib.cpp](#).

3.1.3.10 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 893 du fichier [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.

Renvoie

Return true if CTS is set otherwise false

Définition à la ligne 962 du fichier [serialib.cpp](#).

3.1.3.12 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 940 du fichier [serialib.cpp](#).

3.1.3.13 isRTS()

```
bool seriallib::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 983 du fichier [serialib.cpp](#).

3.1.3.14 openDevice()

```
char seriallib::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

<i>Device</i>	: Port name (COM1, COM2, ... for Windows) or (/dev/ttyS0, /dev/ttyACM0, /dev/ttyUSB0 ... for linux)
<i>Bauds</i>	: Baud rate of the serial port. <pre> \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 </pre>

Paramètres

<i>Databits</i>	: Number of data bits in one UART transmission. \n Supported values: \n <ul style="list-style-type: none"> - SERIAL_DATABITS_5 (5) - SERIAL_DATABITS_6 (6) - SERIAL_DATABITS_7 (7) - SERIAL_DATABITS_8 (8) - SERIAL_DATABITS_16 (16) (not supported on Unix)
<i>Parity</i>	Parity type \n Supported values: \n <ul style="list-style-type: none"> - 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)
<i>Stopbit</i>	Number of stop bits \n Supported values: <ul style="list-style-type: none"> - SERIAL_STOPBITS_1 (1) - SERIAL_STOPBITS_1_5 (1.5) (not supported on Unix) - SERIAL_STOPBITS_2 (2)

Renvoie

- 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

Définition à la ligne 128 du fichier [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)

Paramètres

<i>buffer</i>	: array of bytes read from the serial device
<i>maxNbBytes</i>	: maximum allowed number of bytes read
<i>timeOut_ms</i>	: delay of timeout before giving up the reading
<i>sleepDuration_us</i>	: 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 [614](#) du fichier [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.

Paramètres

<i>pByte</i>	: data read on the serial device
<i>timeout_ms</i>	: 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 [440](#) du fichier [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)

Paramètres

<i>receivedString</i>	: string read on the serial device
<i>finalChar</i>	: final char of the string
<i>maxNbBytes</i>	: maximum allowed number of bytes read
<i>timeout_ms</i>	: 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 540 du fichier [serialib.cpp](#).

3.1.3.18 RTS()

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

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

Paramètres

<i>status</i>	= true set DTR status = false unset DTR
---------------	---

Renvoie

- false if the function fails
- true if the function succeeds

Définition à la ligne 804 du fichier [serialib.cpp](#).

3.1.3.19 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 753 du fichier [serialib.cpp](#).

3.1.3.20 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 [821](#) du fichier [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.

Paramètres

<i>Buffer</i>	: array of bytes to send on the port
<i>NbBytes</i>	: number of byte to send

Renvoie

1 success
-1 error while writting data

Définition à la ligne [408](#) du fichier [serialib.cpp](#).

3.1.3.22 writeChar()

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

Write a char on the current serial port.

Paramètres

<i>Byte</i>	: char to send on the port (must be terminated by '\0')
-------------	---

Renvoie

- 1 success
- 1 error while writting data

Définition à la ligne 342 du fichier [serialib.cpp](#).

3.1.3.23 writeString()

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

Write a string on the current serial port.

Paramètres

<i>receivedString</i>	: string to send on the port (must be terminated by '\0')
-----------------------	---

Renvoie

- 1 success
- 1 error while writting data

Définition à la ligne 374 du fichier [serialib.cpp](#).

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

- [serialib.h](#)
- [serialib.cpp](#)

3.2 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.

3.2.1 Description détaillée

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

Définition à la ligne 245 du fichier [serialib.h](#).

3.2.2 Documentation des constructeurs et destructeur

3.2.2.1 `timeOut()`

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

Constructor of the class [timeOut](#).

Définition à la ligne [1010](#) du fichier [serialib.cpp](#).

3.2.3 Documentation des fonctions membres

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`.

Renvoie

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

Définition à la ligne [1038](#) du fichier [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`.

Définition à la ligne [1018](#) du fichier [serialib.cpp](#).

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

- [serialib.h](#)
- [serialib.cpp](#)

3.3 Référence de la structure Trame

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

Attributs publics

- int [id_exp](#)
- int [id_dest](#)
- int [code_fct](#)
- int [id_trame](#)
- int [size](#)
- vector< char > [data](#)

3.3.1 Description détaillée

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

Définition à la ligne [18](#) du fichier [xbee.cpp](#).

3.3.2 Documentation des données membres

3.3.2.1 code_fct

```
int Trame::code_fct
```

Définition à la ligne [21](#) du fichier [xbee.cpp](#).

3.3.2.2 data

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

Définition à la ligne [24](#) du fichier [xbee.cpp](#).

3.3.2.3 id_dest

```
int Trame::id_dest
```

Définition à la ligne [20](#) du fichier [xbee.cpp](#).

3.3.2.4 id_exp

```
int Trame::id_exp
```

Définition à la ligne [19](#) du fichier [xbee.cpp](#).

3.3.2.5 id_trame

```
int Trame::id_trame
```

Définition à la ligne 22 du fichier [xbee.cpp](#).

3.3.2.6 size

```
int Trame::size
```

Définition à la ligne 23 du fichier [xbee.cpp](#).

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

— [xbee.cpp](#)

3.4 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 <xbee.h>
```

Fonctions membres publiques

- [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 commande 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 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 22 du fichier [xbee.h](#).

3.4.2 Documentation des constructeurs et destructeur

3.4.2.1 xbee()

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

Constructeur de la classe xbee.

Définition à la ligne 33 du fichier [xbee.cpp](#).

3.4.2.2 ~xbee()

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

Destructeur de la classe xbee.

Définition à la ligne 38 du fichier [xbee.cpp](#).

3.4.3 Documentation des fonctions membres

3.4.3.1 checkATConfig()

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

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

Renvoie

- 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'adresse source 16bits
- 11 impossible de sortir du mode AT
- 12 impossible d'écrire les paramètres dans la mémoire flash

Définition à la ligne 91 du fichier [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.

Définition à la ligne 67 du fichier [xbee.cpp](#).

3.4.3.3 enterATMode()

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

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 173 du fichier [xbee.cpp](#).

3.4.3.4 exitATMode()

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

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 186 du fichier [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.

Renvoie

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

Définition à la ligne 57 du fichier [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.

Paramètres

<i>value</i>	: la valeur de réponse attendue pour la commande envoyée
--------------	--

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 147 du fichier [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 commande AT.

Paramètres

<i>command</i>	: le paramètre AT à envoyer au module
<i>value</i>	: la valeur de réponse attendue
<i>mode</i>	: 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 213 du fichier [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.

Paramètres

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

Définition à la ligne 266 du fichier [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.

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 198 du fichier [xbee.cpp](#).

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

- [xbee.h](#)
- [xbee.cpp](#)

Chapitre 4

Documentation des fichiers

4.1 Référence du fichier define.h

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 Documentation des macros

4.1.1.1 AT_EMPTY_VALUE

```
#define AT_EMPTY_VALUE ""
```

Définition à la ligne 55 du fichier [define.h](#).

4.1.1.2 AT_END_LINE

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

Définition à la ligne 30 du fichier [define.h](#).

4.1.1.3 AT_ENTER

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

Définition à la ligne 28 du fichier [define.h](#).

4.1.1.4 AT_ERROR_16BIT_SOURCE_ADDR

```
#define AT_ERROR_16BIT_SOURCE_ADDR -9
```

Définition à la ligne 72 du fichier [define.h](#).

4.1.1.5 AT_ERROR_AES

```
#define AT_ERROR_AES -4
```

Définition à la ligne 66 du fichier [define.h](#).

4.1.1.6 AT_ERROR_AES_KEY

```
#define AT_ERROR_AES_KEY -13
```

Définition à la ligne 67 du fichier [define.h](#).

4.1.1.7 AT_ERROR_API

```
#define AT_ERROR_API -2
```

Définition à la ligne 64 du fichier [define.h](#).

4.1.1.8 AT_ERROR_BAUDRATE

```
#define AT_ERROR_BAUDRATE -3
```

Définition à la ligne 65 du fichier [define.h](#).

4.1.1.9 AT_ERROR_CHANEL

```
#define AT_ERROR_CHANEL -5
```

Définition à la ligne 68 du fichier [define.h](#).

4.1.1.10 AT_ERROR_COORDINATOR

```
#define AT_ERROR_COORDINATOR -7
```

Définition à la ligne 70 du fichier [define.h](#).

4.1.1.11 AT_ERROR_ENTER

```
#define AT_ERROR_ENTER -1
```

Définition à la ligne 63 du fichier [define.h](#).

4.1.1.12 AT_ERROR_EXIT

```
#define AT_ERROR_EXIT -11
```

Définition à la ligne 74 du fichier [define.h](#).

4.1.1.13 AT_ERROR_LOW_DEST_ADDR

```
#define AT_ERROR_LOW_DEST_ADDR -10
```

Définition à la ligne 73 du fichier [define.h](#).

4.1.1.14 AT_ERROR_PAN_ID

```
#define AT_ERROR_PAN_ID -6
```

Définition à la ligne 69 du fichier [define.h](#).

4.1.1.15 AT_ERROR_PARITY

```
#define AT_ERROR_PARITY -8
```

Définition à la ligne 71 du fichier [define.h](#).

4.1.1.16 AT_ERROR_SUCCESS

```
#define AT_ERROR_SUCCESS 0
```

Définition à la ligne 76 du fichier [define.h](#).

4.1.1.17 AT_ERROR_VALUE

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

Définition à la ligne 57 du fichier [define.h](#).

4.1.1.18 AT_ERROR_WRITE_CONFIG

```
#define AT_ERROR_WRITE_CONFIG -12
```

Définition à la ligne 75 du fichier [define.h](#).

4.1.1.19 AT_EXIT

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

Définition à la ligne 29 du fichier [define.h](#).

4.1.1.20 AT_GET_16BIT_SOURCE_ADDR

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

Définition à la ligne 41 du fichier [define.h](#).

4.1.1.21 AT_GET_AES

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

Définition à la ligne 35 du fichier [define.h](#).

4.1.1.22 AT_GET_AES_KEY

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

Définition à la ligne 36 du fichier [define.h](#).

4.1.1.23 AT_GET_API

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

Définition à la ligne 33 du fichier [define.h](#).

4.1.1.24 AT_GET_BAUDRATE

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

Définition à la ligne 34 du fichier [define.h](#).

4.1.1.25 AT_GET_CHANEL

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

Définition à la ligne 37 du fichier [define.h](#).

4.1.1.26 AT_GET_COORDINATOR

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

Définition à la ligne 39 du fichier [define.h](#).

4.1.1.27 AT_GET_LOW_DEST_ADDR

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

Définition à la ligne 42 du fichier [define.h](#).

4.1.1.28 AT_GET_PAN_ID

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

Définition à la ligne 38 du fichier [define.h](#).

4.1.1.29 AT_GET_PARITY

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

Définition à la ligne 40 du fichier [define.h](#).

4.1.1.30 AT_MODE_GET

```
#define AT_MODE_GET 1
```

Définition à la ligne 59 du fichier [define.h](#).

4.1.1.31 AT_MODE_SET

```
#define AT_MODE_SET 2
```

Définition à la ligne 60 du fichier [define.h](#).

4.1.1.32 AT_SUCCESS_VALUE

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

Définition à la ligne 56 du fichier [define.h](#).

4.1.1.33 AT_VALUE_16BIT_SOURCE_ADDR

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

Définition à la ligne 52 du fichier [define.h](#).

4.1.1.34 AT_VALUE_AES

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

Définition à la ligne 46 du fichier [define.h](#).

4.1.1.35 AT_VALUE_AES_KEY

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

Définition à la ligne 47 du fichier [define.h](#).

4.1.1.36 AT_VALUE_API

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

Définition à la ligne 44 du fichier [define.h](#).

4.1.1.37 AT_VALUE_BAUDRATE

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

Définition à la ligne 45 du fichier [define.h](#).

4.1.1.38 AT_VALUE_CHANEL

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

Définition à la ligne 48 du fichier [define.h](#).

4.1.1.39 AT_VALUE_COORDINATOR

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

Définition à la ligne 50 du fichier [define.h](#).

4.1.1.40 AT_VALUE_LOW_DEST_ADDR

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

Définition à la ligne 53 du fichier [define.h](#).

4.1.1.41 AT_VALUE_PAN_ID

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

Définition à la ligne 49 du fichier [define.h](#).

4.1.1.42 AT_VALUE_PARITY

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

Définition à la ligne 51 du fichier [define.h](#).

4.1.1.43 AT_WRITE_CONFIG

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

Définition à la ligne 31 du fichier [define.h](#).

4.1.1.44 BAUDRATE

```
#define BAUDRATE 9600
```

Définition à la ligne 6 du fichier [define.h](#).

4.1.1.45 BROADCAST

```
#define BROADCAST 0x0A
```

Définition à la ligne 12 du fichier [define.h](#).

4.1.1.46 CURRENT_ROBOT

```
#define CURRENT_ROBOT ROBOT_01
```

Définition à la ligne 16 du fichier [define.h](#).

4.1.1.47 DATABITS

```
#define DATABITS SERIAL_DATABITS_8
```

Définition à la ligne 7 du fichier [define.h](#).

4.1.1.48 END_SEQ

```
#define END_SEQ 0x04
```

Définition à la ligne 20 du fichier [define.h](#).

4.1.1.49 ERROR_SUCCESS

```
#define ERROR_SUCCESS 0
```

Définition à la ligne 79 du fichier [define.h](#).

4.1.1.50 PARITY

```
#define PARITY SERIAL_PARITY_NONE
```

Définition à la ligne 8 du fichier [define.h](#).

4.1.1.51 ROBOT_01

```
#define ROBOT_01 0x01
```

Définition à la ligne 13 du fichier [define.h](#).

4.1.1.52 ROBOT_02

```
#define ROBOT_02 0x02
```

Définition à la ligne 14 du fichier [define.h](#).

4.1.1.53 SERIAL_PORT

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

Définition à la ligne 5 du fichier [define.h](#).

4.1.1.54 START_SEQ

```
#define START_SEQ 0x02
```

Définition à la ligne 19 du fichier [define.h](#).

4.1.1.55 STOPBITS

```
#define STOPBITS SERIAL_STOPBITS_1
```

Définition à la ligne 9 du fichier [define.h](#).

4.1.1.56 TEST_ALIVE

```
#define TEST_ALIVE 0x01
```

Définition à la ligne 25 du fichier [define.h](#).

4.2 define.h

[Aller à la documentation de ce fichier.](#)

```
00001 #ifndef DEFINE_XBEE_H
00002 #define DEFINE_XBEE_H
00003
00004 // Paramètres du port série
00005 #define SERIAL_PORT "/dev/ttyAMA0"
00006 #define BAUDRATE 9600
00007 #define DATABITS SERIAL_DATABITS_8
00008 #define PARITY SERIAL_PARITY_NONE
00009 #define STOPBITS SERIAL_STOPBITS_1
00010
00011 // Adresses des robots
00012 #define BROADCAST 0x0A
00013 #define ROBOT_01 0x01
00014 #define ROBOT_02 0x02
00015
00016 #define CURRENT_ROBOT ROBOT_01
00017
00018 // Paramètres de la trame message
00019 #define START_SEQ 0x02
00020 #define END_SEQ 0x04
00021
00022 static unsigned char ID_TRAME = 0x00;
00023
00024 // Codes fonctions
00025 #define TEST_ALIVE 0x01
00026
```

```

00027 // Commandes AT
00028 #define AT_ENTER "+++"
00029 #define AT_EXIT "ATCN"
00030 #define AT_END_LINE "\r"
00031 #define AT_WRITE_CONFIG "ATWR"
00032
00033 #define AT_GET_API "ATAP"
00034 #define AT_GET_BAUDRATE "ATBD"
00035 #define AT_GET_AES "ATEE"
00036 #define AT_GET_AES_KEY "ATKY"
00037 #define AT_GET_CHANNEL "ATCH"
00038 #define AT_GET_PAN_ID "ATID"
00039 #define AT_GET_COORDINATOR "ATCE"
00040 #define AT_GET_PARITY "ATNB"
00041 #define AT_GET_16BIT_SOURCE_ADDR "ATMY"
00042 #define AT_GET_LOW_DEST_ADDR "ATDL"
00043
00044 #define AT_VALUE_API "1"
00045 #define AT_VALUE_BAUDRATE "3"
00046 #define AT_VALUE_AES "1"
00047 #define AT_VALUE_AES_KEY "32303032"
00048 #define AT_VALUE_CHANNEL "C"
00049 #define AT_VALUE_PAN_ID "3332"
00050 #define AT_VALUE_COORDINATOR "0"
00051 #define AT_VALUE_PARITY "0"
00052 #define AT_VALUE_16BIT_SOURCE_ADDR "2"
00053 #define AT_VALUE_LOW_DEST_ADDR "1"
00054
00055 #define AT_EMPTY_VALUE ""
00056 #define AT_SUCCESS_VALUE "OK\r"
00057 #define AT_ERROR_VALUE "ERROR\r"
00058
00059 #define AT_MODE_GET 1
00060 #define AT_MODE_SET 2
00061
00062 // Codes d'erreurs en mode AT
00063 #define AT_ERROR_ENTER -1
00064 #define AT_ERROR_API -2
00065 #define AT_ERROR_BAUDRATE -3
00066 #define AT_ERROR_AES -4
00067 #define AT_ERROR_AES_KEY -13
00068 #define AT_ERROR_CHANNEL -5
00069 #define AT_ERROR_PAN_ID -6
00070 #define AT_ERROR_COORDINATOR -7
00071 #define AT_ERROR_PARITY -8
00072 #define AT_ERROR_16BIT_SOURCE_ADDR -9
00073 #define AT_ERROR_LOW_DEST_ADDR -10
00074 #define AT_ERROR_EXIT -11
00075 #define AT_ERROR_WRITE_CONFIG -12
00076 #define AT_ERROR_SUCCESS 0
00077
00078 // Codes d'erreurs
00079 #define ERROR_SUCCESS 0
00080
00081 #endif

```

4.3 Référence du fichier main.cpp

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

Fonctions

— int `main` (int argc, char *argv[])

4.3.1 Documentation des fonctions

4.3.1.1 main()

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

Définition à la ligne 5 du fichier [main.cpp](#).

4.4 main.cpp

[Aller à la documentation de ce fichier.](#)

```
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 <<
00013         "]" << endl;
00014     else
00015         cout << ": Connexion ouverte avec succès sur le port \"" << SERIAL_PORT << "\".\n" << endl;
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 Référence du fichier serialib.cpp

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

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

4.5.1 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

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE X CONSORTIUM BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

This is a licence-free software, it can be used by anyone who try to build a better world.

Définition dans le fichier [serialib.cpp](#).

4.6 serialib.cpp

[Aller à la documentation de ce fichier.](#)

```
00001
00018 #include "serialib.h"
00019
00020
00021
00022 // _____
00023 // ::: Constructors and destructors :::
00024
00025
00029 serialib::serialib()
00030 {
00031     #if defined (_WIN32) || defined( _WIN64)
00032         // Set default value for RTS and DTR (Windows only)
00033         currentStateRTS=true;
00034         currentStateDTR=true;
00035         hSerial = INVALID_HANDLE_VALUE;
00036     #endif
00037     #if defined (__linux__) || defined(__APPLE__)
00038         fd = -1;
00039     #endif
00040 }
00041
00042
00046 // Class desctructor
00047 serialib::~serialib()
00048 {
00049     closeDevice();
00050 }
00051
00052
00053
00054 // _____
00055 // ::: Configuration and initialization :::
00056
00057
00058
00128 char serialib::openDevice(const char *Device, const unsigned int Bauds,
00129                           SerialDataBits Databits,
00130                           SerialParity Parity,
00131                           SerialStopBits Stopbits) {
00132     #if defined (_WIN32) || defined( _WIN64)
00133         // Open serial port
00134         hSerial = CreateFileA(Device, GENERIC_READ |
00135                               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 -1; // Device not found
00139             // Error while opening the device
00140             return -2;
00141         }
00142
00143         // Set parameters
00144
00145         // Structure for the port parameters
00146         DCB dcbSerialParams;
00147         dcbSerialParams.DCBlength=sizeof(dcbSerialParams);
```

```

00148
00149 // Get the port parameters
00150 if (!GetCommState(hSerial, &dcbSerialParams)) return -3;
00151
00152 // Set the speed (Bauds)
00153 switch (Bauds)
00154 {
00155 case 110 : dcbSerialParams.BaudRate=CBR_110; break;
00156 case 300 : dcbSerialParams.BaudRate=CBR_300; break;
00157 case 600 : dcbSerialParams.BaudRate=CBR_600; break;
00158 case 1200 : dcbSerialParams.BaudRate=CBR_1200; break;
00159 case 2400 : dcbSerialParams.BaudRate=CBR_2400; break;
00160 case 4800 : dcbSerialParams.BaudRate=CBR_4800; break;
00161 case 9600 : dcbSerialParams.BaudRate=CBR_9600; break;
00162 case 14400 : dcbSerialParams.BaudRate=CBR_14400; break;
00163 case 19200 : dcbSerialParams.BaudRate=CBR_19200; break;
00164 case 38400 : dcbSerialParams.BaudRate=CBR_38400; break;
00165 case 56000 : dcbSerialParams.BaudRate=CBR_56000; break;
00166 case 57600 : dcbSerialParams.BaudRate=CBR_57600; break;
00167 case 115200 : dcbSerialParams.BaudRate=CBR_115200; break;
00168 case 128000 : dcbSerialParams.BaudRate=CBR_128000; break;
00169 case 256000 : dcbSerialParams.BaudRate=CBR_256000; break;
00170 default : return -4;
00171 }
00172 //select data size
00173 BYTE bytesize = 0;
00174 switch (Databits) {
00175 case SERIAL_DATABITS_5: bytesize = 5; break;
00176 case SERIAL_DATABITS_6: bytesize = 6; break;
00177 case SERIAL_DATABITS_7: bytesize = 7; break;
00178 case SERIAL_DATABITS_8: bytesize = 8; break;
00179 case SERIAL_DATABITS_16: bytesize = 16; break;
00180 default: return -7;
00181 }
00182 BYTE stopBits = 0;
00183 switch (Stopbits) {
00184 case SERIAL_STOPBITS_1: stopBits = ONESTOPBIT; break;
00185 case SERIAL_STOPBITS_1_5: stopBits = ONE5STOPBITS; break;
00186 case SERIAL_STOPBITS_2: stopBits = TWOSTOPBITS; break;
00187 default: return -8;
00188 }
00189 BYTE parity = 0;
00190 switch (Parity) {
00191 case SERIAL_PARITY_NONE: parity = NOPARITY; break;
00192 case SERIAL_PARITY_EVEN: parity = EVENPARITY; break;
00193 case SERIAL_PARITY_ODD: parity = ODDPARITY; break;
00194 case SERIAL_PARITY_MARK: parity = MARKPARITY; break;
00195 case SERIAL_PARITY_SPACE: parity = SPACEPARITY; break;
00196 default: return -9;
00197 }
00198 // configure byte size
00199 dcbSerialParams.ByteSize = bytesize;
00200 // configure stop bits
00201 dcbSerialParams.StopBits = stopBits;
00202 // configure parity
00203 dcbSerialParams.Parity = parity;
00204
00205 // Write the parameters
00206 if (!SetCommState(hSerial, &dcbSerialParams)) return -5;
00207
00208 // Set TimeOut
00209
00210 // Set the Timeout parameters
00211 timeouts.ReadIntervalTimeout=0;
00212 // No TimeOut
00213 timeouts.ReadTotalTimeoutConstant=MAXDWORD;
00214 timeouts.ReadTotalTimeoutMultiplier=0;
00215 timeouts.WriteTotalTimeoutConstant=MAXDWORD;
00216 timeouts.WriteTotalTimeoutMultiplier=0;
00217
00218 // Write the parameters
00219 if (!SetCommTimeouts(hSerial, &timeouts)) return -6;
00220
00221 // Opening successfull
00222 return 1;
00223 #endif
00224 #if defined (__linux__) || defined (__APPLE__)
00225 // Structure with the device's options
00226 struct termios options;
00227
00228 // Open device
00229 fd = open(Device, O_RDWR | O_NOCTTY | O_NDELAY);
00230 // If the device is not open, return -1
00231 if (fd == -1) return -2;
00232 // Open the device in nonblocking mode
00233 fcntl(fd, F_SETFL, FNDELAY);

```

```

00235
00236
00237 // Get the current options of the port
00238 tcgetattr(fd, &options);
00239 // Clear all the options
00240 bzero(&options, sizeof(options));
00241
00242 // Prepare speed (Bauds)
00243 speed_t Speed;
00244 switch (Bauds)
00245 {
00246 case 110 : Speed=B110; break;
00247 case 300 : Speed=B300; break;
00248 case 600 : Speed=B600; break;
00249 case 1200 : Speed=B1200; break;
00250 case 2400 : Speed=B2400; break;
00251 case 4800 : Speed=B4800; break;
00252 case 9600 : Speed=B9600; break;
00253 case 19200 : Speed=B19200; break;
00254 case 38400 : Speed=B38400; break;
00255 case 57600 : Speed=B57600; break;
00256 case 115200 : Speed=B115200; break;
00257 default : return -4;
00258 }
00259 int databits_flag = 0;
00260 switch(Databits) {
00261 case SERIAL_DATABITS_5: databits_flag = CS5; break;
00262 case SERIAL_DATABITS_6: databits_flag = CS6; break;
00263 case SERIAL_DATABITS_7: databits_flag = CS7; break;
00264 case SERIAL_DATABITS_8: databits_flag = CS8; break;
00265 //16 bits and everything else not supported
00266 default: return -7;
00267 }
00268 int stopbits_flag = 0;
00269 switch(Stopbits) {
00270 case SERIAL_STOPBITS_1: stopbits_flag = 0; break;
00271 case SERIAL_STOPBITS_2: stopbits_flag = CSTOPB; break;
00272 //1.5 stopbits and everything else not supported
00273 default: return -8;
00274 }
00275 int parity_flag = 0;
00276 switch(Parity) {
00277 case SERIAL_PARITY_NONE: parity_flag = 0; break;
00278 case SERIAL_PARITY_EVEN: parity_flag = PARENB; break;
00279 case SERIAL_PARITY_ODD: parity_flag = (PARENB | PARODD); break;
00280 //mark and space parity not supported
00281 default: return -9;
00282 }
00283
00284 // Set the baud rate
00285 cfsetispeed(&options, Speed);
00286 cfsetospeed(&options, Speed);
00287 // Configure the device : data bits, stop bits, parity, no control flow
00288 // Ignore modem control lines (CLOCAL) and Enable receiver (CREAD)
00289 options.c_cflag |= ( CLOCAL | CREAD | databits_flag | parity_flag | stopbits_flag);
00290 options.c_iflag |= ( IGNPAR | IGNBRK );
00291 // Timer unused
00292 options.c_cc[VTIME]=0;
00293 // At least on character before satisfy reading
00294 options.c_cc[VMIN]=0;
00295 // Activate the settings
00296 tcsetattr(fd, TCSANOW, &options);
00297 // Success
00298 return (1);
00299 #endif
00300 }
00301 }
00302
00303 bool seriallib::isDeviceOpen()
00304 {
00305 #if defined ( _WIN32 ) || defined( _WIN64 )
00306 return hSerial != INVALID_HANDLE_VALUE;
00307 #endif
00308 #if defined ( __linux__ ) || defined( __APPLE__ )
00309 return fd >= 0;
00310 #endif
00311 }
00312
00313 void seriallib::closeDevice()
00314 {
00315 #if defined ( _WIN32 ) || defined( _WIN64 )
00316 CloseHandle(hSerial);
00317 hSerial = INVALID_HANDLE_VALUE;
00318 #endif
00319 #if defined ( __linux__ ) || defined( __APPLE__ )
00320 close (fd);
00321 fd = -1;

```

```

00325 #endif
00326 }
00327
00328
00329
00330
00331 // _____
00332 // ::: Read/Write operation on characters :::
00333
00334
00335
00342 char serialib::writeChar(const char Byte)
00343 {
00344     #if defined (_WIN32) || defined( _WIN64)
00345         // Number of bytes written
00346         DWORD dwBytesWritten;
00347         // Write the char to the serial device
00348         // Return -1 if an error occurred
00349         if(!WriteFile(hSerial,&Byte,1,&dwBytesWritten,NULL)) return -1;
00350         // Write operation successfull
00351         return 1;
00352     #endif
00353     #if defined (__linux__) || defined(__APPLE__)
00354         // Write the char
00355         if (write(fd,&Byte,1)!=1) return -1;
00356         // Write operation successfull
00357         return 1;
00358     #endif
00359 }
00360
00361
00362
00363
00364 // _____
00365 // ::: Read/Write operation on strings :::
00366
00367
00374 char serialib::writeString(const char *receivedString)
00375 {
00376     #if defined (_WIN32) || defined( _WIN64)
00377         // Number of bytes written
00378         DWORD dwBytesWritten;
00379         // Write the string
00380         if(!WriteFile(hSerial,receivedString,strlen(receivedString),&dwBytesWritten,NULL))
00381             // Error while writing, return -1
00382             return -1;
00383         // Write operation successfull
00384         return 1;
00385     #endif
00386     #if defined (__linux__) || defined(__APPLE__)
00387         // Length of the string
00388         int Length=strlen(receivedString);
00389         // Write the string
00390         if (write(fd,receivedString,Length)!=Length) return -1;
00391         // Write operation successfull
00392         return 1;
00393     #endif
00394 }
00395
00396 // _____
00397 // ::: Read/Write operation on bytes :::
00398
00399
00400
00408 char serialib::writeBytes(const void *Buffer, const unsigned int NbBytes)
00409 {
00410     #if defined (_WIN32) || defined( _WIN64)
00411         // Number of bytes written
00412         DWORD dwBytesWritten;
00413         // Write data
00414         if(!WriteFile(hSerial, Buffer, NbBytes, &dwBytesWritten, NULL))
00415             // Error while writing, return -1
00416             return -1;
00417         // Write operation successfull
00418         return 1;
00419     #endif
00420     #if defined (__linux__) || defined(__APPLE__)
00421         // Write data
00422         if (write (fd,Buffer,NbBytes)!=((ssize_t)NbBytes) return -1;
00423         // Write operation successfull
00424         return 1;
00425     #endif
00426 }
00427
00428
00429
00440 char serialib::readChar(char *pByte,unsigned int timeOut_ms)

```

```

00441 {
00442 #if defined (_WIN32) || defined(_WIN64)
00443     // Number of bytes read
00444     DWORD dwBytesRead = 0;
00445
00446     // Set the Timeout
00447     timeouts.ReadTotalTimeoutConstant=timeOut_ms;
00448
00449     // Write the parameters, return -1 if an error occurred
00450     if(!SetCommTimeouts(hSerial, &timeouts)) return -1;
00451
00452     // Read the byte, return -2 if an error occurred
00453     if(!ReadFile(hSerial,pByte, 1, &dwBytesRead, NULL)) return -2;
00454
00455     // Return 0 if the timeout is reached
00456     if (dwBytesRead==0) return 0;
00457
00458     // The byte is read
00459     return 1;
00460 #endif
00461 #if defined (__linux__) || defined(__APPLE__)
00462     // Timer used for timeout
00463     timer_t timer;
00464     // Initialise the timer
00465     timer.initTimer();
00466     // While Timeout is not reached
00467     while (timer.elapsedTime_ms() < timeOut_ms || timeOut_ms==0)
00468     {
00469         // Try to read a byte on the device
00470         switch (read(fd,pByte,1)) {
00471             case 1 : return 1; // Read successfull
00472             case -1 : return -2; // Error while reading
00473         }
00474     }
00475     return 0;
00476 #endif
00477 }
00478
00479
00480
00491 int serialib::readStringNoTimeOut(char *receivedString,char finalChar,unsigned int maxNbBytes)
00492 {
00493     // Number of characters read
00494     unsigned int NbBytes=0;
00495     // Returned value from Read
00496     char charRead;
00497
00498     // While the buffer is not full
00499     while (NbBytes<maxNbBytes)
00500     {
00501         // Read a character with the restant time
00502         charRead=readChar(&receivedString[NbBytes]);
00503
00504         // Check a character has been read
00505         if (charRead==1)
00506         {
00507             // Check if this is the final char
00508             if (receivedString[NbBytes]==finalChar)
00509             {
00510                 // This is the final char, add zero (end of string)
00511                 receivedString [++NbBytes]=0;
00512                 // Return the number of bytes read
00513                 return NbBytes;
00514             }
00515
00516             // The character is not the final char, increase the number of bytes read
00517             NbBytes++;
00518         }
00519
00520         // An error occurred while reading, return the error number
00521         if (charRead<0) return charRead;
00522     }
00523     // Buffer is full : return -3
00524     return -3;
00525 }
00526
00527
00540 int serialib::readString(char *receivedString,char finalChar,unsigned int maxNbBytes,unsigned int
    timeOut_ms)
00541 {
00542     // Check if timeout is requested
00543     if (timeOut_ms==0) return readStringNoTimeOut(receivedString,finalChar,maxNbBytes);
00544
00545     // Number of bytes read
00546     unsigned int nbBytes=0;
00547     // Character read on serial device
00548     char charRead;

```

```

00549 // Timer used for timeout
00550 timeout timer;
00551 long int timeoutParam;
00552
00553 // Initialize the timer (for timeout)
00554 timer.initTimer();
00555
00556 // While the buffer is not full
00557 while (nbBytes<maxNbBytes)
00558 {
00559     // Compute the TimeOut for the next call of ReadChar
00560     timeoutParam = timeout_ms-timer.elapsedTime_ms();
00561
00562     // If there is time remaining
00563     if (timeoutParam>0)
00564     {
00565         // Wait for a byte on the serial link with the remaining time as timeout
00566         charRead=readChar(&receivedString[nbBytes],timeoutParam);
00567
00568         // If a byte has been received
00569         if (charRead==1)
00570         {
00571             // Check if the character received is the final one
00572             if (receivedString[nbBytes]==finalChar)
00573             {
00574                 // Final character: add the end character 0
00575                 receivedString [++nbBytes]=0;
00576                 // Return the number of bytes read
00577                 return nbBytes;
00578             }
00579             // This is not the final character, just increase the number of bytes read
00580             nbBytes++;
00581         }
00582         // Check if an error occurred during reading char
00583         // If an error occurred, return the error number
00584         if (charRead<0) return charRead;
00585     }
00586     // Check if timeout is reached
00587     if (timer.elapsedTime_ms()>timeout_ms)
00588     {
00589         // Add the end character
00590         receivedString[nbBytes]=0;
00591         // Return 0 (timeout reached)
00592         return 0;
00593     }
00594 }
00595
00596 // Buffer is full : return -3
00597 return -3;
00598 }
00599
00600
00614 int serialib::readBytes (void *buffer,unsigned int maxNbBytes,unsigned int timeout_ms, unsigned int
sleepDuration_us)
00615 {
00616 #if defined (_WIN32) || defined(_WIN64)
00617 // Avoid warning while compiling
00618 UNUSED(sleepDuration_us);
00619
00620 // Number of bytes read
00621 DWORD dwBytesRead = 0;
00622
00623 // Set the TimeOut
00624 timeouts.ReadTotalTimeoutConstant=(DWORD)timeout_ms;
00625
00626 // Write the parameters and return -1 if an error occurred
00627 if(!SetCommTimeouts(hSerial, &timeouts)) return -1;
00628
00629 // Read the bytes from the serial device, return -2 if an error occurred
00630 if(!ReadFile(hSerial,buffer,(DWORD)maxNbBytes,&dwBytesRead, NULL)) return -2;
00631
00632 // Return the byte read
00633 return dwBytesRead;
00634 #endif
00635 #endif
00636 #if defined (__linux__) || defined(__APPLE__)
00637 // Timer used for timeout
00638 timeout timer;
00639 // Initialise the timer
00640 timer.initTimer();
00641 unsigned int NbByteRead=0;
00642 // While Timeout is not reached
00643 while (timer.elapsedTime_ms()<timeout_ms || timeout_ms==0)
00644 {
00645     // Compute the position of the current byte
00646     unsigned char* Ptr=(unsigned char*)buffer+NbByteRead;
00647     // Try to read a byte on the device

```

```

00648         int Ret=read(fd, (void*)Ptr,maxNbBytes-NbByteRead);
00649         // Error while reading
00650         if (Ret==-1) return -2;
00651
00652         // One or several byte(s) has been read on the device
00653         if (Ret>0)
00654         {
00655             // Increase the number of read bytes
00656             NbByteRead+=Ret;
00657             // Success : bytes has been read
00658             if (NbByteRead>=maxNbBytes)
00659                 return NbByteRead;
00660         }
00661         // Suspend the loop to avoid charging the CPU
00662         usleep (sleepDuration_us);
00663     }
00664     // Timeout reached, return the number of bytes read
00665     return NbByteRead;
00666 #endif
00667 }
00668
00669
00670
00671
00672 // _____
00673 // ::: Special operation :::
00674
00675
00676
00682 char seriallib::flushReceiver()
00683 {
00684     #if defined (_WIN32) || defined(_WIN64)
00685         // Purge receiver
00686         return PurgeComm (hSerial, PURGE_RXCLEAR);
00687     #endif
00688     #if defined (__linux__) || defined(__APPLE__)
00689         // Purge receiver
00690         tcflush(fd,TCIFLUSH);
00691         return true;
00692     #endif
00693 }
00694
00695
00696
00701 int seriallib::available()
00702 {
00703     #if defined (_WIN32) || defined(_WIN64)
00704         // Device errors
00705         DWORD commErrors;
00706         // Device status
00707         COMSTAT commStatus;
00708         // Read status
00709         ClearCommError(hSerial, &commErrors, &commStatus);
00710         // Return the number of pending bytes
00711         return commStatus.cbInQue;
00712     #endif
00713     #if defined (__linux__) || defined(__APPLE__)
00714         int nBytes=0;
00715         // Return number of pending bytes in the receiver
00716         ioctl(fd, FIONREAD, &nBytes);
00717         return nBytes;
00718     #endif
00719 }
00720 }
00721
00722
00723
00724 // _____
00725 // ::: I/O Access :::
00726
00736 bool seriallib::DTR(bool status)
00737 {
00738     if (status)
00739         // Set DTR
00740         return this->setDTR();
00741     else
00742         // Unset DTR
00743         return this->clearDTR();
00744 }
00745
00746
00753 bool seriallib::setDTR()
00754 {
00755     #if defined (_WIN32) || defined(_WIN64)
00756         // Set DTR
00757         currentStateDTR=true;
00758         return EscapeCommFunction (hSerial, SETDTR);

```



```

00759 #endif
00760 #if defined (__linux__) || defined(__APPLE__)
00761     // Set DTR
00762     int status_DTR=0;
00763     ioctl(fd, TIOCMGET, &status_DTR);
00764     status_DTR |= TIOCM_DTR;
00765     ioctl(fd, TIOCMSET, &status_DTR);
00766     return true;
00767 #endif
00768 }
00769
00776 bool serialib::clearDTR()
00777 {
00778     #if defined (_WIN32) || defined(_WIN64)
00779         // Clear DTR
00780         currentStateDTR=true;
00781         return EscapeCommFunction(hSerial, CLRDR);
00782     #endif
00783     #if defined (__linux__) || defined(__APPLE__)
00784         // Clear DTR
00785         int status_DTR=0;
00786         ioctl(fd, TIOCMGET, &status_DTR);
00787         status_DTR &= ~TIOCM_DTR;
00788         ioctl(fd, TIOCMSET, &status_DTR);
00789         return true;
00790     #endif
00791 }
00792
00793
00794
00804 bool serialib::RTS(bool status)
00805 {
00806     if (status)
00807         // Set RTS
00808         return this->setRTS();
00809     else
00810         // Unset RTS
00811         return this->clearRTS();
00812 }
00813
00814
00821 bool serialib::setRTS()
00822 {
00823     #if defined (_WIN32) || defined(_WIN64)
00824         // Set RTS
00825         currentStateRTS=false;
00826         return EscapeCommFunction(hSerial, SETRTS);
00827     #endif
00828     #if defined (__linux__) || defined(__APPLE__)
00829         // Set RTS
00830         int status_RTS=0;
00831         ioctl(fd, TIOCMGET, &status_RTS);
00832         status_RTS |= TIOCM_RTS;
00833         ioctl(fd, TIOCMSET, &status_RTS);
00834         return true;
00835     #endif
00836 }
00837
00838
00839
00846 bool serialib::clearRTS()
00847 {
00848     #if defined (_WIN32) || defined(_WIN64)
00849         // Clear RTS
00850         currentStateRTS=false;
00851         return EscapeCommFunction(hSerial, CLRRTS);
00852     #endif
00853     #if defined (__linux__) || defined(__APPLE__)
00854         // Clear RTS
00855         int status_RTS=0;
00856         ioctl(fd, TIOCMGET, &status_RTS);
00857         status_RTS &= ~TIOCM_RTS;
00858         ioctl(fd, TIOCMSET, &status_RTS);
00859         return true;
00860     #endif
00861 }
00862
00863
00864
00865
00871 bool serialib::isCTS()
00872 {
00873     #if defined (_WIN32) || defined(_WIN64)
00874         DWORD modemStat;
00875         GetCommModemStatus(hSerial, &modemStat);
00876         return modemStat & MS_CTS_ON;
00877     #endif

```

```
00878 #if defined (__linux__) || defined(__APPLE__)
00879     int status=0;
00880     //Get the current status of the CTS bit
00881     ioctl(fd, TIOCMGET, &status);
00882     return status & TIOCM_CTS;
00883 #endif
00884 }
00885
00886
00887
00893 bool seriallib::isDSR()
00894 {
00895     #if defined (_WIN32) || defined(_WIN64)
00896         DWORD modemStat;
00897         GetCommModemStatus(hSerial, &modemStat);
00898         return modemStat & MS_DSR_ON;
00899     #endif
00900     #if defined (__linux__) || defined(__APPLE__)
00901         int status=0;
00902         //Get the current status of the DSR bit
00903         ioctl(fd, TIOCMGET, &status);
00904         return status & TIOCM_DSR;
00905     #endif
00906 }
00907
00908
00909
00910
00911
00912
00919 bool seriallib::isDCD()
00920 {
00921     #if defined (_WIN32) || defined(_WIN64)
00922         DWORD modemStat;
00923         GetCommModemStatus(hSerial, &modemStat);
00924         return modemStat & MS_RLSD_ON;
00925     #endif
00926     #if defined (__linux__) || defined(__APPLE__)
00927         int status=0;
00928         //Get the current status of the DCD bit
00929         ioctl(fd, TIOCMGET, &status);
00930         return status & TIOCM_CAR;
00931     #endif
00932 }
00933
00934
00940 bool seriallib::isRI()
00941 {
00942     #if defined (_WIN32) || defined(_WIN64)
00943         DWORD modemStat;
00944         GetCommModemStatus(hSerial, &modemStat);
00945         return modemStat & MS_RING_ON;
00946     #endif
00947     #if defined (__linux__) || defined(__APPLE__)
00948         int status=0;
00949         //Get the current status of the RING bit
00950         ioctl(fd, TIOCMGET, &status);
00951         return status & TIOCM_RNG;
00952     #endif
00953 }
00954
00955
00962 bool seriallib::isDTR()
00963 {
00964     #if defined (_WIN32) || defined(_WIN64)
00965         return currentStateDTR;
00966     #endif
00967     #if defined (__linux__) || defined(__APPLE__)
00968         int status=0;
00969         //Get the current status of the DTR bit
00970         ioctl(fd, TIOCMGET, &status);
00971         return status & TIOCM_DTR ;
00972     #endif
00973 }
00974
00975
00976
00983 bool seriallib::isRTS()
00984 {
00985     #if defined (_WIN32) || defined(_WIN64)
00986         return currentStateRTS;
00987     #endif
00988     #if defined (__linux__) || defined(__APPLE__)
00989         int status=0;
00990         //Get the current status of the CTS bit
00991         ioctl(fd, TIOCMGET, &status);
00992         return status & TIOCM_RTS;
```

```

00993 #endif
00994 }
00995
00996
00997
00998
00999
01000
01001 // *****
01002 // Class timeOut
01003 // *****
01004
01005
01009 // Constructor
01010 timeOut::timeOut()
01011 {}
01012
01013
01017 //Initialize the timer
01018 void timeOut::initTimer()
01019 {
01020 #if defined (NO_POSIX_TIME)
01021     LARGE_INTEGER tmp;
01022     QueryPerformanceFrequency(&tmp);
01023     counterFrequency = tmp.QuadPart;
01024     // Used to store the previous time (for computing timeout)
01025     QueryPerformanceCounter(&tmp);
01026     previousTime = tmp.QuadPart;
01027 #else
01028     gettimeofday(&previousTime, NULL);
01029 #endif
01030 }
01031
01037 //Return the elapsed time since initialization
01038 unsigned long int timeOut::elapsedTime_ms()
01039 {
01040 #if defined (NO_POSIX_TIME)
01041     // Current time
01042     LARGE_INTEGER CurrentTime;
01043     // Number of ticks since last call
01044     int sec;
01045
01046     // Get current time
01047     QueryPerformanceCounter(&CurrentTime);
01048
01049     // Compute the number of ticks elapsed since last call
01050     sec=CurrentTime.QuadPart-previousTime;
01051
01052     // Return the elapsed time in milliseconds
01053     return sec/(counterFrequency/1000);
01054 #else
01055     // Current time
01056     struct timeval CurrentTime;
01057     // Number of seconds and microseconds since last call
01058     int sec,usec;
01059
01060     // Get current time
01061     gettimeofday(&CurrentTime, NULL);
01062
01063     // Compute the number of seconds and microseconds elapsed since last call
01064     sec=CurrentTime.tv_sec-previousTime.tv_sec;
01065     usec=CurrentTime.tv_usec-previousTime.tv_usec;
01066
01067     // If the previous usec is higher than the current one
01068     if (usec<0)
01069     {
01070         // Recompute the microseconds and subtract one second
01071         usec=1000000-previousTime.tv_usec+CurrentTime.tv_usec;
01072         sec--;
01073     }
01074
01075     // Return the elapsed time in milliseconds
01076     return sec*1000+usec/1000;
01077 #endif
01078 }

```

4.7 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](#) }

4.7.1 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.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE X CONSORTIUM BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

This is a licence-free software, it can be used by anyone who try to build a better world.

Définition dans le fichier [serialib.h](#).

4.7.2 Documentation des macros

4.7.2.1 UNUSED

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

To avoid unused parameters

Définition à la ligne 56 du fichier [serialib.h](#).

4.7.3 Documentation du type de l'énumération

4.7.3.1 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](#).

4.7.3.2 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](#).

4.7.3.3 SerialStopBits

enum `SerialStopBits`

number of serial stop bits

Valeurs énumérées

<code>SERIAL_STOPBITS_1</code>	1 stop bit
<code>SERIAL_STOPBITS_1_5</code>	1.5 stop bits
<code>SERIAL_STOPBITS_2</code>	2 stop bits

Définition à la ligne 72 du fichier `serialib.h`.

4.8 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     char 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,
00165                     unsigned int sleepDuration_us=100);
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
00241 // Class timeOut
00242 class timeOut
00243 {
00244 public:
00245 // Constructor
00246 timeOut();
00247
00248 // Init the timer
00249 void initTimer();
00250
00251 // Return the elapsed time since initialization
00252 unsigned long int elapsedTime_ms();
00253
00254 private:
00255 #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 // seriallib_H
```

4.9 Référence du fichier xbee.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 "xbee.h"
```

Classes

— struct [Tframe](#)

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

Variables

— [serialib](#) [serial](#)

4.9.1 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

1.0

Date

03/02/2022

Définition dans le fichier [xbee.cpp](#).

4.9.2 Documentation des variables

4.9.2.1 serial

`serialib` `serial`

Définition à la ligne 12 du fichier `xbee.cpp`.

4.10 xbee.cpp

[Aller à la documentation de ce fichier.](#)

```

00001
00008 #include "xbee.h"
00009
00010 using namespace std;
00011
00012 serialib serial;
00013
00018 struct Trame{
00019     int id_exp;
00020     int id_dest;
00021     int code_fct;
00022     int id_trame;
00023     int size;
00024     vector<char> data;
00025 };
00026
00027 // _____
00028 // ::: Constructeurs et destructeurs :::
00029
00033 xbee::xbee(){ }
00034
00038 xbee::~xbee(){ }
00039
00040
00041 // _____
00042 // ::: Configuration and initialisation :::
00043
00057 int xbee::openSerialConnection(){
00058     serial.flushReceiver();
00059     char errorOpening = serial.openDevice(SERIAL_PORT, BAUDRATE, DATABITS, PARITY, STOPBITS);
00060
00061     return (int) errorOpening;
00062 }
00063
00067 void xbee::closeSerialConnection(){
00068     serial.flushReceiver();
00069     serial.closeDevice();
00070 }
00071
00072 // _____
00073 // ::: Configuration en mode AT :::
00074
00091 int xbee::checkATConfig(){
00092     if(!enterATMode())
00093         return AT_ERROR_ENTER;
00094
00095     if(!sendATCommand(AT_GET_API, AT_VALUE_API, AT_MODE_SET))
00096         return AT_ERROR_API;
00097
00098     if(!sendATCommand(AT_GET_BAUDRATE, AT_VALUE_BAUDRATE, AT_MODE_SET))
00099         return AT_ERROR_BAUDRATE;
00100
00101     if(!sendATCommand(AT_GET_AES, AT_VALUE_AES, AT_MODE_SET))
00102         return AT_ERROR_AES;
00103
00104     if(!sendATCommand(AT_GET_AES_KEY, AT_VALUE_AES_KEY, AT_MODE_SET))
00105         return AT_ERROR_AES_KEY;
00106
00107     if(!sendATCommand(AT_GET_CHANEL, AT_VALUE_CHANEL, AT_MODE_SET))
00108         return AT_ERROR_CHANEL;
00109
00110     if(!sendATCommand(AT_GET_PAN_ID, AT_VALUE_PAN_ID, AT_MODE_SET))
00111         return AT_ERROR_PAN_ID;
00112
00113     if(!sendATCommand(AT_GET_COORDINATOR, AT_VALUE_COORDINATOR, AT_MODE_SET))
00114         return AT_ERROR_COORDINATOR;
00115
00116     if(!sendATCommand(AT_GET_PARITY, AT_VALUE_PARITY, AT_MODE_SET))
00117         return AT_ERROR_PARITY;

```

```

00118
00119     if(!sendATCommand(AT_GET_16BIT_SOURCE_ADDR, AT_VALUE_16BIT_SOURCE_ADDR, AT_MODE_SET))
00120     return AT_ERROR_16BIT_SOURCE_ADDR;
00121
00122     if(!sendATCommand(AT_GET_LOW_DEST_ADDR, AT_VALUE_LOW_DEST_ADDR, AT_MODE_SET))
00123     return AT_ERROR_LOW_DEST_ADDR;
00124
00125     if(!writeATConfig())
00126     return AT_ERROR_WRITE_CONFIG;
00127
00128     if(!exitATMode())
00129     return AT_ERROR_EXIT;
00130
00131     return AT_ERROR_SUCCESS;
00132 }
00133
00138 void xbee::delay(unsigned int time){ usleep(time*1000000); }
00139
00140
00147 bool xbee::readATResponse(const char *value){
00148     char *reponse(0);
00149     unsigned int timeout = 100;
00150     reponse = new char;
00151     delay(1);
00152     string rep = "";
00153     int i = 0;
00154     while(serial.available() > 0){
00155         i++;
00156         serial.readChar(reponse, timeout);
00157         rep += *reponse;
00158     }
00159     delete reponse;
00160     reponse = 0;
00161
00162     if(rep == value)
00163         return true;
00164     else
00165         return false;
00166 }
00167
00173 bool xbee::enterATMode(){
00174     serial.writeString(AT_ENTER);
00175     //cout << " * Entrée en mode AT..." << endl;
00176     delay(2);
00177     serial.writeString(AT_END_LINE);
00178     return readATResponse(AT_SUCCESS_VALUE);
00179 }
00180
00186 bool xbee::exitATMode(){
00187     serial.writeString(AT_EXIT);
00188     serial.writeString(AT_END_LINE);
00189     //cout << " * Sortie du mode AT..." << endl;
00190     return readATResponse(AT_SUCCESS_VALUE);
00191 }
00192
00198 bool xbee::writeATConfig(){
00199     serial.writeString(AT_WRITE_CONFIG);
00200     serial.writeString(AT_END_LINE);
00201     //cout << " * Ecriture de la configuration AT..." << endl;
00202     return readATResponse(AT_SUCCESS_VALUE);
00203 }
00204
00213 bool xbee::sendATCommand(const char *command, const char *value, unsigned int mode){
00214     serial.writeString(command);
00215     serial.writeString(value);
00216     serial.writeString(AT_END_LINE);
00217     if(mode == AT_MODE_GET){
00218         //cout << " * Envoi de la commande " << command << "...\\n";
00219         return readATResponse(value);
00220     }else{
00221         //cout << " * Envoi de la commande " << command << "=" << value << "...\\n";
00222         return readATResponse(AT_SUCCESS_VALUE);
00223     }
00224 }
00225
00226 //_____
00227 // :: Envoi/Réception/Traitement des trames de messages ::
00228
00234 int xbee::crc16(vector<char> trame){
00235     int crc = 0xFFFF, count = 0;
00236     unsigned char octet_a_traiter;
00237     const int POLYNOME = 0xA001;
00238
00239     octet_a_traiter = trame[0];
00240
00241     do{
00242         crc ^= octet_a_traiter;

```

```

00243         for(int i = 0; i < 8; i++){
00244
00245             if((crc%2)!=0)
00246                 crc = (crc » 1) ^ POLYNOME;
00247
00248             else
00249                 crc = (crc » 1);
00250
00251         }
00252         count++;
00253         octet_a_traiter = trame[count];
00254
00255     }while(count < trame.size());
00256
00257     return crc;
00258 }
00259
00266 void xbee::sendTrame(char ad_dest, char code_fct, char data[]){
00267     vector<char> trame;
00268     string convert_data = data;
00269     uint8_t taille_message = (uint8_t) code_fct + (convert_data.size()) + 0x05;
00270
00271     uint8_t high = (taille_message » 8) & 0xFF;
00272     uint8_t low = taille_message & 0xFF;
00273
00274     //cout << taille_message << endl;
00275     //cout << (int) high << endl;
00276     //cout << (int) low << endl;
00277
00278     //char taille_message_h = (char) high;
00279     //char taille_message_l = (char) low;
00280
00281     trame.push_back(START_SEQ);
00282
00283     trame.push_back(CURRENT_ROBOT);
00284     trame.push_back(ad_dest);
00285     trame.push_back(++ID_TRAME);
00286     trame.push_back((char)taille_message);
00287     //trame.push_back(taille_message_l);
00288
00289     trame.push_back(code_fct);
00290
00291     for(int i=0; i < convert_data.size(); i++)
00292         trame.push_back(convert_data[i]);
00293
00294     int crc = crc16(trame);
00295
00296     trame.push_back((char) crc);
00297
00298     trame.push_back(END_SEQ);
00299
00300     for(int i=0; i < trame.size(); i++)
00301         cout << hex << showbase << setw(4) << static_cast<int>(trame[i]);
00302
00303     cout << endl;
00304
00305     char* message = reinterpret_cast<char*>(trame.data());
00306     //serial.writeString(message);
00307
00308 }

```

4.11 Référence du fichier xbee.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 <string>
#include <vector>
#include <iomanip>
#include <iostream>

```

Classes

— class [xbec](#)

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

4.11.1 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

1.0

Date

03/02/2022

Définition dans le fichier [xbec.h](#).

4.12 xbee.h

[Aller à la documentation de ce fichier.](#)

```
00001
00009 #ifndef XBEE_H
00010 #define XBEE_H
00011
00012 #include "define.h"
00013 #include "serialib.h"
00014 #include <string>
00015 #include <vector>
00016 #include <iomanip>
00017 #include <iostream>
00018
00022 class xbee{
00023
00024 public:
00025
00026     // Constructeur de la classe
00027     xbee();
00028
00029     // Destructeur de la classe
00030     ~xbee();
00031
00032     // Ouverture de la connexion série
00033     int openSerialConnection();
00034
00035     // Fermeture de la connexion série
00036     void closeSerialConnection();
00037
00038     // Entrée dans le mode de configuration AT
00039     bool enterATMode();
00040
00041     // Sortie du mode de configuration AT
00042     bool exitATMode();
00043
00044     // Vérification et correction de la configuration AT du module
00045     int checkATConfig();
00046
00047     // Lecture de la réponse du module à une commande AT
00048     bool readATResponse(const char *value = AT_EMPTY_VALUE);
00049
```

```
00050 // Envoi d'une commande AT
00051 bool sendATCommand(const char *command, const char *value, unsigned int mode);
00052
00053 // Ecriture de la configuration AT dans la mémoire flash du module
00054 bool writeATConfig();
00055
00056 // Création et envoi de la trame de message structurée
00057 void sendTrame(char ad_dest, char code_fct, char data[]);
00058
00059 private:
00060
00061 // Calcul du CRC16 Modbus de la trame
00062 int crc16(std::vector<char> trame);
00063
00064 // Retard de temporisation dans l'exécution du code
00065 void delay(unsigned int time);
00066 };
00067
00068 #endif
```

Index

- ~serialib
 - serialib, [6](#)
- ~xbee
 - xbee, [19](#)
- AT_EMPTY_VALUE
 - define.h, [24](#)
- AT_END_LINE
 - define.h, [24](#)
- AT_ENTER
 - define.h, [24](#)
- AT_ERROR_16BIT_SOURCE_ADDR
 - define.h, [24](#)
- AT_ERROR_AES
 - define.h, [24](#)
- AT_ERROR_AES_KEY
 - define.h, [25](#)
- AT_ERROR_API
 - define.h, [25](#)
- AT_ERROR_BAUDRATE
 - define.h, [25](#)
- AT_ERROR_CHANEL
 - define.h, [25](#)
- AT_ERROR_COORDINATOR
 - define.h, [25](#)
- AT_ERROR_ENTER
 - define.h, [25](#)
- AT_ERROR_EXIT
 - define.h, [26](#)
- AT_ERROR_LOW_DEST_ADDR
 - define.h, [26](#)
- AT_ERROR_PAN_ID
 - define.h, [26](#)
- AT_ERROR_PARITY
 - define.h, [26](#)
- AT_ERROR_SUCCESS
 - define.h, [26](#)
- AT_ERROR_VALUE
 - define.h, [26](#)
- AT_ERROR_WRITE_CONFIG
 - define.h, [27](#)
- AT_EXIT
 - define.h, [27](#)
- AT_GET_16BIT_SOURCE_ADDR
 - define.h, [27](#)
- AT_GET_AES
 - define.h, [27](#)
- AT_GET_AES_KEY
 - define.h, [27](#)
- AT_GET_API
 - define.h, [27](#)
- AT_GET_BAUDRATE
 - define.h, [28](#)
- AT_GET_CHANEL
 - define.h, [28](#)
- AT_GET_COORDINATOR
 - define.h, [28](#)
- AT_GET_LOW_DEST_ADDR
 - define.h, [28](#)
- AT_GET_PAN_ID
 - define.h, [28](#)
- AT_GET_PARITY
 - define.h, [28](#)
- AT_MODE_GET
 - define.h, [29](#)
- AT_MODE_SET
 - define.h, [29](#)
- AT_SUCCESS_VALUE
 - define.h, [29](#)
- AT_VALUE_16BIT_SOURCE_ADDR
 - define.h, [29](#)
- AT_VALUE_AES
 - define.h, [29](#)
- AT_VALUE_AES_KEY
 - define.h, [29](#)
- AT_VALUE_API
 - define.h, [30](#)
- AT_VALUE_BAUDRATE
 - define.h, [30](#)
- AT_VALUE_CHANEL
 - define.h, [30](#)
- AT_VALUE_COORDINATOR
 - define.h, [30](#)
- AT_VALUE_LOW_DEST_ADDR
 - define.h, [30](#)
- AT_VALUE_PAN_ID
 - define.h, [30](#)
- AT_VALUE_PARITY
 - define.h, [31](#)
- AT_WRITE_CONFIG
 - define.h, [31](#)
- available
 - serialib, [7](#)
- BAUDRATE
 - define.h, [31](#)
- BROADCAST
 - define.h, [31](#)
- checkATConfig

- xbee, [19](#)
- clearDTR
 - serialib, [7](#)
- clearRTS
 - serialib, [7](#)
- closeDevice
 - serialib, [7](#)
- closeSerialConnection
 - xbee, [19](#)
- code_fct
 - Trame, [17](#)
- CURRENT_ROBOT
 - define.h, [31](#)
- data
 - Trame, [17](#)
- DATABITS
 - define.h, [31](#)
- define.h, [23](#)
 - AT_EMPTY_VALUE, [24](#)
 - AT_END_LINE, [24](#)
 - AT_ENTER, [24](#)
 - AT_ERROR_16BIT_SOURCE_ADDR, [24](#)
 - AT_ERROR_AES, [24](#)
 - AT_ERROR_AES_KEY, [25](#)
 - AT_ERROR_API, [25](#)
 - AT_ERROR_BAUDRATE, [25](#)
 - AT_ERROR_CHANEL, [25](#)
 - AT_ERROR_COORDINATOR, [25](#)
 - AT_ERROR_ENTER, [25](#)
 - AT_ERROR_EXIT, [26](#)
 - AT_ERROR_LOW_DEST_ADDR, [26](#)
 - AT_ERROR_PAN_ID, [26](#)
 - AT_ERROR_PARITY, [26](#)
 - AT_ERROR_SUCCESS, [26](#)
 - AT_ERROR_VALUE, [26](#)
 - AT_ERROR_WRITE_CONFIG, [27](#)
 - AT_EXIT, [27](#)
 - AT_GET_16BIT_SOURCE_ADDR, [27](#)
 - AT_GET_AES, [27](#)
 - AT_GET_AES_KEY, [27](#)
 - AT_GET_API, [27](#)
 - AT_GET_BAUDRATE, [28](#)
 - AT_GET_CHANEL, [28](#)
 - AT_GET_COORDINATOR, [28](#)
 - AT_GET_LOW_DEST_ADDR, [28](#)
 - AT_GET_PAN_ID, [28](#)
 - AT_GET_PARITY, [28](#)
 - AT_MODE_GET, [29](#)
 - AT_MODE_SET, [29](#)
 - AT_SUCCESS_VALUE, [29](#)
 - AT_VALUE_16BIT_SOURCE_ADDR, [29](#)
 - AT_VALUE_AES, [29](#)
 - AT_VALUE_AES_KEY, [29](#)
 - AT_VALUE_API, [30](#)
 - AT_VALUE_BAUDRATE, [30](#)
 - AT_VALUE_CHANEL, [30](#)
 - AT_VALUE_COORDINATOR, [30](#)
 - AT_VALUE_LOW_DEST_ADDR, [30](#)
 - AT_VALUE_PAN_ID, [30](#)
 - AT_VALUE_PARITY, [31](#)
 - AT_WRITE_CONFIG, [31](#)
 - BAUDRATE, [31](#)
 - BROADCAST, [31](#)
 - CURRENT_ROBOT, [31](#)
 - DATABITS, [31](#)
 - END_SEQ, [32](#)
 - ERROR_SUCCESS, [32](#)
 - PARITY, [32](#)
 - ROBOT_01, [32](#)
 - ROBOT_02, [32](#)
 - SERIAL_PORT, [32](#)
 - START_SEQ, [33](#)
 - STOPBITS, [33](#)
 - TEST_ALIVE, [33](#)
- DTR
 - serialib, [7](#)
- elapsedTime_ms
 - timeOut, [16](#)
- END_SEQ
 - define.h, [32](#)
- enterATMode
 - xbee, [20](#)
- ERROR_SUCCESS
 - define.h, [32](#)
- exitATMode
 - xbee, [20](#)
- flushReceiver
 - serialib, [8](#)
- id_dest
 - Trame, [17](#)
- id_exp
 - Trame, [17](#)
- id_trame
 - Trame, [17](#)
- initTimer
 - timeOut, [16](#)
- isCTS
 - serialib, [8](#)
- isDCD
 - serialib, [8](#)
- isDeviceOpen
 - serialib, [8](#)
- isDSR
 - serialib, [9](#)
- isDTR
 - serialib, [9](#)
- isRI
 - serialib, [9](#)
- isRTS
 - serialib, [9](#)
- main
 - main.cpp, [34](#)
- main.cpp, [34](#)

- main, [34](#)
- openDevice
 - serialib, [10](#)
- openSerialConnection
 - xbee, [20](#)
- PARITY
 - define.h, [32](#)
- readATResponse
 - xbee, [20](#)
- readBytes
 - serialib, [11](#)
- readChar
 - serialib, [12](#)
- readString
 - serialib, [12](#)
- ROBOT_01
 - define.h, [32](#)
- ROBOT_02
 - define.h, [32](#)
- RTS
 - serialib, [13](#)
- sendATCommand
 - xbee, [21](#)
- sendTrame
 - xbee, [21](#)
- serial
 - xbee.cpp, [51](#)
- SERIAL_DATABITS_16
 - serialib.h, [47](#)
- SERIAL_DATABITS_5
 - serialib.h, [47](#)
- SERIAL_DATABITS_6
 - serialib.h, [47](#)
- SERIAL_DATABITS_7
 - serialib.h, [47](#)
- SERIAL_DATABITS_8
 - serialib.h, [47](#)
- SERIAL_PARITY_EVEN
 - serialib.h, [47](#)
- SERIAL_PARITY_MARK
 - serialib.h, [47](#)
- SERIAL_PARITY_NONE
 - serialib.h, [47](#)
- SERIAL_PARITY_ODD
 - serialib.h, [47](#)
- SERIAL_PARITY_SPACE
 - serialib.h, [47](#)
- SERIAL_PORT
 - define.h, [32](#)
- SERIAL_STOPBITS_1
 - serialib.h, [48](#)
- SERIAL_STOPBITS_1_5
 - serialib.h, [48](#)
- SERIAL_STOPBITS_2
 - serialib.h, [48](#)
- SerialDataBits
 - serialib.h, [47](#)
- serialib, [5](#)
 - ~serialib, [6](#)
 - available, [7](#)
 - clearDTR, [7](#)
 - clearRTS, [7](#)
 - closeDevice, [7](#)
 - DTR, [7](#)
 - flushReceiver, [8](#)
 - isCTS, [8](#)
 - isDCD, [8](#)
 - isDeviceOpen, [8](#)
 - isDSR, [9](#)
 - isDTR, [9](#)
 - isRI, [9](#)
 - isRTS, [9](#)
 - openDevice, [10](#)
 - readBytes, [11](#)
 - readChar, [12](#)
 - readString, [12](#)
 - RTS, [13](#)
 - serialib, [6](#)
 - setDTR, [13](#)
 - setRTS, [13](#)
 - writeBytes, [14](#)
 - writeChar, [14](#)
 - writeString, [15](#)
- serialib.cpp, [35](#)
- serialib.h, [45](#)
 - SERIAL_DATABITS_16, [47](#)
 - SERIAL_DATABITS_5, [47](#)
 - SERIAL_DATABITS_6, [47](#)
 - SERIAL_DATABITS_7, [47](#)
 - SERIAL_DATABITS_8, [47](#)
 - SERIAL_PARITY_EVEN, [47](#)
 - SERIAL_PARITY_MARK, [47](#)
 - SERIAL_PARITY_NONE, [47](#)
 - SERIAL_PARITY_ODD, [47](#)
 - SERIAL_PARITY_SPACE, [47](#)
 - SERIAL_STOPBITS_1, [48](#)
 - SERIAL_STOPBITS_1_5, [48](#)
 - SERIAL_STOPBITS_2, [48](#)
 - SerialDataBits, [47](#)
 - SerialParity, [47](#)
 - SerialStopBits, [47](#)
 - UNUSED, [46](#)
- SerialParity
 - serialib.h, [47](#)
- SerialStopBits
 - serialib.h, [47](#)
- setDTR
 - serialib, [13](#)
- setRTS
 - serialib, [13](#)
- size
 - Trame, [18](#)
- START_SEQ

- define.h, [33](#)
- STOPBITS
 - define.h, [33](#)
- TEST_ALIVE
 - define.h, [33](#)
- timeOut, [15](#)
 - elapsedTime_ms, [16](#)
 - initTimer, [16](#)
 - timeOut, [16](#)
- Trame, [16](#)
 - code_fct, [17](#)
 - data, [17](#)
 - id_dest, [17](#)
 - id_exp, [17](#)
 - id_trame, [17](#)
 - size, [18](#)
- UNUSED
 - serialib.h, [46](#)
- writeATConfig
 - xbee, [22](#)
- writeBytes
 - serialib, [14](#)
- writeChar
 - serialib, [14](#)
- writeString
 - serialib, [15](#)
- xbee, [18](#)
 - ~xbee, [19](#)
 - checkATConfig, [19](#)
 - closeSerialConnection, [19](#)
 - enterATMode, [20](#)
 - exitATMode, [20](#)
 - openSerialConnection, [20](#)
 - readATResponse, [20](#)
 - sendATCommand, [21](#)
 - sendTrame, [21](#)
 - writeATConfig, [22](#)
 - xbee, [19](#)
- xbee.cpp, [51](#)
 - serial, [51](#)
- xbee.h, [54](#)