PRELIMINARY

Manual library C++ for Ethernet communication for magnet control

Library release 2.00

Be2811 software release	V506

Table of contents

1 General

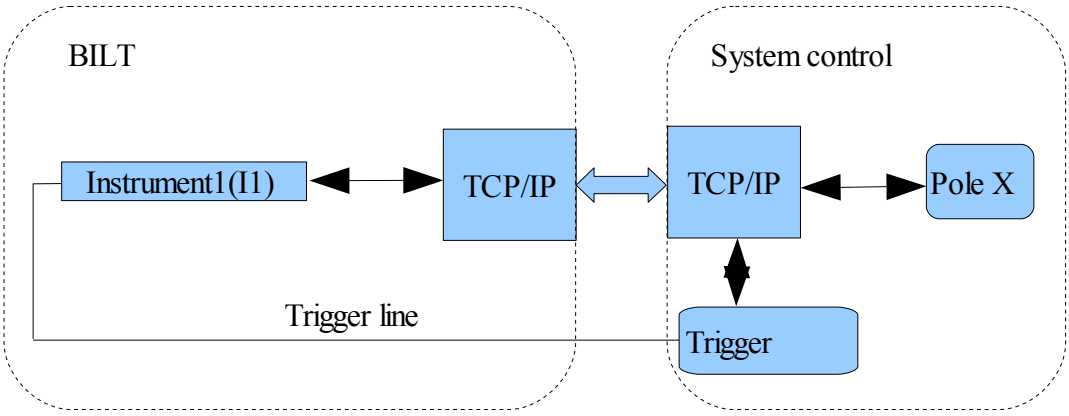
The remote control of all power supplies is made by TANGO system using Ethernet. All Bilt chassis are connected to the network and each one monitors magnets thanks to independent modules current dc source.
The static library (*.a) C++ of itest wraps Bilt protocol in friendly objects in order to simplify the integration in the control system.

1.1 Hardware architecture

The Bilt system is a versatile modular system, it allows the user to control different type of magnets. Depending on their architecture the number of power supplies used in the chassis can vary.

1.2 Software architecture

Each magnet is a pole (the power supply connected to the coil) named as **instrument**(I*X*).



1.3 TCP/IP stack in BILT

Bilt embeds a TCP/IP stack working at 100 Mbps allowing up to 10 simultaneous sockets at same time. Data are sent on port 5025 (SCPI raw port).

2 itPole2811 class

The pole object allows the user to manipulate the power supply connected to the magnet.

```
#include "ClassMagnet.h"
#include "Exception.h"

using namespace itest;                              //name space itest

main (int argc, char **argv)
{ try{
      itPole2811 Magnet1("192,168,150,128",1,1);              //Create Magnet 1 -> open socket
      Magnet1.set_current(1.111);                   //Send current setting and close socket
  }catch(ItestException &e)
  {//itest excpetion block}                        //managing itest excpetion stacked in error vector
   return 0;
}
```

- All functions used in the library are system or standard functions (no third party library).
- Decimal mark is the dot.

2.1 Create a magnet

Create a pole object opens a socket on the PC which is available until the destruction of the object.

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

itPole2811(const char *_AddrIp, int _NumPole,  int _NumInst);
```

The constructor needs the Bilt IP address containing the magnet and the positions of the instruments corresponding to its pole, the pole number is a user definition.

2.2 Using the magnet object

- Properties

int num_pole(void)	Magnet position (user definition)
Const char *addrIP(void)	IP address of the class (« 192.168.150.128 »)
Int instrument(void)	Position of the power supply in the Bilt

- Methods

GENERAL	
void clear_all_err(void)	Clear stacked errors in Bilt buffer
void clear_alarm(void)	Clear poles alarms
void set_state(bool val)	Switch on/off power supply. 0 = OFF / 1 = ON
int get_state(void)	Return state of the magnet (*see table below*).
void clear_alarm(void)	Clear alarm
int get_num_alarm(void)	Return last triggered alarm number (*see table below*).
void get_idata(struct idata *)	*See below*
Void get_ctn(int *,int *)	Get temperature sensors of the power supply(no accuracy specified, test method)
VOLTAGE	
double get_measure_voltage(void)	Return voltage measurement in V
CURRENT	
double get_measure_current(void)	Return current measurement in A
double get_current(void)	Return the current value
void set_current(double ValF)	Set current value in A
double get_current_tracking_level(void)	Return the maximum gap between the measure and the current setting max in which the tracking status will change its state
void set_current_tracking_level(double ValF)	Set the maximum gap between the measure and the current setting in which the tracking status will change its state
bool get_current_tracking_status(void)	Return 1 if (current – measure_current) < current_tracking_level else return 0
bool get_current_slope_completion(void)	Return 1 when the run time current setting has reached the user current setting else return 0
TRIGGER	
void set_trigger_delay(double ValI)	Set the delay before changing current setting after the trigger event in s
double get_trigger_delay(void)	Return the delay before changing current setting after the trigger event in s
Void initiate_trigger(void)	Simulate a software trigger event only for the pole
bool get_trigger_state(void)	Return the state of the trigger 0=no trigger allowed 1 = trigger allowed
void set_trigger_state(bool Val)	Set the state of the trigger 0=no trigger allowed 1 = trigger allowed

CURRENT WITH TRIGGER	
bool get_current_latch_source(void)	Return the source of the current setting when triggered 0 = current latch value / 1=current buffer values
void set_current_latch_source(bool Val)	Select the source of the current setting when triggered 0 = current latch value / 1=current buffer values
double get_current_latch(void)	Return the setting of the current that will be triggered (return NaN if no setting available).
Void set_current_latch(double ValF)	Set current value that will be triggered in A
CURRENT BUFFER WITH TRIGGER	
void set_current_buffer(float *valF,int index,int count);	Store a set of current value in the buffer beginning at index (1024 values maximum).
void get_current_buffer(float *valF,int index,int count);	Return a set of current value stored in the buffer beginning at index
void set_current_buffer_loop(bool state);	Select automatic loop in the range of values selected
bool get_current_buffer_loop(void);	Return state of the automatic loop
void set_current_buffer_sample(float valF);	Set the Sample period between two values
float get_current_buffer_sample(void);	Return the sample period
void set_current_buffer_range(int index,int count);	Select the range of values that will be executed. The first value is at index and the last is index + count -1
void get_current_buffer_range(int *index,int *count);	Return the range of values used in the buffer

1.

2.1 Returning state and measure in one operation

The method **void get_idata(struct idata *)** allows the user to get in one read from the **pole**, a structure of data useful for visualization.

- Structure idata

Int magnet	Magnet number
Int pole	Pole number
Int state	Pole state
Std::string fail	Default string (*see table below*).
Double measvolt	Voltage measurement
Double measurr	Current measurement

- Pole state

state	Description
0	Pole off
1	Pole on
2	Warning: default reported without action on the pole.
3	Alarm: default causing the stop of the pole.

This table is true for get_state().

- Pole default

	fail	Description	Magnet stops after triggered
0	NO	No alarm	
13	PSO	Power source > 100W during 25ms	YES
14	PSI	Power sink >0,2A during 25ms	YES
15	SAT	The voltage measure has saturated to 20V (cable broken, no load...)	YES
16	TEMP	Over temperature semiconductor detection	YES

This table is true for get_num_alarm()

2.1 Example

Initial state stopped without alarm:

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

using namespace itest;                          //name space itest

main (int argc, char **argv)
{ try{
        struct idata myData1;
        itPole2811 Magnet1("192,168,150,128",1",1);                    //Create Steerer 1 -> open socket
        Magnet1.set_current(1.111);             //Send current setting
        Magnet1.set_state(ON);                  //switch on axis
        Magnet1.get_idata(&myData1);            //reading idata and visualize data

       std::cout<<"MAGNET:"<< myData1.magnet<<std::endl;
       std::cout<<"STATE:"<< myData1.state<<std::endl;
       std::cout<<"POLE:"<< myData1.pole<<std::endl;
       std::cout<<"STATE:"<< myData.state<<std::endl;
       std::cout<<"FAIL:"<< myData1.fail<<std::endl;
       std::cout<<"MV:"<< myData1.measvolt<<std::endl;
       std::cout<<"MC:"<< myData1.measurr<<std::endl;

        Magnet1.set_state(OFF);                 //switch off magnet
  }catch(ItestException &e)
  {for(int i=0;i<e.errors.size();i++)           //managing itest excpetion stacked in error vector
        std::cout<<e.errors[i].reason<<" , "<<e.errors[i].desc<<" , "<<e.errors[i].origin<<std::endl;}
   return 0;
}
```

3 Exceptions

The ItestException class implements all exceptions of the library. Each level of exception is stacked in the vector **errors** from class Error.

- The Error class

properties	Description
Const char *reason / const std::string&	Exception reason
Const char *desc / const std::string&	Exception description
Const char *origin / const std::string&	Exception origin
Int severity	Exception severity(not used for BILT)*
Int err_code	Exception error code(only for BILT)

Severity is not managed by the library (always 1)

All exceptions are propagated on 2 levels:

- **Socket or Bilt**: Bilt command failed , ItestException exception caught while trying ":i1;:curr 11"->"i1:Parameter data out of range" , itest::TBilt::write,1,-222

segment	Description
Bilt command failed	Exception reason
ItestException exception caught while trying ":i1;:curr 11"->"i1:Parameter data out of range"	Exception description : « command sent » -> « message get».
itest::TBilt::write	Exception origin
1	Exception severity(not used for BILT)
-222	Exception error code

- **Magnet**: Current could not be changed , ItestException exception caught on magnet1/pole X(I1) , itest::itPole2811::set_current

segment	Description
Current could not be changed	Exception reason
ItestException exception caught on magnet1/Pole X(I1)	Exception description
itest::itPole2811::set_current	Exception origin
	Exception severity(not used for BILT)
	Exception error code

- Exception examples

Type	name	Description
socket	*system*	System error sent by errno et strerror(errno)
socket	"Could not connect with peripheral"	Cannot connect to a Bilt within10s.
socket	"Could not read data on peripheral"	Cannot read data from a Bilt within10s.
BILT	"Parameter data out of range"	Setting Value out of range (ex: 11A for set_curr(*val*) while Be548 is a 10A max power supply)
BILT	"Parameter error"	Bad parameter on command (ex: send a string while the command needs a number)
BILT	"Missing parameter"	Parameter is missing
BILT	"Settings conflict"	Instrument or group bad state while sending a command (ex: switch on the Be548 while it is already on).
BILT	"Undefined header"	Unknown command(ex: trying to send a command to an instrument which is not in the Bilt)
BILT	"Instrument Max-Time"	Unknown instrument error

```
#include "ClassMagnet.h"
#include "Exception.h"

using namespace itest;                                  //name space itest

main (int argc, char **argv)
{ try{
        itPole2811 Magnet1("192,168,150,128",1,1);                  //Create Magnet 1 -> open socket
        Magnet1.set_current(11);                                    ///Send current setting to axis X (value > range)
   }catch(ItestException &e)
   {for(int i=0;i<e.errors.size();i++)                  //managing itest excpetion stacked in error vector
        std::cout<<e.errors[i].reason<<" , "<<e.errors[i].desc<<" , "<<e.errors[i].origin<<" , "<<e.errors[i].code<<std::endl;}
    return 0;
}
```

1 Using the library

1.1 Create a magnet

```
itPole2811 Magnet1("192,168,150,128",1,1);                    //Create Magnet 1 -> open socket
Magnet1.clear_all_err();
```

1. Create magnet with IP address and instrument position.
1. Clear error buffer (Errors could have been stacked in Bilt on a previous use).

1.1 Get data

```
struct idata myData_magnet1;

Magnet1.get_idata(&myData_magnet1)
```

1. Get in myData structure, a set of information needed for monitoring.

1.1 Managing poles defaults

Defaults triggered by poles are read in **idata.fail** or with **get_num_alarm()**. A default of level 3 stops the pole while a default of level 2 is just reported. **An application must check poles state at a frequency compliant with the system in order to act properly if a default is triggered.**

1.2 Managing exceptions

- Socket exception type : caught by ethernet problem (system error, cable disconnected...).
- Bilt exception type: caught by command errors (bad state of pole while sending command...).

The catch block on ItestException gets all exception of the library.

No exception caught in the magnet constructor. If a problem occurred during the creation of the object (no communication with Bilt...) exceptions will be caught at all Bilt access.

Of course an another block catch can be placed below the bloc ItestException in order to catch unknown errors.

1 Trigger function

The trigger function configures the trigger and send a broadcast trigger on UDP.

void initiate_udp_trigger(const char *_mask)	Generate a broadcast trigger limited to the mask on the LAN

2 Linux version and GCC

Distribution	Fedora
Linux release	2.6.21
GCC release	4.1.2

3 The makefile

The make file distributed with the compiled library compiles and links files to create a static library.

- make: compile and link. Library named « libMagnet.a »
- make clean : clean files *.o
- make install : install library in a folder (edit makefile and change INSTALLDIR).