Robotech 2021 - Ecran

Généré par Doxygen 1.9.3

1 Index hiérarchique 1

1.1 Hiérarchie des classes . . . 1

2 Index des classes 3

2.1 Liste des classes . . . 3

3 Index des fichiers 5

3.1 Liste des fichiers . . . 5

4 Documentation des classes 7

4.1 Référence de la classe Ecran . . . 7

4.1.1 Description détaillée . . . 7

4.1.2 Documentation des constructeurs et destructeur . . . 7

4.1.2.1 Ecran() . . . 8

4.1.2.2 ~Ecran() . . . 8

4.1.3 Documentation des fonctions membres . . . 8

4.1.3.1 closeSerialConnection() . . . 8

4.1.3.2 delay() . . . 8

4.1.3.3 openSerialConnection() . . . 9

4.1.3.4 sendTrame() . . . 9

4.2 Référence de la classe Log . . . 10

4.2.1 Description détaillée . . . 11

4.2.2 Documentation des constructeurs et destructeur . . . 11

4.2.2.1 Log() . . . 11

4.2.3 Documentation des fonctions membres . . . 11

4.2.3.1 save() . . . 11

4.2.4 Documentation des fonctions amies et associées . . . 11

4.2.4.1 operator<< [1/2] . . . 12

4.2.4.2 operator<< [2/2] . . . 12

4.2.5 Documentation des données membres . . . 12

4.2.5.1 name . . . 12

4.2.5.2 ss . . . 12

4.3 Référence de la structure Mendl . . . 13

4.3.1 Description détaillée . . . 13

4.4 Référence de la classe serialib . . . 13

4.4.1 Description détaillée . . . 14

4.4.2 Documentation des constructeurs et destructeur . . . 14

4.4.2.1 serialib() . . . 14

4.4.2.2 ~serialib() . . . 15

4.4.3 Documentation des fonctions membres . . . 15

4.4.3.1 available() . . . 15

4.4.3.2 clearDTR() . . . 15

4.4.3.3 clearRTS() . . . 16

4.4.3.4 closeDevice() . . . 16
4.4.3.5 DTR() . . . 16
4.4.3.6 flushReceiver() . . . 17
4.4.3.7 isCTS() . . . 17
4.4.3.8 isDCD() . . . 18
4.4.3.9 isDeviceOpen() . . . 18
4.4.3.10 isDSR() . . . 18
4.4.3.11 isDTR() . . . 19
4.4.3.12 isRI() . . . 19
4.4.3.13 isRTS() . . . 19
4.4.3.14 openDevice() . . . 20
4.4.3.15 readBytes() . . . 23
4.4.3.16 readChar() . . . 24
4.4.3.17 readString() . . . 25
4.4.3.18 readStringNoTimeOut() . . . 26
4.4.3.19 RTS() . . . 27
4.4.3.20 setDTR() . . . 28
4.4.3.21 setRTS() . . . 28
4.4.3.22 writeBytes() . . . 28
4.4.3.23 writeChar() . . . 29
4.4.3.24 writeString() . . . 30
4.4.4 Documentation des données membres . . . 30
4.4.4.1 currentStateDTR . . . 30
4.4.4.2 currentStateRTS . . . 31
4.5 Référence de la classe timeOut . . . 31
4.5.1 Description détaillée . . . 31
4.5.2 Documentation des constructeurs et destructeur . . . 31
4.5.2.1 timeOut() . . . 31
4.5.3 Documentation des fonctions membres . . . 32
4.5.3.1 elapsedTime_ms() . . . 32
4.5.3.2 initTimer() . . . 32
4.5.4 Documentation des données membres . . . 33
4.5.4.1 previousTime . . . 33
4.6 Référence de la structure Trame_t . . . 33
4.6.1 Description détaillée . . . 33
4.6.2 Documentation des données membres . . . 33
4.6.2.1 categorie . . . 33
4.6.2.2 mode . . . 34
4.6.2.3 sous_categorie . . . 34
4.6.2.4 taille . . . 34
4.6.2.5 valeur . . . 34

5 Documentation des fichiers **35**
5.1 Référence du fichier ecran_define.h . . . 35
5.1.1 Documentation des macros . . . 36
5.1.1.1 EC_BAUDRATE_DEFAULT . . . 36
5.1.1.2 EC_DATABITS_DEFAULT . . . 36
5.1.1.3 EC_FIN_TRAME_1 . . . 36
5.1.1.4 EC_FIN_TRAME_2 . . . 36
5.1.1.5 EC_PARITY_DEFAULT . . . 36
5.1.1.6 EC_SER_E_CONFIG . . . 37
5.1.1.7 EC_SER_E_NOT_FOUND . . . 37
5.1.1.8 EC_SER_E_OPEN . . . 37
5.1.1.9 EC_SER_E_PARAM . . . 37
5.1.1.10 EC_SER_E_SUCCESS . . . 37
5.1.1.11 EC_SER_E_TIMOUT . . . 37
5.1.1.12 EC_SER_E_UKN_BAUDRATE . . . 38
5.1.1.13 EC_SER_E_UKN_DATABITS . . . 38
5.1.1.14 EC_SER_E_UKN_PARITY . . . 38
5.1.1.15 EC_SER_E_UKN_STOPBITS . . . 38
5.1.1.16 EC_SERIAL_PORT_DEFAULT . . . 38
5.1.1.17 EC_STOPBITS_DEFAULT . . . 38
5.1.1.18 EC_TRAME_E_CAT . . . 39
5.1.1.19 EC_TRAME_E_DATA . . . 39
5.1.1.20 EC_TRAME_E_END . . . 39
5.1.1.21 EC_TRAME_E_MODE . . . 39
5.1.1.22 EC_TRAME_E_SIZE . . . 39
5.1.1.23 EC_TRAME_E_SOUS_CAT . . . 39
5.1.1.24 EC_TRAME_E_SUCCESS . . . 39
5.1.2 Documentation du type de l'énumération . . . 39
5.1.2.1 Categorie_Ecran_t . . . 39
5.1.2.2 Mode_Ecran_t . . . 40
5.1.2.3 Sous_Categorie_Ecran_t . . . 40
5.2 ecran_define.h . . . 41
5.3 Référence du fichier ecranlib.cpp . . . 42
5.3.1 Documentation des variables . . . 42
5.3.1.1 logEcran . . . 42
5.3.1.2 serial . . . 42
5.4 ecranlib.cpp . . . 43
5.5 Référence du fichier ecranlib.h . . . 43
5.5.1 Description détaillée . . . 44
5.6 ecranlib.h . . . 44
5.7 Référence du fichier loglib.cpp . . . 45
5.7.1 Documentation des fonctions . . . 45

5.7.1.1 operator<<() 45
5.7.1.2 stringToChar() 46
5.8 loglib.cpp 46
5.9 Référence du fichier loglib.h 46
5.9.1 Documentation des fonctions 47
5.9.1.1 operator<<() 47
5.9.1.2 stringToChar() 47
5.9.2 Documentation des variables 47
5.9.2.1 mendl 47
5.10 loglib.h 48
5.11 Référence du fichier main.cpp 48
5.11.1 Documentation des fonctions 48
5.11.1.1 main() 49
5.12 main.cpp 49
5.13 Référence du fichier serialib.cpp 49
5.13.1 Description détaillée 50
5.14 serialib.cpp 50
5.15 Référence du fichier serialib.h 60
5.15.1 Description détaillée 60
5.15.2 Documentation des macros 61
5.15.2.1 UNUSED 61
5.15.3 Documentation du type de l'énumération 61
5.15.3.1 SerialDataBits 61
5.15.3.2 SerialParity 61
5.15.3.3 SerialStopBits 62
5.16 serialib.h 62

Index **67**

Chapitre 1

Index hiérarchique

1.1 Hiérarchie des classes

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

Ecran 7
Mendl 13
std::ostream
    Log 10
serialib 13
timeOut 31
Trame_t 33

Chapitre 2

Index des classes

2.1 Liste des classes

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

Ecran
: Cette classe est utilisée pour la communication entre un ecran STM32F7G et une RaspberryPi 7

Log 10

Mendl 13

serialib
: This class is used for communication over a serial device 13

timeOut
: This class can manage a timer which is used as a timeout 31

Trame_t 33

Chapitre 3

Index des fichiers

3.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

ecran_define.h 35
ecranlib.cpp 42
ecranlib.h
    Fichier d'en-tête de la classe Ecran. Cette classe est utilisée afin de communique en UART avec l'écran sur la RaspberryPi 43
loglib.cpp 45
loglib.h 46
main.cpp 48
serialib.cpp
    Source file of the class serialib. This class is used for communication over a serial device . . . 49
serialib.h
    Header file of the class serialib. This class is used for communication over a serial device . . . 60

Chapitre 4

Documentation des classes

4.1 Référence de la classe Ecran

Cette classe est utilisée pour la communication entre un ecran STM32F7G et une RaspberryPi.

`#include <ecranlib.h>`

Fonctions membres publiques

- Ecran ()
  Constructeur de la classe Ecran.
- ~Ecran ()
  Destructeur de la classe Ecran.
- int openSerialConnection ()
  Nettoyage du buffer et ouverture de la connexion UART entre la RaspberryPi et l'écran.
- void closeSerialConnection ()
  Nettoyage du buffer et fermeture de la connexion UART entre la RaspberryPi et l'écran.
- void sendTrame (Trame_t trame)
  Fonction permettant d'envoyer une trame de message structurée via UART à l'écran.

Fonctions membres privées

- void delay (unsigned int time)
  Fonction permettant de retarder l'exécution du code.

4.1.1 Description détaillée

Cette classe est utilisée pour la communication entre un ecran STM32F7G et une RaspberryPi.

Définition à la ligne 36 du fichier ecranlib.h.

4.1.2 Documentation des constructeurs et destructeur

4.1.2.1 Ecran()

```
Ecran::Ecran ( )
```

Constructeur de la classe Ecran.

Définition à la ligne 21 du fichier ecranlib.cpp.
```
00021 { }
```

4.1.2.2 ~Ecran()

```
Ecran::~Ecran ( )
```

Destructeur de la classe Ecran.

Définition à la ligne 26 du fichier ecranlib.cpp.
```
00026 { }
```

4.1.3 Documentation des fonctions membres

4.1.3.1 closeSerialConnection()

```
void Ecran::closeSerialConnection ( )
```

Nettoyage du buffer et fermeture de la connexion UART entre la RaspberryPi et l'écran.

Définition à la ligne 61 du fichier ecranlib.cpp.
```
00061                                   {
00062     serial.flushReceiver();
00063     logEcran « "(serial) buffer Rx nettoyé avec succès" « mendl;
00064
00065     serial.closeDevice();
00066     logEcran « "(serial) connexion série fermée avec succès" « mendl;
00067 }
```

4.1.3.2 delay()

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

Fonction permettant de retarder l'exécution du code.

Paramètres

time	: temps du retard en secondes

Définition à la ligne 114 du fichier ecranlib.cpp.
```
00114 { std::this_thread::sleep_for(std::chrono::milliseconds(time*1000)); }
```

4.1.3.3 openSerialConnection()

```
int Ecran::openSerialConnection ( )
```

Nettoyage du buffer et ouverture de la connexion UART entre la RaspberryPi et l'écran.

Renvoie

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

Définition à la ligne 45 du fichier ecranlib.cpp.
```
00045                                {
00046     int errorOpening;
00047     errorOpening = serial.openDevice(EC_SERIAL_PORT_DEFAULT, EC_BAUDRATE_DEFAULT, EC_DATABITS_DEFAULT,
      EC_PARITY_DEFAULT, EC_STOPBITS_DEFAULT);
00048
00049     if (errorOpening != EC_SER_E_SUCCESS)
00050         logEcran << "(serial) /!\\ erreur " << errorOpening << " : impossible d'ouvrir le port " <<
      EC_SERIAL_PORT_DEFAULT  << " - baudrate : " << EC_BAUDRATE_DEFAULT << " - parité : " << EC_PARITY_DEFAULT
      << mendl;
00051     else{
00052         logEcran << "(serial) connexion ouverte avec succès sur le port " << EC_SERIAL_PORT_DEFAULT << "
      - baudrate : " << EC_BAUDRATE_DEFAULT << " - parité : " << EC_PARITY_DEFAULT << mendl;
00053     }
00054
00055     return errorOpening;
00056 }
```

4.1.3.4 sendTrame()

```
int Ecran::sendTrame (
            Trame_t trame_recue )
```

Fonction permettant d'envoyer une trame de message structurée via UART à l'écran.

Paramètres

mode	: le mode de configuration de l'écran
categorie	: la catégorie correspondant au mode choisi
sous_categorie	: la sous-catégorie (spécification) en fonction de la catégorie choisie
value	: les data à transmettre

Renvoie

200 succès
-201 taille de la trame incorrecte
-202 data de la trame incorrecte
-203 fin de trame incorrecte
-204 mode choisi pour la trame incorrect ou inconnu
-205 catégorie choisie pour la trame incorrecte ou inconnue
-206 sous-catégorie choisie pour la trame incorrecte ou inconnue

Définition à la ligne 83 du fichier ecranlib.cpp.

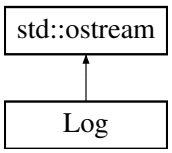
```
00083                                       {
00084     int length_trame = 0;
00085
00086     if(trame_recue.taille >= 5)
00087         length_trame = trame_recue.taille;
00088     else
00089         return EC_TRAME_E_SIZE;
00090
00091     uint8_t trame[length_trame];
00092
00093     trame[0] = trame_recue.mode;
00094     trame[1] = trame_recue.categorie;
00095     trame[2] = trame_recue.sous_categorie;
00096
00097     for(size_t i = 0; i < sizeof(trame_recue.valeur)/sizeof(trame_recue.valeur[0]); i++){
00098         trame[i+3] = trame_recue.valeur[i];
00099     }
00100
00101     trame[length_trame-2] = EC_FIN_TRAME_1;
00102     trame[length_trame-1] = EC_FIN_TRAME_2;
00103
00104     serial.writeBytes(trame, length_trame);
00105
00106     logEcran « "(sendMsg) envoi de la trame effectué avec succès" « mendl;
00107     return EC_TRAME_E_SUCCESS;
00108 }
```

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

4.2 Référence de la classe Log

`#include <loglib.h>`

Graphe d'héritage de Log:



Fonctions membres publiques

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

Attributs publics

— std::string name

Attributs privés

— std::stringstream ss

Amis

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

4.2.1 Description détaillée

Définition à la ligne 18 du fichier loglib.h.

4.2.2 Documentation des constructeurs et destructeur

4.2.2.1 Log()

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

Définition à la ligne 14 du fichier loglib.cpp.

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

4.2.3 Documentation des fonctions membres

4.2.3.1 save()

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

4.2.4 Documentation des fonctions amies et associées

4.2.4.1 operator<< [1/2]

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

Définition à la ligne 20 du fichier loglib.cpp.

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

4.2.4.2 operator<< [2/2]

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

Définition à la ligne 34 du fichier loglib.h.

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

4.2.5 Documentation des données membres

4.2.5.1 name

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

Définition à la ligne 23 du fichier loglib.h.

4.2.5.2 ss

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

Définition à la ligne 20 du fichier loglib.h.

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

- loglib.h
- loglib.cpp

4.3 Référence de la structure Mendl

`#include <loglib.h>`

4.3.1 Description détaillée

Définition à la ligne 10 du fichier loglib.h.

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

— loglib.h

4.4 Référence de la classe serialib

This class is used for communication over a serial device.

`#include <serialib.h>`

Fonctions membres publiques

— serialib ()
Constructor of the class serialib.
— ~serialib ()
Destructor of the class serialib. It close the connection.
— int openDevice (const char *Device, const unsigned int Bauds, SerialDataBits Databits=SERIAL_DATABITS_8, SerialParity Parity=SERIAL_PARITY_NONE, SerialStopBits Stopbits=SERIAL_STOPBITS_1)
Open the serial port.
— bool isDeviceOpen ()
— void closeDevice ()
Close the connection with the current device.
— char writeChar (char)
Write a char on the current serial port.
— char readChar (char *pByte, const unsigned int timeOut_ms=0)
Wait for a byte from the serial device and return the data read.
— char writeString (const char *String)
Write a string on the current serial port.
— int readString (char *receivedString, char finalChar, unsigned int maxNbBytes, const unsigned int timeOut↩ _ms=0)
Read a string from the serial device (with timeout)
— char writeBytes (const void *Buffer, const unsigned int NbBytes)
Write an array of data on the current serial port.
— int readBytes (void *buffer, unsigned int maxNbBytes, const unsigned int timeOut_ms=0, unsigned int 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.

Fonctions membres privées

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

Attributs privés

- bool currentStateRTS
- bool currentStateDTR

4.4.1 Description détaillée

This class is used for communication over a serial device.

Définition à la ligne 92 du fichier serialib.h.

4.4.2 Documentation des constructeurs et destructeur

4.4.2.1 serialib()

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

Constructor of the class serialib.

Définition à la ligne 30 du fichier serialib.cpp.

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

4.4.2.2 ~serialib()

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

Destructor of the class serialib. It close the connection.

Définition à la ligne 48 du fichier serialib.cpp.

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

4.4.3 Documentation des fonctions membres

4.4.3.1 available()

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

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

Renvoie

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

Définition à la ligne 702 du fichier serialib.cpp.

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

4.4.3.2 clearDTR()

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

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

Renvoie

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

Définition à la ligne 777 du fichier serialib.cpp.

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

4.4.3.3 clearRTS()

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

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

Renvoie

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

Définition à la ligne 847 du fichier serialib.cpp.

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

4.4.3.4 closeDevice()

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

Close the connection with the current device.

Définition à la ligne 317 du fichier serialib.cpp.

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

4.4.3.5 DTR()

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

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

Paramètres

status	= true set DTR status = false unset DTR

Renvoie

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

Définition à la ligne 737 du fichier serialib.cpp.

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

4.4.3.6 flushReceiver()

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

Empty receiver buffer.

Renvoie

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

Définition à la ligne 683 du fichier serialib.cpp.

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

4.4.3.7 isCTS()

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

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

Renvoie

Return true if CTS is set otherwise false

Définition à la ligne 872 du fichier serialib.cpp.

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

4.4.3.8 isDCD()

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

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

Renvoie

true if DCD is set
false otherwise

Définition à la ligne 920 du fichier serialib.cpp.

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

4.4.3.9 isDeviceOpen()

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

Définition à la ligne 304 du fichier serialib.cpp.

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

4.4.3.10 isDSR()

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

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

Renvoie

Return true if DTR is set otherwise false

Définition à la ligne 894 du fichier serialib.cpp.

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

4.4.3.11 isDTR()

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

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

Renvoie

Return true if CTS is set otherwise false

Définition à la ligne 963 du fichier serialib.cpp.

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

4.4.3.12 isRI()

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

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

Renvoie

Return true if RING is set otherwise false

Définition à la ligne 941 du fichier serialib.cpp.

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

4.4.3.13 isRTS()

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

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

Renvoie

Return true if RTS is set otherwise false

Définition à la ligne 984 du fichier serialib.cpp.

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

4.4.3.14 openDevice()

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

Open the serial port.

Paramètres

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

Renvoie

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

Définition à la ligne 129 du fichier serialib.cpp.

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

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

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

4.4.3.15 readBytes()

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

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

Paramètres

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

Renvoie

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

Définition à la ligne 615 du fichier serialib.cpp.

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

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

4.4.3.16 readChar()

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

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

Paramètres

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

Renvoie

1 success

0 Timeout reached

-1 error while setting the Timeout

-2 error while reading the byte

Définition à la ligne 441 du fichier serialib.cpp.

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

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

4.4.3.17 readString()

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

Read a string from the serial device (with timeout)

Paramètres

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

Renvoie

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

Définition à la ligne 541 du fichier serialib.cpp.

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

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

4.4.3.18 readStringNoTimeOut()

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

Read a string from the serial device (without TimeOut)

Paramètres

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

Renvoie

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

Définition à la ligne 492 du fichier serialib.cpp.

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

4.4.3.19 RTS()

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

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

Paramètres

status	= true set DTR status = false unset DTR

Renvoie

false if the function fails

true if the function succeeds

Définition à la ligne 805 du fichier serialib.cpp.

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

4.4.3.20 setDTR()

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

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

Renvoie

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

Définition à la ligne 754 du fichier serialib.cpp.

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

4.4.3.21 setRTS()

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

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

Renvoie

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

Définition à la ligne 822 du fichier serialib.cpp.

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

4.4.3.22 writeBytes()

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

Write an array of data on the current serial port.

Paramètres

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

Renvoie

1 success

-1 error while writting data

Définition à la ligne 409 du fichier serialib.cpp.

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

4.4.3.23 writeChar()

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

Write a char on the current serial port.

Paramètres

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

Renvoie

1 success

-1 error while writting data

Définition à la ligne 343 du fichier serialib.cpp.

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

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

4.4.3.24 writeString()

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

Write a string on the current serial port.

Paramètres

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

Renvoie

1 success

-1 error while writting data

Définition à la ligne 375 du fichier serialib.cpp.

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

4.4.4 Documentation des données membres

4.4.4.1 currentStateDTR

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

Définition à la ligne 221 du fichier serialib.h.

4.4.4.2 currentStateRTS

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

Définition à la ligne 220 du fichier serialib.h.

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

4.5 Référence de la classe timeOut

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

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

Fonctions membres publiques

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

Attributs privés

- struct timeval previousTime

4.5.1 Description détaillée

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

Définition à la ligne 245 du fichier serialib.h.

4.5.2 Documentation des constructeurs et destructeur

4.5.2.1 timeOut()

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

Constructor of the class timeOut.

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

4.5.3 Documentation des fonctions membres

4.5.3.1 elapsedTime_ms()

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

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

Renvoie

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

Définition à la ligne 1039 du fichier serialib.cpp.

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

4.5.3.2 initTimer()

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

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

Définition à la ligne 1019 du fichier serialib.cpp.

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

4.5.4 Documentation des données membres

4.5.4.1 previousTime

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

Définition à la ligne 265 du fichier serialib.h.

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

- serialib.h
- serialib.cpp

4.6 Référence de la structure Trame_t

```
#include <ecranlib.h>
```

Attributs publics

- Mode_Ecran_t mode
- Categorie_Ecran_t categorie
- Sous_Categorie_Ecran_t sous_categorie
- uint8_t * valeur
- int taille

4.6.1 Description détaillée

Définition à la ligne 25 du fichier ecranlib.h.

4.6.2 Documentation des données membres

4.6.2.1 categorie

```
Categorie_Ecran_t Trame_t::categorie
```

Définition à la ligne 27 du fichier ecranlib.h.

4.6.2.2 mode

```
Mode_Ecran_t Trame_t::mode
```

Définition à la ligne 26 du fichier ecranlib.h.

4.6.2.3 sous_categorie

```
Sous_Categorie_Ecran_t Trame_t::sous_categorie
```

Définition à la ligne 28 du fichier ecranlib.h.

4.6.2.4 taille

```
int Trame_t::taille
```

Définition à la ligne 30 du fichier ecranlib.h.

4.6.2.5 valeur

```
uint8_t* Trame_t::valeur
```

Définition à la ligne 29 du fichier ecranlib.h.

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

— ecranlib.h

Chapitre 5

Documentation des fichiers

5.1 Référence du fichier ecran_define.h

Macros

- #define EC_SERIAL_PORT_DEFAULT "COM8"
- #define EC_BAUDRATE_DEFAULT 115200
- #define EC_DATABITS_DEFAULT SERIAL_DATABITS_8
- #define EC_PARITY_DEFAULT SERIAL_PARITY_NONE
- #define EC_STOPBITS_DEFAULT SERIAL_STOPBITS_1
- #define EC_FIN_TRAME_1 0x0A
- #define EC_FIN_TRAME_2 0x0D
- #define EC_TRAME_E_SUCCESS 200
- #define EC_TRAME_E_SIZE -201
- #define EC_TRAME_E_DATA -202
- #define EC_TRAME_E_END -203
- #define EC_TRAME_E_MODE -204
- #define EC_TRAME_E_CAT -205
- #define EC_TRAME_E_SOUS_CAT -206
- #define EC_SER_E_SUCCESS 500
- #define EC_SER_E_NOT_FOUND -501
- #define EC_SER_E_OPEN -502
- #define EC_SER_E_PARAM -503
- #define EC_SER_E_UKN_BAUDRATE -504
- #define EC_SER_E_CONFIG -505
- #define EC_SER_E_TIMOUT -506
- #define EC_SER_E_UKN_DATABITS -507
- #define EC_SER_E_UKN_STOPBITS -508
- #define EC_SER_E_UKN_PARITY -509

Énumérations

- enum Mode_Ecran_t { EC_MODE_COMPETITION = 0x43 , EC_MODE_MAINTENANCE = 0x4D }
- enum Categorie_Ecran_t { EC_CAT_TOF = 0x54 , EC_CAT_INIT = 0x49 , EC_CAT_SCORE = 0x53 }
- enum Sous_Categorie_Ecran_t {
  EC_SOUS_CAT_SCORE_PET_ROB = 0x50 , EC_SOUS_CAT_SCORE_GRD_ROB = 0x47 , EC_SOUS_CAT_SCORE_TOTAL = 0x54 , EC_SOUS_CAT_ODO = 0x4F ,
  EC_SOUS_CAT_BASE = 0x42 , EC_SOUS_CAT_XBEE = 0x58 , EC_SOUS_CAT_CAN = 0x43 ,
  EC_SOUS_CAT_TOF_0_AVG = 0x30 ,
  EC_SOUS_CAT_TOF_1_MG = 0x31 , EC_SOUS_CAT_TOF_2_ARG = 0x32 , EC_SOUS_CAT_TOF_3_B = 0x33 , EC_SOUS_CAT_TOF_4_ARD = 0x34 ,
  EC_SOUS_CAT_TOF_5_MD = 0x35 , EC_SOUS_CAT_TOF_6_AVD = 0x36 , EC_SOUS_CAT_TOF_7_HD = 0x37 , EC_SOUS_CAT_TOF_8_HG = 0x38 }

5.1.1 Documentation des macros

5.1.1.1 EC_BAUDRATE_DEFAULT

```
#define EC_BAUDRATE_DEFAULT 115200
```

Définition à la ligne 13 du fichier ecran_define.h.

5.1.1.2 EC_DATABITS_DEFAULT

```
#define EC_DATABITS_DEFAULT SERIAL_DATABITS_8
```

Définition à la ligne 14 du fichier ecran_define.h.

5.1.1.3 EC_FIN_TRAME_1

```
#define EC_FIN_TRAME_1 0x0A
```

Définition à la ligne 57 du fichier ecran_define.h.

5.1.1.4 EC_FIN_TRAME_2

```
#define EC_FIN_TRAME_2 0x0D
```

Définition à la ligne 58 du fichier ecran_define.h.

5.1.1.5 EC_PARITY_DEFAULT

```
#define EC_PARITY_DEFAULT SERIAL_PARITY_NONE
```

Définition à la ligne 15 du fichier ecran_define.h.

5.1.1.6 EC_SER_E_CONFIG

```
#define EC_SER_E_CONFIG -505
```

Définition à la ligne 75 du fichier ecran_define.h.

5.1.1.7 EC_SER_E_NOT_FOUND

```
#define EC_SER_E_NOT_FOUND -501
```

Définition à la ligne 71 du fichier ecran_define.h.

5.1.1.8 EC_SER_E_OPEN

```
#define EC_SER_E_OPEN -502
```

Définition à la ligne 72 du fichier ecran_define.h.

5.1.1.9 EC_SER_E_PARAM

```
#define EC_SER_E_PARAM -503
```

Définition à la ligne 73 du fichier ecran_define.h.

5.1.1.10 EC_SER_E_SUCCESS

```
#define EC_SER_E_SUCCESS 500
```

Définition à la ligne 70 du fichier ecran_define.h.

5.1.1.11 EC_SER_E_TIMOUT

```
#define EC_SER_E_TIMOUT -506
```

Définition à la ligne 76 du fichier ecran_define.h.

5.1.1.12 EC_SER_E_UKN_BAUDRATE

```
#define EC_SER_E_UKN_BAUDRATE -504
```

Définition à la ligne 74 du fichier ecran_define.h.

5.1.1.13 EC_SER_E_UKN_DATABITS

```
#define EC_SER_E_UKN_DATABITS -507
```

Définition à la ligne 77 du fichier ecran_define.h.

5.1.1.14 EC_SER_E_UKN_PARITY

```
#define EC_SER_E_UKN_PARITY -509
```

Définition à la ligne 79 du fichier ecran_define.h.

5.1.1.15 EC_SER_E_UKN_STOPBITS

```
#define EC_SER_E_UKN_STOPBITS -508
```

Définition à la ligne 78 du fichier ecran_define.h.

5.1.1.16 EC_SERIAL_PORT_DEFAULT

```
#define EC_SERIAL_PORT_DEFAULT "COM8"
```

Définition à la ligne 12 du fichier ecran_define.h.

5.1.1.17 EC_STOPBITS_DEFAULT

```
#define EC_STOPBITS_DEFAULT SERIAL_STOPBITS_1
```

Définition à la ligne 16 du fichier ecran_define.h.

5.1.1.18 EC_TRAME_E_CAT

```
#define EC_TRAME_E_CAT -205
```

Définition à la ligne 66 du fichier ecran_define.h.

5.1.1.19 EC_TRAME_E_DATA

```
#define EC_TRAME_E_DATA -202
```

Définition à la ligne 63 du fichier ecran_define.h.

5.1.1.20 EC_TRAME_E_END

```
#define EC_TRAME_E_END -203
```

Définition à la ligne 64 du fichier ecran_define.h.

5.1.1.21 EC_TRAME_E_MODE

```
#define EC_TRAME_E_MODE -204
```

Définition à la ligne 65 du fichier ecran_define.h.

5.1.1.22 EC_TRAME_E_SIZE

```
#define EC_TRAME_E_SIZE -201
```

Définition à la ligne 62 du fichier ecran_define.h.

5.1.1.23 EC_TRAME_E_SOUS_CAT

```
#define EC_TRAME_E_SOUS_CAT -206
```

Définition à la ligne 67 du fichier ecran_define.h.

5.1.1.24 EC_TRAME_E_SUCCESS

```
#define EC_TRAME_E_SUCCESS 200
```

Définition à la ligne 61 du fichier ecran_define.h.

5.1.2 Documentation du type de l'énumération

5.1.2.1 Categorie_Ecran_t

```
enum Categorie_Ecran_t
```

Valeurs énumérées

EC_CAT_TOF	
EC_CAT_INIT	
EC_CAT_SCORE	

Définition à la ligne 25 du fichier ecran_define.h.

```
00025             {
00026     EC_CAT_TOF = 0x54,
00027     EC_CAT_INIT = 0x49,
00028     EC_CAT_SCORE = 0x53
00029 } Categorie_Ecran_t;
```

5.1.2.2 Mode_Ecran_t

```
enum Mode_Ecran_t
```

Valeurs énumérées

EC_MODE_COMPETITION	
EC_MODE_MAINTENANCE	

Définition à la ligne 19 du fichier ecran_define.h.

```
00019             {
00020     EC_MODE_COMPETITION = 0x43,
00021     EC_MODE_MAINTENANCE = 0x4D
00022 } Mode_Ecran_t;
```

5.1.2.3 Sous_Categorie_Ecran_t

```
enum Sous_Categorie_Ecran_t
```

Valeurs énumérées

EC_SOUS_CAT_SCORE_PET_ROB	
EC_SOUS_CAT_SCORE_GRD_ROB	
EC_SOUS_CAT_SCORE_TOTAL	
EC_SOUS_CAT_ODO	
EC_SOUS_CAT_BASE	
EC_SOUS_CAT_XBEE	
EC_SOUS_CAT_CAN	
EC_SOUS_CAT_TOF_0_AVG	
EC_SOUS_CAT_TOF_1_MG	
EC_SOUS_CAT_TOF_2_ARG	
EC_SOUS_CAT_TOF_3_B	
EC_SOUS_CAT_TOF_4_ARD	
EC_SOUS_CAT_TOF_5_MD	
EC_SOUS_CAT_TOF_6_AVD	
EC_SOUS_CAT_TOF_7_HD	
EC_SOUS_CAT_TOF_8_HG	

Définition à la ligne 32 du fichier ecran_define.h.

```
00032             {
00033     // Score
00034     EC_SOUS_CAT_SCORE_PET_ROB = 0x50,
00035     EC_SOUS_CAT_SCORE_GRD_ROB = 0x47,
00036     EC_SOUS_CAT_SCORE_TOTAL = 0x54,
00037
00038     // Initialisation
00039     EC_SOUS_CAT_ODO = 0x4F,
00040     EC_SOUS_CAT_BASE = 0x42,
00041     EC_SOUS_CAT_XBEE = 0x58,
00042     EC_SOUS_CAT_CAN = 0x43,
00043
00044     // Maintenance ToF
00045     EC_SOUS_CAT_TOF_0_AVG = 0x30,
00046     EC_SOUS_CAT_TOF_1_MG = 0x31,
00047     EC_SOUS_CAT_TOF_2_ARG = 0x32,
00048     EC_SOUS_CAT_TOF_3_B = 0x33,
00049     EC_SOUS_CAT_TOF_4_ARD = 0x34,
00050     EC_SOUS_CAT_TOF_5_MD = 0x35,
00051     EC_SOUS_CAT_TOF_6_AVD = 0x36,
00052     EC_SOUS_CAT_TOF_7_HD = 0x37,
00053     EC_SOUS_CAT_TOF_8_HG = 0x38
00054 } Sous_Categorie_Ecran_t;
```

5.2 ecran_define.h

Aller à la documentation de ce fichier.

```
00001
00008 #ifndef DEFINE_ECRAN_H
00009 #define DEFINE_ECRAN_H
00010
00011 // Paramètres du port série
00012 #define EC_SERIAL_PORT_DEFAULT "COM8"
00013 #define EC_BAUDRATE_DEFAULT 115200
00014 #define EC_DATABITS_DEFAULT SERIAL_DATABITS_8
00015 #define EC_PARITY_DEFAULT SERIAL_PARITY_NONE
00016 #define EC_STOPBITS_DEFAULT SERIAL_STOPBITS_1
00017
00018 // Modes de l'écran
00019 typedef enum {
00020     EC_MODE_COMPETITION = 0x43,
00021     EC_MODE_MAINTENANCE = 0x4D
00022 } Mode_Ecran_t;
00023
00024 // Codes catégories
00025 typedef enum {
00026     EC_CAT_TOF = 0x54,
00027     EC_CAT_INIT = 0x49,
00028     EC_CAT_SCORE = 0x53
00029 } Categorie_Ecran_t;
00030
00031 // Codes sous catégories
00032 typedef enum {
00033     // Score
00034     EC_SOUS_CAT_SCORE_PET_ROB = 0x50,
00035     EC_SOUS_CAT_SCORE_GRD_ROB = 0x47,
00036     EC_SOUS_CAT_SCORE_TOTAL = 0x54,
00037
00038     // Initialisation
00039     EC_SOUS_CAT_ODO = 0x4F,
00040     EC_SOUS_CAT_BASE = 0x42,
00041     EC_SOUS_CAT_XBEE = 0x58,
00042     EC_SOUS_CAT_CAN = 0x43,
00043
00044     // Maintenance ToF
00045     EC_SOUS_CAT_TOF_0_AVG = 0x30,
00046     EC_SOUS_CAT_TOF_1_MG = 0x31,
00047     EC_SOUS_CAT_TOF_2_ARG = 0x32,
00048     EC_SOUS_CAT_TOF_3_B = 0x33,
00049     EC_SOUS_CAT_TOF_4_ARD = 0x34,
00050     EC_SOUS_CAT_TOF_5_MD = 0x35,
00051     EC_SOUS_CAT_TOF_6_AVD = 0x36,
00052     EC_SOUS_CAT_TOF_7_HD = 0x37,
00053     EC_SOUS_CAT_TOF_8_HG = 0x38
00054 } Sous_Categorie_Ecran_t;
00055
00056 // Codes fin de trame
00057 #define EC_FIN_TRAME_1 0x0A
00058 #define EC_FIN_TRAME_2 0x0D
00059
```

```
00060 // Codes erreurs traitement de trame
00061 #define EC_TRAME_E_SUCCESS 200
00062 #define EC_TRAME_E_SIZE -201
00063 #define EC_TRAME_E_DATA -202
00064 #define EC_TRAME_E_END -203
00065 #define EC_TRAME_E_MODE -204
00066 #define EC_TRAME_E_CAT -205
00067 #define EC_TRAME_E_SOUS_CAT -206
00068
00069 // Codes d'erreurs ouverture connexion série
00070 #define EC_SER_E_SUCCESS 500
00071 #define EC_SER_E_NOT_FOUND -501
00072 #define EC_SER_E_OPEN -502
00073 #define EC_SER_E_PARAM -503
00074 #define EC_SER_E_UKN_BAUDRATE -504
00075 #define EC_SER_E_CONFIG -505
00076 #define EC_SER_E_TIMOUT -506
00077 #define EC_SER_E_UKN_DATABITS -507
00078 #define EC_SER_E_UKN_STOPBITS -508
00079 #define EC_SER_E_UKN_PARITY -509
00080
00081 #endif
```

5.3 Référence du fichier ecranlib.cpp

```
#include "ecranlib.h"
```

Variables

— serialib serial
— Log logEcran ("ecran")

5.3.1 Documentation des variables

5.3.1.1 logEcran

```
Log logEcran("ecran") (
             "ecran"  )
```

5.3.1.2 serial

```
serialib serial
```

Définition à la ligne 12 du fichier ecranlib.cpp.

5.4 ecranlib.cpp

Aller à la documentation de ce fichier.

```
00001
00008 #include "ecranlib.h"
00009
00010 using namespace std;
00011
00012 serialib serial;
00013 Log logEcran("ecran");
00014
00015 //_____________________________________
00016 // ::: Constructeurs et destructeurs :::
00017
00021 Ecran::Ecran(){ }
00022
00026 Ecran::~Ecran(){ }
00027
00028
00029 //_____________________________________
00030 // ::: Configuration and initialisation :::
00031
00045 int Ecran::openSerialConnection(){
00046     int errorOpening;
00047     errorOpening = serial.openDevice(EC_SERIAL_PORT_DEFAULT, EC_BAUDRATE_DEFAULT, EC_DATABITS_DEFAULT,
      EC_PARITY_DEFAULT, EC_STOPBITS_DEFAULT);
00048
00049     if (errorOpening != EC_SER_E_SUCCESS)
00050         logEcran « "(serial) /!\\ erreur " « errorOpening « " : impossible d'ouvrir le port " «
      EC_SERIAL_PORT_DEFAULT  « " - baudrate : " « EC_BAUDRATE_DEFAULT « " - parité : " « EC_PARITY_DEFAULT
      « mendl;
00051     else{
00052         logEcran « "(serial) connexion ouverte avec succès sur le port " « EC_SERIAL_PORT_DEFAULT « "
      - baudrate : " « EC_BAUDRATE_DEFAULT « " - parité : " « EC_PARITY_DEFAULT « mendl;
00053     }
00054
00055     return errorOpening;
00056 }
00057
00061 void Ecran::closeSerialConnection(){
00062     serial.flushReceiver();
00063     logEcran « "(serial) buffer Rx nettoyé avec succès" « mendl;
00064
00065     serial.closeDevice();
00066     logEcran « "(serial) connexion série fermée avec succès" « mendl;
00067 }
00068
00083 int Ecran::sendTrame(Trame_t trame_recue){
00084     int length_trame = 0;
00085
00086     if(trame_recue.taille >= 5)
00087         length_trame = trame_recue.taille;
00088     else
00089         return EC_TRAME_E_SIZE;
00090
00091     uint8_t trame[length_trame];
00092
00093     trame[0] = trame_recue.mode;
00094     trame[1] = trame_recue.categorie;
00095     trame[2] = trame_recue.sous_categorie;
00096
00097     for(size_t i = 0; i < sizeof(trame_recue.valeur)/sizeof(trame_recue.valeur[0]); i++){
00098         trame[i+3] = trame_recue.valeur[i];
00099     }
00100
00101     trame[length_trame-2] = EC_FIN_TRAME_1;
00102     trame[length_trame-1] = EC_FIN_TRAME_2;
00103
00104     serial.writeBytes(trame, length_trame);
00105
00106     logEcran « "(sendMsg) envoi de la trame effectué avec succès" « mendl;
00107     return EC_TRAME_E_SUCCESS;
00108 }
00109
00114 void Ecran::delay(unsigned int time){
      std::this_thread::sleep_for(std::chrono::milliseconds(time*1000)); }
```

5.5 Référence du fichier ecranlib.h

Fichier d'en-tête de la classe Ecran. Cette classe est utilisée afin de communique en UART avec l'écran sur la RaspberryPi.

```
#include "ecran_define.h"
#include "serialib.h"
#include "loglib.h"
#include <string>
#include <iomanip>
#include <iostream>
#include <chrono>
#include <thread>
#include <iterator>
```

Classes

— struct Trame_t
— class Ecran
Cette classe est utilisée pour la communication entre un ecran STM32F7G et une RaspberryPi.

5.5.1 Description détaillée

Fichier d'en-tête de la classe Ecran. Cette classe est utilisée afin de communique en UART avec l'écran sur la RaspberryPi.

Auteur

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

Version

1.0

Date

31/05/2022

Définition dans le fichier ecranlib.h.

5.6 ecranlib.h

Aller à la documentation de ce fichier.

```
00001
00008 #ifndef ECRAN_H
00009 #define ECRAN_H
00010
00011 #include "ecran_define.h"
00012 #include "serialib.h"
00013 #include "loglib.h"
00014 #include <string>
00015 #include <iomanip>
00016 #include <iostream>
00017 #include <chrono>
00018 #include <thread>
00019 #include <iterator>
00020
00025 typedef struct{
00026   Mode_Ecran_t mode;
00027   Categorie_Ecran_t categorie;
00028   Sous_Categorie_Ecran_t sous_categorie;
00029   uint8_t *valeur;
00030   int taille;
```

```
00031 } Trame_t;
00032
00036 class Ecran{
00037
00038 public:
00039
00040     // Constructeur de la classe écran
00041     Ecran();
00042
00043     // Desctructeur de la classe écran
00044     ~Ecran();
00045
00046     // Ouverture de la connexion série
00047     int openSerialConnection();
00048
00049     // Fermeture de la connexion série
00050     void closeSerialConnection();
00051
00052     // Envoi d'une trame en UART à l'écran
00053     void sendTrame(Trame_t trame);
00054
00055 private:
00056
00057     // Retard de temporisation dans l'exécution du code
00058     void delay(unsigned int time);
00059
00060 };
00061
00062 #endif // XBEE_H
```

5.7 Référence du fichier loglib.cpp

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

Fonctions

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

5.7.1 Documentation des fonctions

5.7.1.1 operator<<()

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

Définition à la ligne 20 du fichier loglib.cpp.

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

5.7.1.2 stringToChar()

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

Définition à la ligne 8 du fichier loglib.cpp.

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

5.8 loglib.cpp

Aller à la documentation de ce fichier.

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

5.9 Référence du fichier loglib.h

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

Classes

- struct Mendl
- class Log

Fonctions

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

Variables

— const Mendl mendl

5.9.1 Documentation des fonctions

5.9.1.1 operator<<()

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

Définition à la ligne 34 du fichier loglib.h.

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

5.9.1.2 stringToChar()

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

Définition à la ligne 8 du fichier loglib.cpp.

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

5.9.2 Documentation des variables

5.9.2.1 mendl

```
const Mendl mendl
```

Définition à la ligne 13 du fichier loglib.h.

5.10 loglib.h

Aller à la documentation de ce fichier.

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

5.11 Référence du fichier main.cpp

```
#include "ecranlib.h"
```

Fonctions

— int main (int argc, char ∗argv[ ])

5.11.1 Documentation des fonctions

5.11.1.1 main()

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

Définition à la ligne 5 du fichier main.cpp.

```
00005                                {
00006
00007     Ecran ecran;
00008     int status = ecran.openSerialConnection();
00009     if(status != EC_SER_E_SUCCESS)
00010         return 0;
00011
00012     uint8_t data[] = {'2', '3', '1'};
00013
00014     Trame_t trame;
00015
00016     trame.taille = (sizeof(data)/sizeof(data[0]))+5;
00017     trame.mode = EC_MODE_COMPETITION;
00018     trame.categorie = EC_CAT_SCORE;
00019     trame.sous_categorie = EC_SOUS_CAT_SCORE_TOTAL;
00020     trame.valeur = data;
00021
00022     ecran.sendTrame(trame);
00023
00024     ecran.closeSerialConnection();
00025     return 0;
00026 }
```

5.12 main.cpp

Aller à la documentation de ce fichier.

```
00001 #include "ecranlib.h"
00002
00003 using namespace std;
00004
00005 int main(int argc, char *argv[]){
00006
00007     Ecran ecran;
00008     int status = ecran.openSerialConnection();
00009     if(status != EC_SER_E_SUCCESS)
00010         return 0;
00011
00012     uint8_t data[] = {'2', '3', '1'};
00013
00014     Trame_t trame;
00015
00016     trame.taille = (sizeof(data)/sizeof(data[0]))+5;
00017     trame.mode = EC_MODE_COMPETITION;
00018     trame.categorie = EC_CAT_SCORE;
00019     trame.sous_categorie = EC_SOUS_CAT_SCORE_TOTAL;
00020     trame.valeur = data;
00021
00022     ecran.sendTrame(trame);
00023
00024     ecran.closeSerialConnection();
00025     return 0;
00026 }
```

5.13 Référence du fichier serialib.cpp

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

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

5.13.1 Description détaillée

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

Auteur

Philippe Lucidarme (University of Angers)

Version

2.0

Date

december the 27th of 2019

Définition dans le fichier serialib.cpp.

5.14 serialib.cpp

Aller à la documentation de ce fichier.

```
00001
00018 #include "serialib.h"
00019 #include "ecran_define.h"
00020
00021
00022
00023 //_____________________________________
00024 // ::: Constructors and destructors :::
00025
00026
00030 serialib::serialib()
00031 {
00032 #if defined (_WIN32) || defined( _WIN64)
00033     // Set default value for RTS and DTR (Windows only)
00034     currentStateRTS=true;
00035     currentStateDTR=true;
00036     hSerial = INVALID_HANDLE_VALUE;
00037 #endif
00038 #if defined (__linux__) || defined(__APPLE__)
00039     fd = -1;
00040 #endif
00041 }
00042
00043
00047 // Class desctructor
00048 serialib::~serialib()
00049 {
00050     closeDevice();
00051 }
00052
00053
00054
00055 //_____________________________________________
00056 // ::: Configuration and initialization :::
00057
00058
00059
00129 int serialib::openDevice(const char *Device, const unsigned int Bauds,
```

```
00130                           SerialDataBits Databits,
00131                           SerialParity Parity,
00132                           SerialStopBits Stopbits) {
00133 #if defined (_WIN32) || defined( _WIN64)
00134     // Open serial port
00135     hSerial = CreateFileA(Device,GENERIC_READ |
      GENERIC_WRITE,0,0,OPEN_EXISTING,/*FILE_ATTRIBUTE_NORMAL*/0,0);
00136     if(hSerial==INVALID_HANDLE_VALUE) {
00137         if(GetLastError()==ERROR_FILE_NOT_FOUND)
00138             return EC_SER_E_NOT_FOUND; // Device not found
00139
00140         // Error while opening the device
00141         return EC_SER_E_OPEN;
00142     }
00143
00144     // Set parameters
00145
00146     // Structure for the port parameters
00147     DCB dcbSerialParams;
00148     dcbSerialParams.DCBlength=sizeof(dcbSerialParams);
00149
00150     // Get the port parameters
00151     if (!GetCommState(hSerial, &dcbSerialParams)) return EC_SER_E_PARAM;
00152
00153     // Set the speed (Bauds)
00154     switch (Bauds)
00155     {
00156     case 110  :     dcbSerialParams.BaudRate=CBR_110; break;
00157     case 300  :     dcbSerialParams.BaudRate=CBR_300; break;
00158     case 600  :     dcbSerialParams.BaudRate=CBR_600; break;
00159     case 1200 :     dcbSerialParams.BaudRate=CBR_1200; break;
00160     case 2400 :     dcbSerialParams.BaudRate=CBR_2400; break;
00161     case 4800 :     dcbSerialParams.BaudRate=CBR_4800; break;
00162     case 9600 :     dcbSerialParams.BaudRate=CBR_9600; break;
00163     case 14400 :    dcbSerialParams.BaudRate=CBR_14400; break;
00164     case 19200 :    dcbSerialParams.BaudRate=CBR_19200; break;
00165     case 38400 :    dcbSerialParams.BaudRate=CBR_38400; break;
00166     case 56000 :    dcbSerialParams.BaudRate=CBR_56000; break;
00167     case 57600 :    dcbSerialParams.BaudRate=CBR_57600; break;
00168     case 115200 :   dcbSerialParams.BaudRate=CBR_115200; break;
00169     case 128000 :   dcbSerialParams.BaudRate=CBR_128000; break;
00170     case 256000 :   dcbSerialParams.BaudRate=CBR_256000; break;
00171     default : return EC_SER_E_UKN_BAUDRATE;
00172     }
00173     //select data size
00174     BYTE bytesize = 0;
00175     switch(Databits) {
00176         case SERIAL_DATABITS_5: bytesize = 5; break;
00177         case SERIAL_DATABITS_6: bytesize = 6; break;
00178         case SERIAL_DATABITS_7: bytesize = 7; break;
00179         case SERIAL_DATABITS_8: bytesize = 8; break;
00180         case SERIAL_DATABITS_16: bytesize = 16; break;
00181         default: return EC_SER_E_UKN_DATABITS;
00182     }
00183     BYTE stopBits = 0;
00184     switch(Stopbits) {
00185         case SERIAL_STOPBITS_1: stopBits = ONESTOPBIT; break;
00186         case SERIAL_STOPBITS_1_5: stopBits = ONE5STOPBITS; break;
00187         case SERIAL_STOPBITS_2: stopBits = TWOSTOPBITS; break;
00188         default: return EC_SER_E_UKN_STOPBITS;
00189     }
00190     BYTE parity = 0;
00191     switch(Parity) {
00192         case SERIAL_PARITY_NONE: parity = NOPARITY; break;
00193         case SERIAL_PARITY_EVEN: parity = EVENPARITY; break;
00194         case SERIAL_PARITY_ODD: parity = ODDPARITY; break;
00195         case SERIAL_PARITY_MARK: parity = MARKPARITY; break;
00196         case SERIAL_PARITY_SPACE: parity = SPACEPARITY; break;
00197         default: return EC_SER_E_UKN_PARITY;
00198     }
00199     // configure byte size
00200     dcbSerialParams.ByteSize = bytesize;
00201     // configure stop bits
00202     dcbSerialParams.StopBits = stopBits;
00203     // configure parity
00204     dcbSerialParams.Parity = parity;
00205
00206     // Write the parameters
00207     if(!SetCommState(hSerial, &dcbSerialParams)) return EC_SER_E_CONFIG;
00208
00209     // Set TimeOut
00210
00211     // Set the Timeout parameters
00212     timeouts.ReadIntervalTimeout=0;
00213     // No TimeOut
00214     timeouts.ReadTotalTimeoutConstant=MAXDWORD;
00215     timeouts.ReadTotalTimeoutMultiplier=0;
```

```
00216     timeouts.WriteTotalTimeoutConstant=MAXDWORD;
00217     timeouts.WriteTotalTimeoutMultiplier=0;
00218
00219     // Write the parameters
00220     if(!SetCommTimeouts(hSerial, &timeouts)) return EC_SER_E_TIMOUT;
00221
00222     // Opening successfull
00223     return 1;
00224 #endif
00225 #if defined (__linux__) || defined(__APPLE__)
00226     // Structure with the device's options
00227     struct termios options;
00228
00229
00230     // Open device
00231     fd = open(Device, O_RDWR | O_NOCTTY | O_NDELAY);
00232     // If the device is not open, return -1
00233     if (fd == -1) return EC_SER_E_OPEN;
00234     // Open the device in nonblocking mode
00235     fcntl(fd, F_SETFL, FNDELAY);
00236
00237
00238     // Get the current options of the port
00239     tcgetattr(fd, &options);
00240     // Clear all the options
00241     bzero(&options, sizeof(options));
00242
00243     // Prepare speed (Bauds)
00244     speed_t         Speed;
00245     switch (Bauds)
00246     {
00247     case 110  :     Speed=B110; break;
00248     case 300  :     Speed=B300; break;
00249     case 600  :     Speed=B600; break;
00250     case 1200 :     Speed=B1200; break;
00251     case 2400 :     Speed=B2400; break;
00252     case 4800 :     Speed=B4800; break;
00253     case 9600 :     Speed=B9600; break;
00254     case 19200 :    Speed=B19200; break;
00255     case 38400 :    Speed=B38400; break;
00256     case 57600 :    Speed=B57600; break;
00257     case 115200 :   Speed=B115200; break;
00258     default : return EC_SER_E_UKN_BAUDRATE;
00259     }
00260     int databits_flag = 0;
00261     switch(Databits) {
00262         case SERIAL_DATABITS_5: databits_flag = CS5; break;
00263         case SERIAL_DATABITS_6: databits_flag = CS6; break;
00264         case SERIAL_DATABITS_7: databits_flag = CS7; break;
00265         case SERIAL_DATABITS_8: databits_flag = CS8; break;
00266         //16 bits and everything else not supported
00267         default: return EC_SER_E_UKN_DATABITS;
00268     }
00269     int stopbits_flag = 0;
00270     switch(Stopbits) {
00271         case SERIAL_STOPBITS_1: stopbits_flag = 0; break;
00272         case SERIAL_STOPBITS_2: stopbits_flag = CSTOPB; break;
00273         //1.5 stopbits and everything else not supported
00274         default: return EC_SER_E_UKN_STOPBITS;
00275     }
00276     int parity_flag = 0;
00277     switch(Parity) {
00278         case SERIAL_PARITY_NONE: parity_flag = 0; break;
00279         case SERIAL_PARITY_EVEN: parity_flag = PARENB; break;
00280         case SERIAL_PARITY_ODD: parity_flag = (PARENB | PARODD); break;
00281         //mark and space parity not supported
00282         default: return EC_SER_E_UKN_PARITY;
00283     }
00284
00285     // Set the baud rate
00286     cfsetispeed(&options, Speed);
00287     cfsetospeed(&options, Speed);
00288     // Configure the device : data bits, stop bits, parity, no control flow
00289     // Ignore modem control lines (CLOCAL) and Enable receiver (CREAD)
00290     options.c_cflag |= ( CLOCAL | CREAD | databits_flag | parity_flag | stopbits_flag);
00291     options.c_iflag |= ( IGNPAR | IGNBRK );
00292     // Timer unused
00293     options.c_cc[VTIME]=0;
00294     // At least on character before satisfy reading
00295     options.c_cc[VMIN]=0;
00296     // Activate the settings
00297     tcsetattr(fd, TCSANOW, &options);
00298     // Success
00299     return (EC_SER_E_SUCCESS);
00300 #endif
00301
00302 }
```

```
00303
00304 bool serialib::isDeviceOpen()
00305 {
00306 #if defined (_WIN32) || defined( _WIN64)
00307     return hSerial != INVALID_HANDLE_VALUE;
00308 #endif
00309 #if defined (__linux__) || defined(__APPLE__)
00310     return fd >= 0;
00311 #endif
00312 }
00313
00317 void serialib::closeDevice()
00318 {
00319 #if defined (_WIN32) || defined( _WIN64)
00320     CloseHandle(hSerial);
00321     hSerial = INVALID_HANDLE_VALUE;
00322 #endif
00323 #if defined (__linux__) || defined(__APPLE__)
00324     close (fd);
00325     fd = -1;
00326 #endif
00327 }
00328
00329
00330
00331
00332 //_____________________________________________
00333 // ::: Read/Write operation on characters :::
00334
00335
00336
00343 char serialib::writeChar(const char Byte)
00344 {
00345 #if defined (_WIN32) || defined( _WIN64)
00346     // Number of bytes written
00347     DWORD dwBytesWritten;
00348     // Write the char to the serial device
00349     // Return -1 if an error occured
00350     if(!WriteFile(hSerial,&Byte,1,&dwBytesWritten,NULL)) return -1;
00351     // Write operation successfull
00352     return 1;
00353 #endif
00354 #if defined (__linux__) || defined(__APPLE__)
00355     // Write the char
00356     if (write(fd,&Byte,1)!=1) return -1;
00357
00358     // Write operation successfull
00359     return 1;
00360 #endif
00361 }
00362
00363
00364
00365 //_____________________________________________
00366 // ::: Read/Write operation on strings :::
00367
00368
00375 char serialib::writeString(const char *receivedString)
00376 {
00377 #if defined (_WIN32) || defined( _WIN64)
00378     // Number of bytes written
00379     DWORD dwBytesWritten;
00380     // Write the string
00381     if(!WriteFile(hSerial,receivedString,strlen(receivedString),&dwBytesWritten,NULL))
00382         // Error while writing, return -1
00383         return -1;
00384     // Write operation successfull
00385     return 1;
00386 #endif
00387 #if defined (__linux__) || defined(__APPLE__)
00388     // Lenght of the string
00389     int Lenght=strlen(receivedString);
00390     // Write the string
00391     if (write(fd,receivedString,Lenght)!=Lenght) return -1;
00392     // Write operation successfull
00393     return 1;
00394 #endif
00395 }
00396
00397 // _____________________________________
00398 // ::: Read/Write operation on bytes :::
00399
00400
00401
00409 char serialib::writeBytes(const void *Buffer, const unsigned int NbBytes)
00410 {
00411 #if defined (_WIN32) || defined( _WIN64)
```

```
00412     // Number of bytes written
00413     DWORD dwBytesWritten;
00414     // Write data
00415     if(!WriteFile(hSerial, Buffer, NbBytes, &dwBytesWritten, NULL))
00416         // Error while writing, return -1
00417         return -1;
00418     // Write operation successfull
00419     return 1;
00420 #endif
00421 #if defined (__linux__) || defined(__APPLE__)
00422     // Write data
00423     if (write (fd,Buffer,NbBytes)!=(ssize_t)NbBytes) return -1;
00424     // Write operation successfull
00425     return 1;
00426 #endif
00427 }
00428
00429
00430
00441 char serialib::readChar(char *pByte,unsigned int timeOut_ms)
00442 {
00443 #if defined (_WIN32) || defined(_WIN64)
00444     // Number of bytes read
00445     DWORD dwBytesRead = 0;
00446
00447     // Set the TimeOut
00448     timeouts.ReadTotalTimeoutConstant=timeOut_ms;
00449
00450     // Write the parameters, return -1 if an error occured
00451     if(!SetCommTimeouts(hSerial, &timeouts)) return -1;
00452
00453     // Read the byte, return -2 if an error occured
00454     if(!ReadFile(hSerial,pByte, 1, &dwBytesRead, NULL)) return -2;
00455
00456     // Return 0 if the timeout is reached
00457     if (dwBytesRead==0) return 0;
00458
00459     // The byte is read
00460     return 1;
00461 #endif
00462 #if defined (__linux__) || defined(__APPLE__)
00463     // Timer used for timeout
00464     timeOut         timer;
00465     // Initialise the timer
00466     timer.initTimer();
00467     // While Timeout is not reached
00468     while (timer.elapsedTime_ms()<timeOut_ms || timeOut_ms==0)
00469     {
00470         // Try to read a byte on the device
00471         switch (read(fd,pByte,1)) {
00472         case 1  : return 1; // Read successfull
00473         case -1 : return -2; // Error while reading
00474         }
00475     }
00476     return 0;
00477 #endif
00478 }
00479
00480
00481
00492 int serialib::readStringNoTimeOut(char *receivedString,char finalChar,unsigned int maxNbBytes)
00493 {
00494     // Number of characters read
00495     unsigned int    NbBytes=0;
00496     // Returned value from Read
00497     char            charRead;
00498
00499     // While the buffer is not full
00500     while (NbBytes<maxNbBytes)
00501     {
00502         // Read a character with the restant time
00503         charRead=readChar(&receivedString[NbBytes]);
00504
00505         // Check a character has been read
00506         if (charRead==1)
00507         {
00508             // Check if this is the final char
00509             if (receivedString[NbBytes]==finalChar)
00510             {
00511                 // This is the final char, add zero (end of string)
00512                 receivedString  [++NbBytes]=0;
00513                 // Return the number of bytes read
00514                 return NbBytes;
00515             }
00516
00517             // The character is not the final char, increase the number of bytes read
00518             NbBytes++;
```

```
00519         }
00520
00521         // An error occured while reading, return the error number
00522         if (charRead<0) return charRead;
00523     }
00524     // Buffer is full : return -3
00525     return -3;
00526 }
00527
00528
00541 int serialib::readString(char *receivedString,char finalChar,unsigned int maxNbBytes,unsigned int
       timeOut_ms)
00542 {
00543     // Check if timeout is requested
00544     if (timeOut_ms==0) return readStringNoTimeOut(receivedString,finalChar,maxNbBytes);
00545
00546     // Number of bytes read
00547     unsigned int    nbBytes=0;
00548     // Character read on serial device
00549     char            charRead;
00550     // Timer used for timeout
00551     timeOut         timer;
00552     long int        timeOutParam;
00553
00554     // Initialize the timer (for timeout)
00555     timer.initTimer();
00556
00557     // While the buffer is not full
00558     while (nbBytes<maxNbBytes)
00559     {
00560         // Compute the TimeOut for the next call of ReadChar
00561         timeOutParam = timeOut_ms-timer.elapsedTime_ms();
00562
00563         // If there is time remaining
00564         if (timeOutParam>0)
00565         {
00566             // Wait for a byte on the serial link with the remaining time as timeout
00567             charRead=readChar(&receivedString[nbBytes],timeOutParam);
00568
00569             // If a byte has been received
00570             if (charRead==1)
00571             {
00572                 // Check if the character received is the final one
00573                 if (receivedString[nbBytes]==finalChar)
00574                 {
00575                     // Final character: add the end character 0
00576                     receivedString  [++nbBytes]=0;
00577                     // Return the number of bytes read
00578                     return nbBytes;
00579                 }
00580                 // This is not the final character, just increase the number of bytes read
00581                 nbBytes++;
00582             }
00583             // Check if an error occured during reading char
00584             // If an error occurend, return the error number
00585             if (charRead<0) return charRead;
00586         }
00587         // Check if timeout is reached
00588         if (timer.elapsedTime_ms()>timeOut_ms)
00589         {
00590             // Add the end caracter
00591             receivedString[nbBytes]=0;
00592             // Return 0 (timeout reached)
00593             return 0;
00594         }
00595     }
00596
00597     // Buffer is full : return -3
00598     return -3;
00599 }
00600
00601
00615 int serialib::readBytes (void *buffer,unsigned int maxNbBytes,unsigned int timeOut_ms, unsigned int
       sleepDuration_us)
00616 {
00617 #if defined (_WIN32) || defined(_WIN64)
00618     // Avoid warning while compiling
00619     UNUSED(sleepDuration_us);
00620
00621     // Number of bytes read
00622     DWORD dwBytesRead = 0;
00623
00624     // Set the TimeOut
00625     timeouts.ReadTotalTimeoutConstant=(DWORD)timeOut_ms;
00626
00627     // Write the parameters and return -1 if an error occrured
00628     if(!SetCommTimeouts(hSerial, &timeouts)) return -1;
```

```
00629
00630
00631     // Read the bytes from the serial device, return -2 if an error occured
00632     if(!ReadFile(hSerial,buffer,(DWORD)maxNbBytes,&dwBytesRead, NULL))  return -2;
00633
00634     // Return the byte read
00635     return dwBytesRead;
00636 #endif
00637 #if defined (__linux__) || defined(__APPLE__)
00638     // Timer used for timeout
00639     timeOut           timer;
00640     // Initialise the timer
00641     timer.initTimer();
00642     unsigned int      NbByteRead=0;
00643     // While Timeout is not reached
00644     while (timer.elapsedTime_ms()<timeOut_ms || timeOut_ms==0)
00645     {
00646         // Compute the position of the current byte
00647         unsigned char* Ptr=(unsigned char*)buffer+NbByteRead;
00648         // Try to read a byte on the device
00649         int Ret=read(fd,(void*)Ptr,maxNbBytes-NbByteRead);
00650         // Error while reading
00651         if (Ret==-1) return -2;
00652
00653         // One or several byte(s) has been read on the device
00654         if (Ret>0)
00655         {
00656             // Increase the number of read bytes
00657             NbByteRead+=Ret;
00658             // Success : bytes has been read
00659             if (NbByteRead>=maxNbBytes)
00660                 return NbByteRead;
00661         }
00662         // Suspend the loop to avoid charging the CPU
00663         usleep (sleepDuration_us);
00664     }
00665     // Timeout reached, return the number of bytes read
00666     return NbByteRead;
00667 #endif
00668 }
00669
00670
00671
00672
00673 // ____________________________
00674 // ::: Special operation :::
00675
00676
00677
00683 char serialib::flushReceiver()
00684 {
00685 #if defined (_WIN32) || defined(_WIN64)
00686     // Purge receiver
00687     return PurgeComm (hSerial, PURGE_RXCLEAR);
00688 #endif
00689 #if defined (__linux__) || defined(__APPLE__)
00690     // Purge receiver
00691     tcflush(fd,TCIFLUSH);
00692     return true;
00693 #endif
00694 }
00695
00696
00697
00702 int serialib::available()
00703 {
00704 #if defined (_WIN32) || defined(_WIN64)
00705     // Device errors
00706     DWORD commErrors;
00707     // Device status
00708     COMSTAT commStatus;
00709     // Read status
00710     ClearCommError(hSerial, &commErrors, &commStatus);
00711     // Return the number of pending bytes
00712     return commStatus.cbInQue;
00713 #endif
00714 #if defined (__linux__) || defined(__APPLE__)
00715     int nBytes=0;
00716     // Return number of pending bytes in the receiver
00717     ioctl(fd, FIONREAD, &nBytes);
00718     return nBytes;
00719 #endif
00720
00721 }
00722
00723
00724
```

```
00725 // _________________
00726 // ::: I/O Access :::
00727
00737 bool serialib::DTR(bool status)
00738 {
00739     if (status)
00740         // Set DTR
00741         return this->setDTR();
00742     else
00743         // Unset DTR
00744         return this->clearDTR();
00745 }
00746
00747
00754 bool serialib::setDTR()
00755 {
00756 #if defined (_WIN32) || defined(_WIN64)
00757     // Set DTR
00758     currentStateDTR=true;
00759     return EscapeCommFunction(hSerial,SETDTR);
00760 #endif
00761 #if defined (__linux__) || defined(__APPLE__)
00762     // Set DTR
00763     int status_DTR=0;
00764     ioctl(fd, TIOCMGET, &status_DTR);
00765     status_DTR |= TIOCM_DTR;
00766     ioctl(fd, TIOCMSET, &status_DTR);
00767     return true;
00768 #endif
00769 }
00770
00777 bool serialib::clearDTR()
00778 {
00779 #if defined (_WIN32) || defined(_WIN64)
00780     // Clear DTR
00781     currentStateDTR=true;
00782     return EscapeCommFunction(hSerial,CLRDTR);
00783 #endif
00784 #if defined (__linux__) || defined(__APPLE__)
00785     // Clear DTR
00786     int status_DTR=0;
00787     ioctl(fd, TIOCMGET, &status_DTR);
00788     status_DTR &= ~TIOCM_DTR;
00789     ioctl(fd, TIOCMSET, &status_DTR);
00790     return true;
00791 #endif
00792 }
00793
00794
00795
00805 bool serialib::RTS(bool status)
00806 {
00807     if (status)
00808         // Set RTS
00809         return this->setRTS();
00810     else
00811         // Unset RTS
00812         return this->clearRTS();
00813 }
00814
00815
00822 bool serialib::setRTS()
00823 {
00824 #if defined (_WIN32) || defined(_WIN64)
00825     // Set RTS
00826     currentStateRTS=false;
00827     return EscapeCommFunction(hSerial,SETRTS);
00828 #endif
00829 #if defined (__linux__) || defined(__APPLE__)
00830     // Set RTS
00831     int status_RTS=0;
00832     ioctl(fd, TIOCMGET, &status_RTS);
00833     status_RTS |= TIOCM_RTS;
00834     ioctl(fd, TIOCMSET, &status_RTS);
00835     return true;
00836 #endif
00837 }
00838
00839
00840
00847 bool serialib::clearRTS()
00848 {
00849 #if defined (_WIN32) || defined(_WIN64)
00850     // Clear RTS
00851     currentStateRTS=false;
00852     return EscapeCommFunction(hSerial,CLRRTS);
00853 #endif
```

```
00854 #if defined (__linux__) || defined(__APPLE__)
00855     // Clear RTS
00856     int status_RTS=0;
00857     ioctl(fd, TIOCMGET, &status_RTS);
00858     status_RTS &= ~TIOCM_RTS;
00859     ioctl(fd, TIOCMSET, &status_RTS);
00860     return true;
00861 #endif
00862 }
00863
00864
00865
00866
00872 bool serialib::isCTS()
00873 {
00874 #if defined (_WIN32) || defined(_WIN64)
00875     DWORD modemStat;
00876     GetCommModemStatus(hSerial, &modemStat);
00877     return modemStat & MS_CTS_ON;
00878 #endif
00879 #if defined (__linux__) || defined(__APPLE__)
00880     int status=0;
00881     //Get the current status of the CTS bit
00882     ioctl(fd, TIOCMGET, &status);
00883     return status & TIOCM_CTS;
00884 #endif
00885 }
00886
00887
00888
00894 bool serialib::isDSR()
00895 {
00896 #if defined (_WIN32) || defined(_WIN64)
00897     DWORD modemStat;
00898     GetCommModemStatus(hSerial, &modemStat);
00899     return modemStat & MS_DSR_ON;
00900 #endif
00901 #if defined (__linux__) || defined(__APPLE__)
00902     int status=0;
00903     //Get the current status of the DSR bit
00904     ioctl(fd, TIOCMGET, &status);
00905     return status & TIOCM_DSR;
00906 #endif
00907 }
00908
00909
00910
00911
00912
00913
00920 bool serialib::isDCD()
00921 {
00922 #if defined (_WIN32) || defined(_WIN64)
00923     DWORD modemStat;
00924     GetCommModemStatus(hSerial, &modemStat);
00925     return modemStat & MS_RLSD_ON;
00926 #endif
00927 #if defined (__linux__) || defined(__APPLE__)
00928     int status=0;
00929     //Get the current status of the DCD bit
00930     ioctl(fd, TIOCMGET, &status);
00931     return status & TIOCM_CAR;
00932 #endif
00933 }
00934
00935
00941 bool serialib::isRI()
00942 {
00943 #if defined (_WIN32) || defined(_WIN64)
00944     DWORD modemStat;
00945     GetCommModemStatus(hSerial, &modemStat);
00946     return modemStat & MS_RING_ON;
00947 #endif
00948 #if defined (__linux__) || defined(__APPLE__)
00949     int status=0;
00950     //Get the current status of the RING bit
00951     ioctl(fd, TIOCMGET, &status);
00952     return status & TIOCM_RNG;
00953 #endif
00954 }
00955
00956
00963 bool serialib::isDTR()
00964 {
00965 #if defined (_WIN32) || defined( _WIN64)
00966     return currentStateDTR;
00967 #endif
```

```
00968 #if defined (__linux__) || defined(__APPLE__)
00969     int status=0;
00970     //Get the current status of the DTR bit
00971     ioctl(fd, TIOCMGET, &status);
00972     return status & TIOCM_DTR  ;
00973 #endif
00974 }
00975
00976
00977
00984 bool serialib::isRTS()
00985 {
00986 #if defined (_WIN32) || defined(_WIN64)
00987     return currentStateRTS;
00988 #endif
00989 #if defined (__linux__) || defined(__APPLE__)
00990     int status=0;
00991     //Get the current status of the CTS bit
00992     ioctl(fd, TIOCMGET, &status);
00993     return status & TIOCM_RTS;
00994 #endif
00995 }
00996
00997
00998
00999
01000
01001
01002 // ****************************************
01003 //  Class timeOut
01004 // ****************************************
01005
01006
01010 // Constructor
01011 timeOut::timeOut()
01012 {}
01013
01014
01018 //Initialize the timer
01019 void timeOut::initTimer()
01020 {
01021 #if defined (NO_POSIX_TIME)
01022     LARGE_INTEGER tmp;
01023     QueryPerformanceFrequency(&tmp);
01024     counterFrequency = tmp.QuadPart;
01025     // Used to store the previous time (for computing timeout)
01026     QueryPerformanceCounter(&tmp);
01027     previousTime = tmp.QuadPart;
01028 #else
01029     gettimeofday(&previousTime, NULL);
01030 #endif
01031 }
01032
01038 //Return the elapsed time since initialization
01039 unsigned long int timeOut::elapsedTime_ms()
01040 {
01041 #if defined (NO_POSIX_TIME)
01042     // Current time
01043     LARGE_INTEGER CurrentTime;
01044     // Number of ticks since last call
01045     int sec;
01046
01047     // Get current time
01048     QueryPerformanceCounter(&CurrentTime);
01049
01050     // Compute the number of ticks elapsed since last call
01051     sec=CurrentTime.QuadPart-previousTime;
01052
01053     // Return the elapsed time in milliseconds
01054     return sec/(counterFrequency/1000);
01055 #else
01056     // Current time
01057     struct timeval CurrentTime;
01058     // Number of seconds and microseconds since last call
01059     int sec,usec;
01060
01061     // Get current time
01062     gettimeofday(&CurrentTime, NULL);
01063
01064     // Compute the number of seconds and microseconds elapsed since last call
01065     sec=CurrentTime.tv_sec-previousTime.tv_sec;
01066     usec=CurrentTime.tv_usec-previousTime.tv_usec;
01067
01068     // If the previous usec is higher than the current one
01069     if (usec<0)
01070     {
01071         // Recompute the microseonds and substract one second
```

```
01072         usec=1000000-previousTime.tv_usec+CurrentTime.tv_usec;
01073         sec--;
01074     }
01075
01076     // Return the elapsed time in milliseconds
01077     return sec*1000+usec/1000;
01078 #endif
01079 }
```

5.15 Référence du fichier serialib.h

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

Classes

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

Macros

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

Énumérations

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

5.15.1 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.

5.15.2 Documentation des macros

5.15.2.1 UNUSED

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

To avoid unused parameters

Définition à la ligne 56 du fichier serialib.h.

5.15.3 Documentation du type de l'énumération

5.15.3.1 SerialDataBits

```
enum SerialDataBits
```

number of serial data bits

Valeurs énumérées

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

Définition à la ligne 61 du fichier serialib.h.

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

5.15.3.2 SerialParity

```
enum SerialParity
```

type of serial parity bits

Valeurs énumérées

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

Définition à la ligne 81 du fichier serialib.h.

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

5.15.3.3 SerialStopBits

```
enum SerialStopBits
```

number of serial stop bits

Valeurs énumérées

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

Définition à la ligne 72 du fichier serialib.h.

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

5.16 serialib.h

Aller à la documentation de ce fichier.

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

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

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

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

Index

~Ecran
    Ecran, 8
~serialib
    serialib, 14

available
    serialib, 15

categorie
    Trame_t, 33
Categorie_Ecran_t
    ecran_define.h, 39
clearDTR
    serialib, 15
clearRTS
    serialib, 15
closeDevice
    serialib, 16
closeSerialConnection
    Ecran, 8
currentStateDTR
    serialib, 30
currentStateRTS
    serialib, 30

delay
    Ecran, 8
DTR
    serialib, 16

EC_BAUDRATE_DEFAULT
    ecran_define.h, 36
EC_CAT_INIT
    ecran_define.h, 40
EC_CAT_SCORE
    ecran_define.h, 40
EC_CAT_TOF
    ecran_define.h, 40
EC_DATABITS_DEFAULT
    ecran_define.h, 36
EC_FIN_TRAME_1
    ecran_define.h, 36
EC_FIN_TRAME_2
    ecran_define.h, 36
EC_MODE_COMPETITION
    ecran_define.h, 40
EC_MODE_MAINTENANCE
    ecran_define.h, 40
EC_PARITY_DEFAULT
    ecran_define.h, 36
EC_SER_E_CONFIG
    ecran_define.h, 36
EC_SER_E_NOT_FOUND
    ecran_define.h, 37
EC_SER_E_OPEN
    ecran_define.h, 37
EC_SER_E_PARAM
    ecran_define.h, 37
EC_SER_E_SUCCESS
    ecran_define.h, 37
EC_SER_E_TIMOUT
    ecran_define.h, 37
EC_SER_E_UKN_BAUDRATE
    ecran_define.h, 37
EC_SER_E_UKN_DATABITS
    ecran_define.h, 38
EC_SER_E_UKN_PARITY
    ecran_define.h, 38
EC_SER_E_UKN_STOPBITS
    ecran_define.h, 38
EC_SERIAL_PORT_DEFAULT
    ecran_define.h, 38
EC_SOUS_CAT_BASE
    ecran_define.h, 40
EC_SOUS_CAT_CAN
    ecran_define.h, 40
EC_SOUS_CAT_ODO
    ecran_define.h, 40
EC_SOUS_CAT_SCORE_GRD_ROB
    ecran_define.h, 40
EC_SOUS_CAT_SCORE_PET_ROB
    ecran_define.h, 40
EC_SOUS_CAT_SCORE_TOTAL
    ecran_define.h, 40
EC_SOUS_CAT_TOF_0_AVG
    ecran_define.h, 40
EC_SOUS_CAT_TOF_1_MG
    ecran_define.h, 40
EC_SOUS_CAT_TOF_2_ARG
    ecran_define.h, 40
EC_SOUS_CAT_TOF_3_B
    ecran_define.h, 40
EC_SOUS_CAT_TOF_4_ARD
    ecran_define.h, 40
EC_SOUS_CAT_TOF_5_MD
    ecran_define.h, 40
EC_SOUS_CAT_TOF_6_AVD
    ecran_define.h, 40
EC_SOUS_CAT_TOF_7_HD

ecran_define.h, 40
EC_SOUS_CAT_TOF_8_HG
ecran_define.h, 40
EC_SOUS_CAT_XBEE
ecran_define.h, 40
EC_STOPBITS_DEFAULT
ecran_define.h, 38
EC_TRAME_E_CAT
ecran_define.h, 38
EC_TRAME_E_DATA
ecran_define.h, 39
EC_TRAME_E_END
ecran_define.h, 39
EC_TRAME_E_MODE
ecran_define.h, 39
EC_TRAME_E_SIZE
ecran_define.h, 39
EC_TRAME_E_SOUS_CAT
ecran_define.h, 39
EC_TRAME_E_SUCCESS
ecran_define.h, 39
Ecran, 7
~Ecran, 8
closeSerialConnection, 8
delay, 8
Ecran, 7
openSerialConnection, 9
sendTrame, 9
ecran_define.h, 35
Categorie_Ecran_t, 39
EC_BAUDRATE_DEFAULT, 36
EC_CAT_INIT, 40
EC_CAT_SCORE, 40
EC_CAT_TOF, 40
EC_DATABITS_DEFAULT, 36
EC_FIN_TRAME_1, 36
EC_FIN_TRAME_2, 36
EC_MODE_COMPETITION, 40
EC_MODE_MAINTENANCE, 40
EC_PARITY_DEFAULT, 36
EC_SER_E_CONFIG, 36
EC_SER_E_NOT_FOUND, 37
EC_SER_E_OPEN, 37
EC_SER_E_PARAM, 37
EC_SER_E_SUCCESS, 37
EC_SER_E_TIMOUT, 37
EC_SER_E_UKN_BAUDRATE, 37
EC_SER_E_UKN_DATABITS, 38
EC_SER_E_UKN_PARITY, 38
EC_SER_E_UKN_STOPBITS, 38
EC_SERIAL_PORT_DEFAULT, 38
EC_SOUS_CAT_BASE, 40
EC_SOUS_CAT_CAN, 40
EC_SOUS_CAT_ODO, 40
EC_SOUS_CAT_SCORE_GRD_ROB, 40
EC_SOUS_CAT_SCORE_PET_ROB, 40
EC_SOUS_CAT_SCORE_TOTAL, 40
EC_SOUS_CAT_TOF_0_AVG, 40
EC_SOUS_CAT_TOF_1_MG, 40
EC_SOUS_CAT_TOF_2_ARG, 40
EC_SOUS_CAT_TOF_3_B, 40
EC_SOUS_CAT_TOF_4_ARD, 40
EC_SOUS_CAT_TOF_5_MD, 40
EC_SOUS_CAT_TOF_6_AVD, 40
EC_SOUS_CAT_TOF_7_HD, 40
EC_SOUS_CAT_TOF_8_HG, 40
EC_SOUS_CAT_XBEE, 40
EC_STOPBITS_DEFAULT, 38
EC_TRAME_E_CAT, 38
EC_TRAME_E_DATA, 39
EC_TRAME_E_END, 39
EC_TRAME_E_MODE, 39
EC_TRAME_E_SIZE, 39
EC_TRAME_E_SOUS_CAT, 39
EC_TRAME_E_SUCCESS, 39
Mode_Ecran_t, 40
Sous_Categorie_Ecran_t, 40
ecranlib.cpp, 42
logEcran, 42
serial, 42
ecranlib.h, 43
elapsedTime_ms
timeOut, 32

flushReceiver
serialib, 17

initTimer
timeOut, 32
isCTS
serialib, 17
isDCD
serialib, 17
isDeviceOpen
serialib, 18
isDSR
serialib, 18
isDTR
serialib, 18
isRI
serialib, 19
isRTS
serialib, 19

Log, 10
Log, 11
name, 12
operator<<, 11, 12
save, 11
ss, 12
logEcran
ecranlib.cpp, 42
loglib.cpp, 45
operator<<, 45
stringToChar, 45
loglib.h, 46
mendl, 47

operator<<, 47
stringToChar, 47

main
main.cpp, 48
main.cpp, 48
main, 48
Mendl, 13
mendl
loglib.h, 47
mode
Trame_t, 33
Mode_Ecran_t
ecran_define.h, 40

name
Log, 12

openDevice
serialib, 19
openSerialConnection
Ecran, 9
operator<<
Log, 11, 12
loglib.cpp, 45
loglib.h, 47

previousTime
timeOut, 33

readBytes
serialib, 23
readChar
serialib, 24
readString
serialib, 25
readStringNoTimeOut
serialib, 26
RTS
serialib, 27

save
Log, 11
sendTrame
Ecran, 9
serial
ecranlib.cpp, 42
SERIAL_DATABITS_16
serialib.h, 61
SERIAL_DATABITS_5
serialib.h, 61
SERIAL_DATABITS_6
serialib.h, 61
SERIAL_DATABITS_7
serialib.h, 61
SERIAL_DATABITS_8
serialib.h, 61
SERIAL_PARITY_EVEN
serialib.h, 62
SERIAL_PARITY_MARK
serialib.h, 62
SERIAL_PARITY_NONE
serialib.h, 62
SERIAL_PARITY_ODD
serialib.h, 62
SERIAL_PARITY_SPACE
serialib.h, 62
SERIAL_STOPBITS_1
serialib.h, 62
SERIAL_STOPBITS_1_5
serialib.h, 62
SERIAL_STOPBITS_2
serialib.h, 62
SerialDataBits
serialib.h, 61
serialib, 13
~serialib, 14
available, 15
clearDTR, 15
clearRTS, 15
closeDevice, 16
currentStateDTR, 30
currentStateRTS, 30
DTR, 16
flushReceiver, 17
isCTS, 17
isDCD, 17
isDeviceOpen, 18
isDSR, 18
isDTR, 18
isRI, 19
isRTS, 19
openDevice, 19
readBytes, 23
readChar, 24
readString, 25
readStringNoTimeOut, 26
RTS, 27
serialib, 14
setDTR, 27
setRTS, 28
writeBytes, 28
writeChar, 29
writeString, 30
serialib.cpp, 49
serialib.h, 60
SERIAL_DATABITS_16, 61
SERIAL_DATABITS_5, 61
SERIAL_DATABITS_6, 61
SERIAL_DATABITS_7, 61
SERIAL_DATABITS_8, 61
SERIAL_PARITY_EVEN, 62
SERIAL_PARITY_MARK, 62
SERIAL_PARITY_NONE, 62
SERIAL_PARITY_ODD, 62
SERIAL_PARITY_SPACE, 62
SERIAL_STOPBITS_1, 62
SERIAL_STOPBITS_1_5, 62

SERIAL_STOPBITS_2, 62
SerialDataBits, 61
SerialParity, 61
SerialStopBits, 62
UNUSED, 61
SerialParity
serialib.h, 61
SerialStopBits
serialib.h, 62
setDTR
serialib, 27
setRTS
serialib, 28
sous_categorie
Trame_t, 34
Sous_Categorie_Ecran_t
ecran_define.h, 40
ss
Log, 12
stringToChar
loglib.cpp, 45
loglib.h, 47

taille
Trame_t, 34
timeOut, 31
elapsedTime_ms, 32
initTimer, 32
previousTime, 33
timeOut, 31
Trame_t, 33
categorie, 33
mode, 33
sous_categorie, 34
taille, 34
valeur, 34

UNUSED
serialib.h, 61

valeur
Trame_t, 34

writeBytes
serialib, 28
writeChar
serialib, 29
writeString
serialib, 30