

Manuel de la bibliothèque C++ de communication ethernet pour le projet soleil Bibliothèque version 2.02

Table des matières

1 Généralités.....	2
1.1 Architecture matérielle.....	2
1.2 Architecture logicielle.....	2
1.3 La connexion TCP/IP sur BILT.....	2
2 La classe TSteerer.....	3
2.1 Créer un steerer.....	3
2.2 Les opérations accessibles sur un steerer.....	4
3 La classe TAxis.....	4
3.1 Les opérations accessibles sur l'axe d'un steerer.....	4
3.2 Lecture de l'état de l'axe et des mesures en une opération.....	5
3.3 exemple.....	6
4 Les exceptions de la bibliothèque.....	7
5 Exemples d'utilisation de la classe.....	9
5.1 Création du steerer.....	9
5.2 Alimentation d'un axe et passage en consigne LIBERA.....	9
5.3 Récupération les données de l'axe.....	9
5.4 Gestion des défauts de l'axe.....	10
5.5 Gestion des exceptions.....	10
6 Le makefile.....	10

BILT intègre une pile TCP/IP fonctionnant à 100 Mbps acceptant jusqu'à 4 connexions simultanées. Les données sont envoyées par le port 5025(port dédié au protocole SCPI).

2 La classe TSteerer

L'utilisateur de la classe manipule un objet steerer contenant deux axes X et Y. Toutes les méthodes de gestion des alimentations sont sous cet indentation. Chaque méthode peut générer une exception qui sera remontée aux différents niveaux aboutissant à une description précise de l'erreur.

```
#include "ClassSteerer.h"
#include "Exception.h"

using namespace itest;                                //espace de nom itest

main (int argc, char **argv)
{ try{
    TSteerer Steerer1("192.168.150.113",1);           //Création du steerer n°1 -> ouverture de la socket
    Steerer1.axis(X).set_current(1.111);              //Envoie de la consigne en courant à l'axe X du steerer et fermeture de la socket
} catch (IttestException &e)
{ //gestion exception itest }                         //gestion des exceptions empilées dans le vecteur d'erreur
return 0;
}
```

Toutes les fonctions utilisées dans la bibliothèque sont des fonctions systèmes ou de la bibliothèque standard.

ATTENTION: le séparateur décimal est le point.

2.1 Créer un steerer

Chaque objet steerer crée un socket sur le PC. Il établit une ligne de communication avec le BILT dont c'est l'adresse. La liaison reste ouverte jusqu'à la destruction de l'objet. Si la liaison ne peut pas se faire dans le temps de MAXTIME_CONNECT_READ(10s par défaut), une exception est remontée et l'objet n'est pas créé.

```
#include "ClassSteerer.h"
#include "Exception.h"
using namespace itest;

Tsteerer(const char *_AddrIp, int NumSteer);
```

Le constructeur a besoin de l'adresse IP du BILT contenant les 2 alimentations du steerer et du numéro de celui-ci (les steerers sont numérotés de 1 à 48, chaque BILT pilote deux steerers).

2.2 Les opérations accessibles sur un steerer

- Les propriétés de la classe

int num_steerer(void)	Numéro du steerer
Const char *addrIP_steerer(void)	Adresse IP de la classe (« 192.168.150.9 »)

- Les méthodes de la classe

TAxis & axis(int Num)	Sélectionne l'axe X ou Y du steerer
void clear_all_err(void)	Efface toutes les erreurs de la pile de BILT
void clear_alarm(void)	Efface les alarmes des deux axes du steerer
void set_switch(bool val)	Change la provenance des consignes en courant(Ethernet-local/LIBERA) des deux axes. 0 = Ethernet-local / 1 = LIBERA
void set_state(bool val)	Allume ou éteint les deux axes du steerer. 0 = OFF / 1 = ON

3 La classe TAxis

La classe TAxis permet d'accéder à toutes les opérations sur chaque axe d'un steerer. Cette classe est encapsulée dans la classe steerer. N'est accessible que la méthode axis() qui sélectionne le bon objet TAxis.

3.1 Les opérations accessibles sur l'axe d'un steerer

double get_measure_current(void)	Lecture de la mesure de courant en A
double get_measure_voltage(void)	Lecture de la mesure de tension en V
void get_mdata(struct mdata *)	Lecture de tous les états et mesures de l'axe(<i>voir ci dessous</i>).
int get_switch(void)	Lecture de l'état du switch de consigne 0 = Ethernet-local / 1 = LIBERA
double get_current(void)	Lecture de la consigne en courant de configuration (la consigne en courant envoyée par le libera ne peut être visualisé)
int get_state(void)	Lecture de l'état de l'axe(<i>voir ci dessous</i>)
int get_num_alarm(void)	Lecture du numéro d'alarme déclenché par l'axe(<i>voir ci dessous</i>)
const char * get_str_alarm(void)	Lecture de la chaîne de caractères d'alarme déclenchée par l'axe
void set_state(bool val)	Allume ou éteint l'axe 0 = OFF / 1 = ON
void set_current(double ValF)	Programme la consigne en courant de configuration en A
void set_switch(bool val)	Programme la provenance des consignes en courant 0 = Ethernet-local / 1 = LIBERA
void clear_alarm(void)	Efface les alarmes de l'axe

3.2 Lecture de l'état de l'axe et des mesures en une opération

La méthode **void get_mdata(struct mdata *)** permet de récupérer dans une structure de type mdata toutes les informations utiles à l'affichage.

- La structure mdata

Int steerer	Numéro du steerer
Int axis	Numéro de l'axe
Int state	État de l'axe
Std::string fail	Intitulé du défaut de l'axe(voir tableau ci dessous)
Double measvolt	Mesure de tension
Double meascurr	Mesure de courant

- Etat de l'axe(il est sauvegardé même après un arrêt de BILT)

state	Description
-1	Axe non créé au niveau BILT
0	Axe éteint
1	Axe allumé
2	Warning: défaut sur l'alimentation qui ne produit pas un arrêt de l'alimentation
3	Alarm: défaut sur l'alimentation qui produit un arrêt de l'alimentation

Ce tableau est aussi valable pour get_state().

- Défauts de l'axe(il n'est pas sauvegardé suite à une mise hors tension de BILT)

	fail	Description	Arrêt automatique de l'alimentation
0	NO	Pas d'alarme	
1	VIGT	Vigitherme	NON
2	OVU	Détection surtension (24V sur 10ms)	NON
3	OVI	Détection court circuit d'un des deux pont	OUI
4	PWSK	Puissance absorbée >10W sur 10ms	OUI
5	TEMP	Seuil de température semiconducteur	OUI
6	PWSO	Puissance fournie >120W sur 10ms	OUI

Ce tableau est aussi valable pour get_num_alarm() et get_str_alarm()

- Cas particulier d'un arrêt secteur alors que les alimentations sont allumées:

Une coupure secteur alors que les alimentations sont allumées provoque une alarme. Lors du redémarrage du BILT, l'état de l'axe est en alarme(3) mais le défaut de l'axe est à 0.(Il en va de même pour tout défaut non effacé après une mise hors tension du BILT).

3.3 exemple

État initial du steerer à l'arrêt et sans alarmes:

```
#include <sstream>
#include <iostream>
#include "ClassSteerer.h"
#include "Exception.h"

using namespace itest;                                //espace de nom itest

main (int argc, char **argv)
{ try{
    struct mdata myData1;
    TSteerer Steerer1("192.168.150.113",1);           //Création du steerer n°1 -> ouverture de la socket
    Steerer1.axis(X).set_current(1.111);               //Envoie de la consigne en courant à l'axe X du steerer
    Steerer1.axis(X).set_state(ON);                   //allume axe
    Steerer1.axis(X).set_switch(LIBERA);               //sélectionne consigne en courant venant du libera
    Steerer1.axis(X).get_mdata(&myData1);              //lecture de mdata et mise en forme des données dans la structure

    std::cout<<"STEERER:"<< myData1.steerer<<std::endl;
    std::cout<<"AXIS:"<< myData1.axis<<std::endl;
    std::cout<<"STATE:"<< myData1.state<<std::endl;
    std::cout<<"FAIL:"<< myData1.fail<<std::endl;
    std::cout<<"MV:"<< myData1.measvolt<<std::endl;
    std::cout<<"MC:"<< myData1.meascurr<<std::endl;

    Steerer1.axis(X).set_state(OFF);
} catch (IttestException &e)
{for(int i=0;i<e.errors.size();i++)                    //gestion des exceptions empilées dans le vecteur d'erreur
    std::cout<<e.errors[i].reason<<" " <<e.errors[i].desc<<" " <<e.errors[i].origin<<std::endl;}
    return 0;
}
```

4 Les exceptions de la bibliothèque

La classe ItestException regroupe toutes les exceptions de la bibliothèque. Chaque niveau d'exception est empilé dans un vecteur **errors** de classe Error.

- La classe Error

Membre	Description
Const char *reason / const std::string&	Raison de l'exception
Const char *desc / const std::string&	Description de l'exception
Const char *origin / const std::string&	Origine de l'exception
Int severity	Sévérité de l'exception
Int err_code	Code d'erreur de l'exception(géré seulement pour BILT)

La sévérité des exceptions n'est pas géré par la bibliothèque.

Toutes exceptions remontent sur deux niveaux:

- **Socket ou Bilt:** Bilt command failed , ItestException exception caught while trying "i2::curr 11"->"i2:Parameter data out of range" , itest::TBilt::write

Membre	Description
Bilt command failed	Raison de l'exception
ItestException exception caught while trying "i2::curr 11"->"i2:Parameter data out of range"	Description de l'exception : « comande envoyée » -> « message reçu ».
itest::TBilt::write	Origine de l'exception
	Sévérité de l'exception
-222	Code d'erreur de l'exception

- **Steerer:** Current could not be changed , ItestException exception caught on steerer1/Axis X(I2) , itest::TAxis::set_current

Membre	Description
Current could not be changed	Raison de l'exception
ItestException exception caught on steerer1/Axis X(I2)	Description de l'exception
itest::TAxis::set_current	Origine de l'exception
	Sévérité de l'exception
	Code d'erreur de l'exception

- Quelques exemples d'exceptions

Typee	Intitulé de l'exception	Description
socket	<i>système</i>	Erreur système renvoyée par errno et strerror(errno)
socket	"Could not connect with peripheral"	Impossible de se connecter au BILT dans les 10s.
socket	"Could not read data on peripheral"	Impossible de lire des données dans le BILT dans les 10s.
BILT	"Parameter data out of range"	Valeur de la commande hors gamme(ex: 11A pour set_curr(val) alors que l'alimentation fait 10A max)
BILT	"Parameter error"	Mauvais paramètre passé à la commande(ex: envoie une chaîne de caractère alors que la commande veut un chiffre)
BILT	"Missing parameter"	Manque le paramètre de la commande
BILT	"Settings conflict"	Mauvais état de l'instrument ou du groupe pour appliquer la commande voulue(ex: demande un allumage de l'alimentation alors qu'elle l'est déjà).
BILT	"Undefined header"	Commande inconnue(ex: essaie de communiquer avec un instrument qui n'est plus dans le châssis)
BILT	"Instrument Max-Time"	Problème matériel sur l'instrument

```
#include <sstream>
#include <iostream>
#include "ClassSteerer.h"
#include "Exception.h"

using namespace itest;                                //espace de nom itest

main (int argc, char **argv)
{ try {
    TSteerer Steerer1("192.168.150.113",1);           //Création du steerer n°1 -> ouverture de la socket
    Steerer1.axis(X).set_current(11);                 //Envoie de la consigne en courant à l'axe X du steerer(consigne > à la gamme)
} catch (IttestException &e)
{for(int i=0;i<e.errors.size();i++)                   //gestion des exceptions empilées dans le vecteur d'erreur
    std::cout<<e.errors[i].reason<<" , "<<e.errors[i].desc<<" , "<<e.errors[i].origin<<" , "<<e.errors[i].code<<std::endl;}
    return 0;
}
```


5 Exemples d'utilisation de la classe

5.1 Création du steerer

```
TSteerer Steerer1("192.168.150.113",1);    //Création du steerer n°1 -> ouverture de la socket
Steerer1.clear_all_err();
```

1. Création du Steerer avec son adresse IP et son numéro.
1. On vide le buffer d'erreurs de BILT(des erreurs peuvent être envoyées par la liaison locale et lors de la première commande ethernet une exception peut être générée).

5.2 Alimentation d'un axe et passage en consigne LIBERA

```
Steerer1.axis(X).set_current(1.2);          //Envoie de la consigne en courant à l'axe X du steerer
Steerer1.axis(X).get_mdata(&myData);
if(myData.state==3)
    { Steerer1.axis(X).clear_alarm(); Steerer1.axis(X).get_mdata(&myData); }
if(myData.state==0)
    { Steerer1.axis(X).set_state(ON); }
Steerer1.axis(X).set_switch(LIBERA);
```

1. Programmation de la consigne en courant initiale.
2. Lecture de mdata pour récupérer l'état de l'alimentation(on pourrait aussi utiliser get_state).
2. Test sur la structure. Si un défaut est apparu(state=3, qu'il a été identifié et réparé), on peut l'effacer.
3. Allume l'axe X avec la consigne de 1,2A programmé précédemment.
4. Change la sélection des consigne pour accepter celles du LIBERA.

5.3 Récupération les données de l'axe

```
struct mdata myDataX_Steer1,myDataY_Steer1;

Steerer1.axis(X).get_mdata(&myDataX_Steer1);
Steerer1.axis(Y).get_mdata(&myDataY_Steer1);
```

1. Récupère dans myData de toutes les informations de l'axe.

5.4 Gestion des défauts de l'axe

Les défauts de l'axe sont lus en toutes lettres dans **mdata.fail** ou avec **get_state()**(un arrêt de **BILT** efface l'intitulé du défaut mais pas son état). Les défauts qui arrêtent l'alimentation provoquent sur mdata.state un passage à l'état 3 alors que les défauts mineurs qui n'arrêtent pas l'alimentation provoquent sur mdata.state un passage à l'état 2. **Une application tiers doit donc vérifier l'état de l'axe en permanence et agir en conséquence.**

5.5 Gestion des exceptions

Toutes les exceptions sont sous la classe ItestException. Les exceptions de type socket sont levées lors de problème de liaison ethernet(liaison coupée, problème système...). Les exceptions de type BILT sont levées lors de problème de commande sur le châssis(mauvais état de l'alimentation pour modifier cette consigne, alimentation déjà allumée...). Le bloc catch sur ItestException lève donc toutes les exceptions de la bibliothèque.

Aucune exception n'est levée pendant la création d'un objet steerer. Si un problème est survenu pendant la création de l'objet(communication non établie avec le BILT...) une exception sera levée à chaque accès au BILT.

Un autre bloc catch(...) peut suivre pour lever les exceptions inattendues.

6 Le makefile

Le makefile distribué avec la bibliothèque compile et lie les différents fichiers entre eux pour créer une bibliothèque statique.

- make: compile et fait l'édition des liens. Par défaut la bibliothèque s'appelle « libSteer.a »
- make clean : nettoie les fichiers *.o
- make install : installe la bibliothèque dans le répertoire désiré(éditer le makefile et compléter INSTALLDIR).