JRA1 Telescope: NI Flex RIO DAQ

Source Tree installation – Libraries – Coding conventions

Gilles Claus[1], Mathieu Goffe[1], Kimmo Jaaskelainen[1], Cayetano Santos[1], Matthieu Specht[1]

January 17, 2011

Abstract

The EUDET JRA1 Pixel Telescope is using a custom-made data acquisition system since a couple of years. In preparation for AIDA, the group decided to investigate different off the shelf I/O systems. The advantage of such a system is the easier support and the availability over the next years. The IPHC group selected the NI Flex Rio system and prepared LabView sources, which can rather easy be connected to the existing DAQ. In this memo the basics of the source code is described.

[1] IPHC, Strasbourg, France

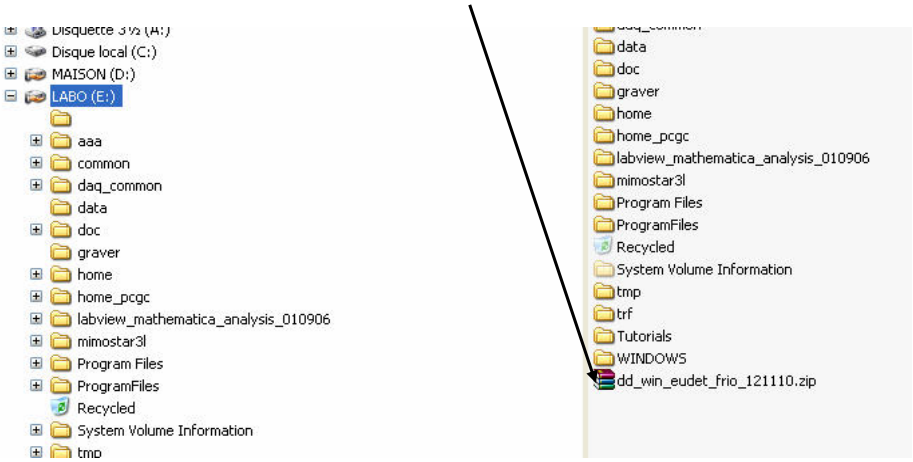
Inhaltsverzeichnis

1 Introduction

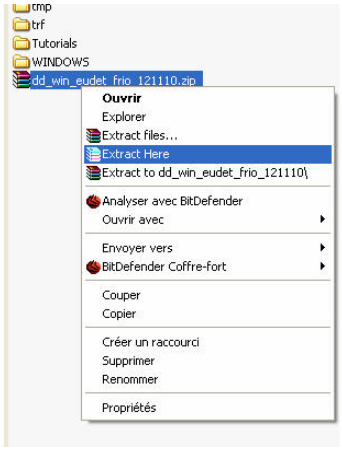
This document explains the source tree : the goal of each library, the source files organization, the conventions used for identifier names etc ...

2 Installation

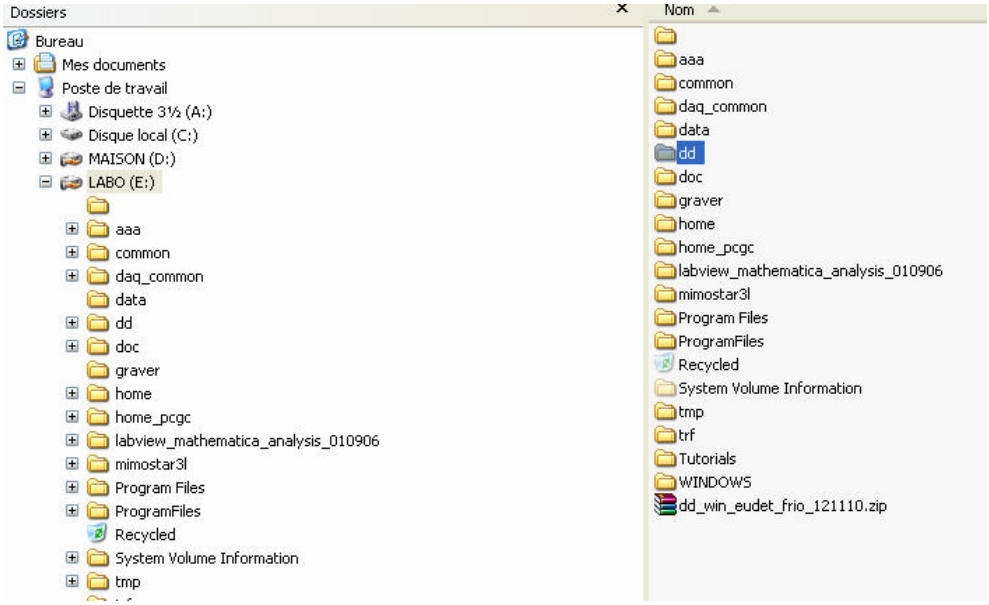
Please copy the file dd_win_eudet_frio_ddmmyy.zip on your hard disk root directory, for example dd_win_eudet_frio_121110.zip.



Unzip this file



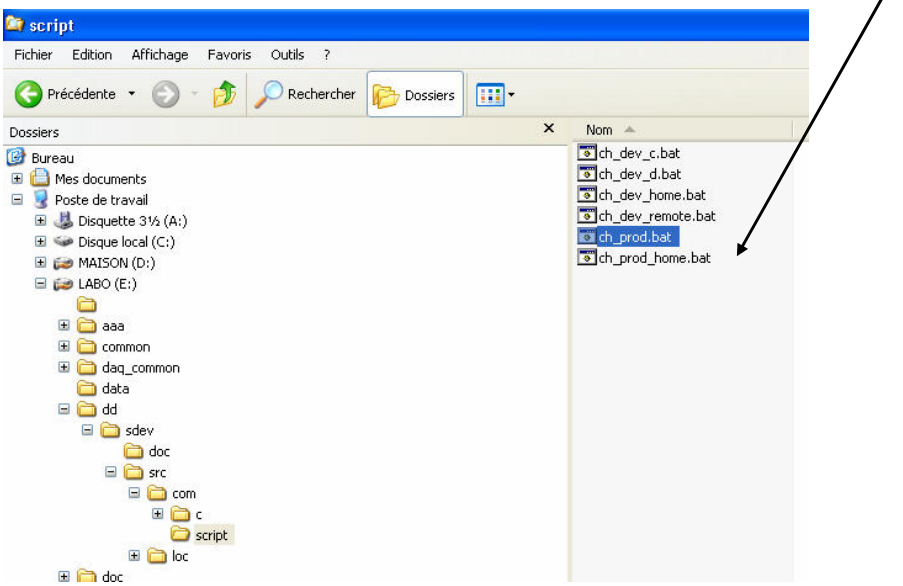
The directory \dd is created



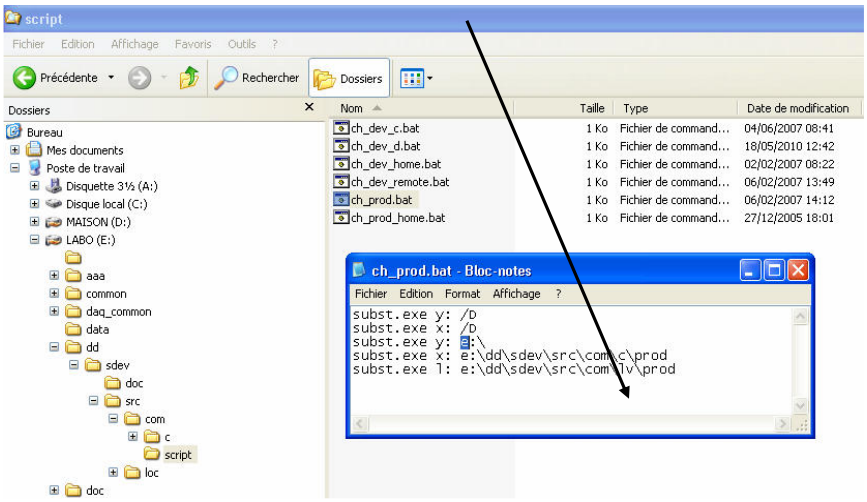
WARNING !!!

The drive names x: and y: must be available on your PC. Because the script files create virtual drives named X: and Y: (and L: for Labview).

By default we assume that your hard disk name is C:. If it's not, you will need to modify the script files ch_prod.bat which is located in directory dd\sdev\src\com\script. Edit this file and replace C: by the name of your hard disk.



Edit this file and replace C: by the name of your hard disk → for example E: in our case.



3 Virtual drives creation

Source tree is organized in two root working directories named dev and prod. They contain the same set of files. You develop in dev directory, when sources are stable you copy them in prod directory.

You specify the current source file set and binary target directory with the scripts ch_dev.bat and ch_prod.bat.

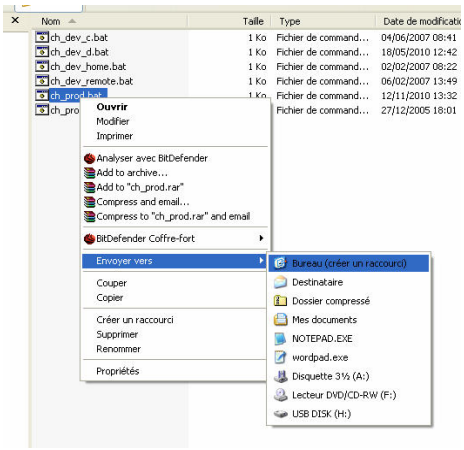
These scripts create virtual drives X and Y which makes sources independent of root installation drive which can be C:, D: E: …

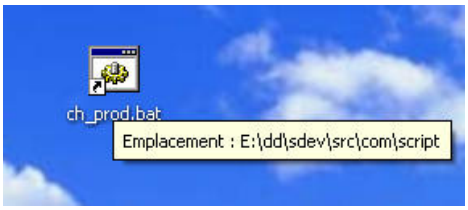
Therefore, before starting to work you must execute either

- ch_dev.bat => in order to work in the dev directory
- ch_prod.bat => in order to work in the prod directory

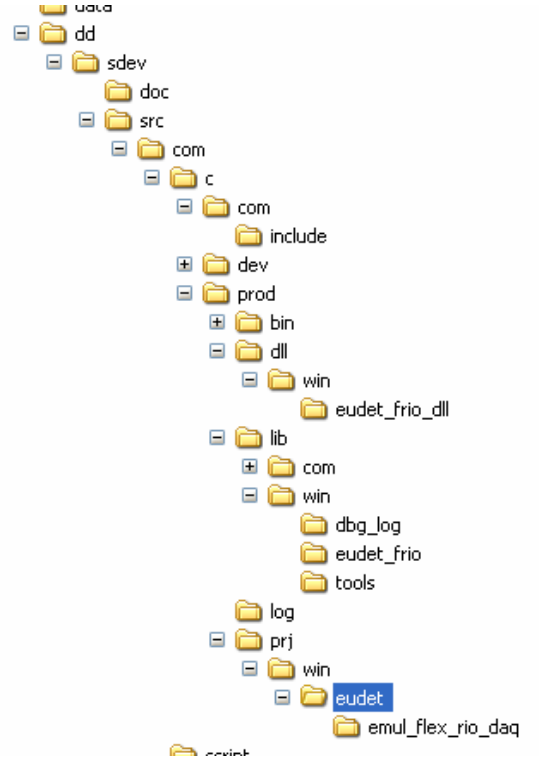
We provide a source tree with a dev directory empty, because everything is stable ;-) and installed in prod directory.

You can create a shortcut to ch_prod.bat and place it on the desktop





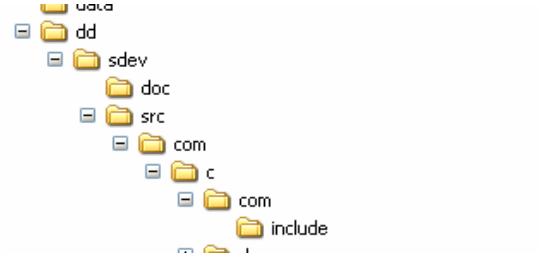
4 Source tree



The tree root directory is dd, the most useful sub directories are :

- \dd\sdev\src\com\c\com\include → common files (See 4.1.)
- \prod\lib\win\eudet_frio → Eudet flex RIO lib (See 4.2.9)
- \prod\prj\win\eudet\emul_flex_rio_daq → DAQ emulator (See 4.4.1)
- \prod\dll\win\eudet_frio_dll → EUDET flex RIO DLL (See 4.3.1)

4.1 Common files



The file com\c\com\include\sys.inc includes all the common files required :

- system.def => System and platform definitions
- cc.def => Global Conditional Compilation macros
- types.typ => Types redefinition
- globals.def => Global constants and macros

We don't use standard C data types, we have defined new data types to make source independent from compiler and target processor.

The main types defined are :

- UInt8 => Unsigned Integer 8 bits
- SInt8 => Signed Integer 8 bits
- UInt16 => Unsigned Integer 16 bits
- SInt16=> Signed Integer 16 bits
- UInt32 => Unsigned Integer 32 bits
- SInt32=> Signed Integer 32 bits

Therefore a UInt16 in source will always be a 16 bits unsigned integer, regardless of the processor word size. Platform dependent definition of a 16 bits unsigned integer will be done in one file - types.typ - by conditional compilation.

4.2 Library

4.2.1 Introduction

Each library has its own directory which contains all required files. Library compilation and interface to user program is done with two files in parent directory :

- my_lib.inc => includes all sources files needed to compile my_lib
- My_lib.int => includes library interface files required by user program

It's not the standard way of programming (makefile and so on …) one can say : " That's an ugly way … including C source via *.inc !!!

Yes … and so what ? The architecture is set, organization of sources is modular, if someone want to create makefiles, please do it ☺ … I never have found the time to do it.

4.2.2 Source files organization

The library directory contains the following files :

- my_lib.def => Constants and macro definitions
- my_lib.typ => Types definitions
- my_lib.var => Global variables definition
- my_lib.h => Functions headers
- my_lib.c => Functions C code

The file my_lib.inc includes all the above files, with the conditional compilation directives required in order to compile the library.

The file my_lib.int includes the files *.def, *.typ, *.h, *.var, with the conditional compilation directives required to export library constants, types, functions etc …

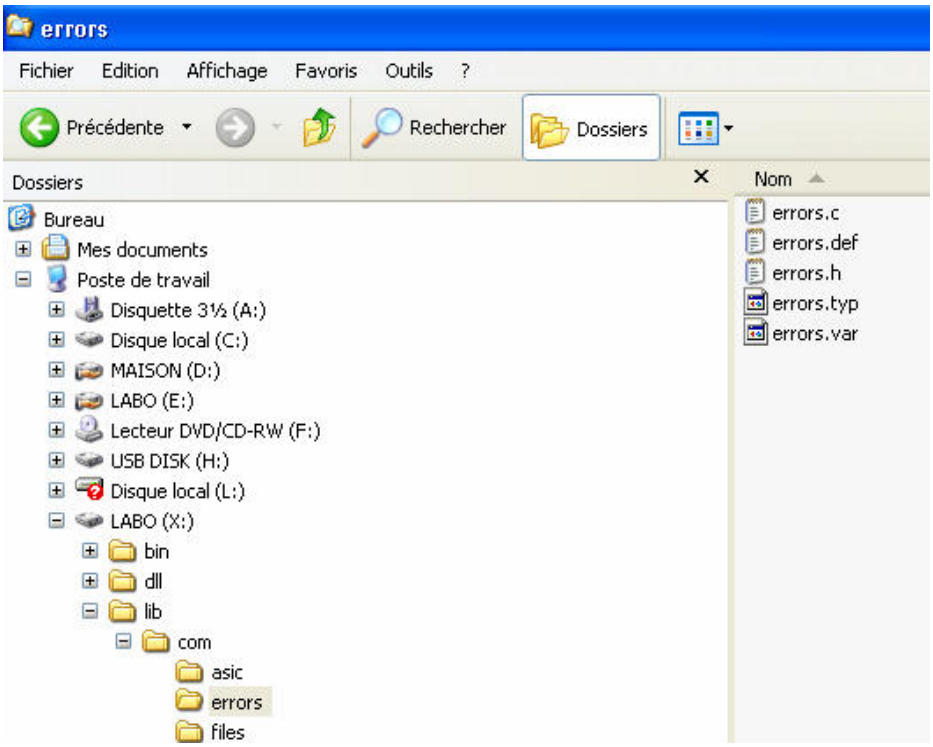
4.2.3 Errors handling lib

This library provides macros to print errors messages. You can place the macro in your source code with your own error message. It will automatically add source file name, function name, source line to your message and print the result in a log text file.

These macros work like printf function, it means you can mix text and variables values in a message, and everything is done in one source code line.

You can also set a " user print function " which capture the error message and print it on your favourite output, for example in a window of the GUI application.

This library is in directory x:\lib\com\errors.



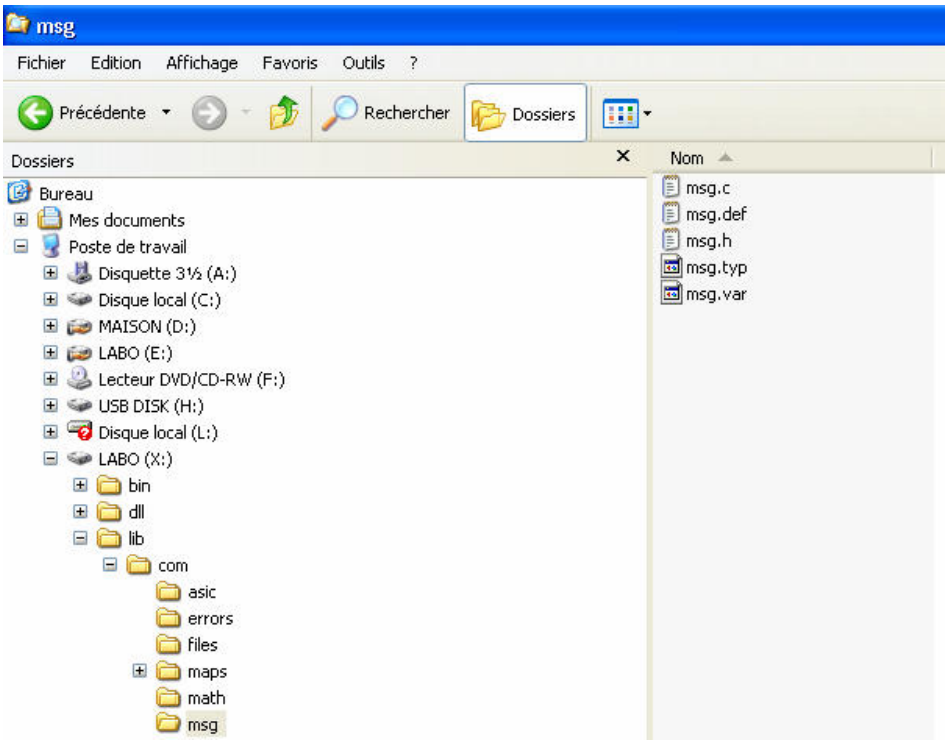
4.2.4 General messages lib

This library provides macros to print general messages. You place the macro in your source code with your own error message. It will print the message in a log text file.

These macros work like printf function, it means you can mix text and variables values in a message, and everything is done in one source code line.

You can also set a " user print function " which capture the message and print it on your favourite output, for example in a window of the GUI application.

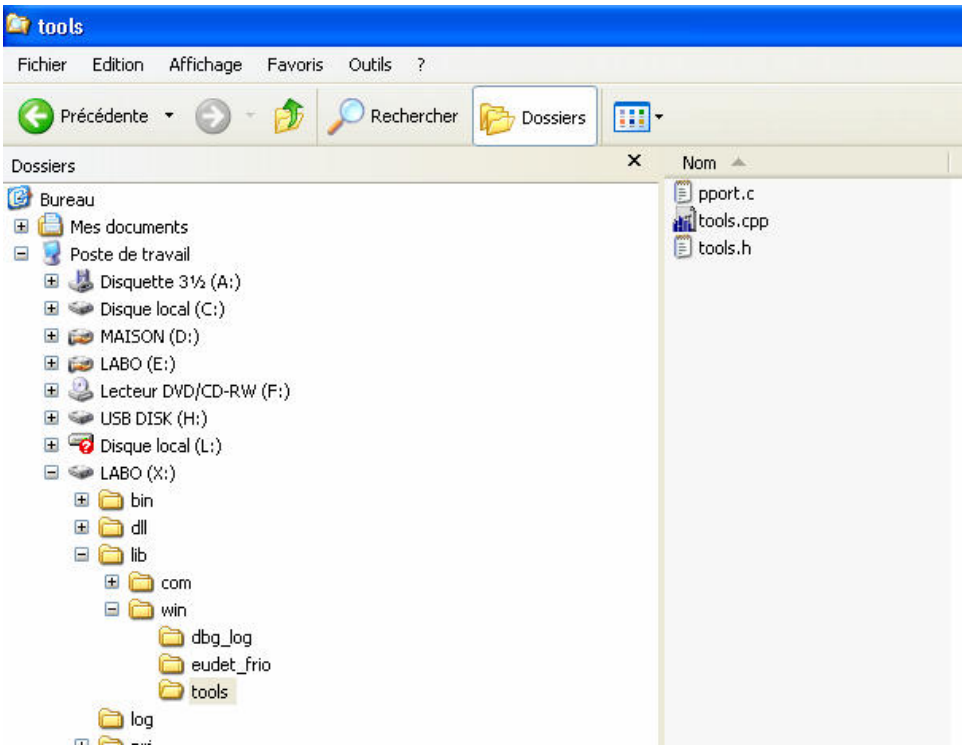
This library is in directory x:\lib\com\msg.



4.2.5 Tools lib

This library provides functions to easily read from and write to C++ Builder GUI control fields. Most controls store information as text … this library makes the conversion from number to text and text to number.

This library is in directory x:\lib\win\tools.

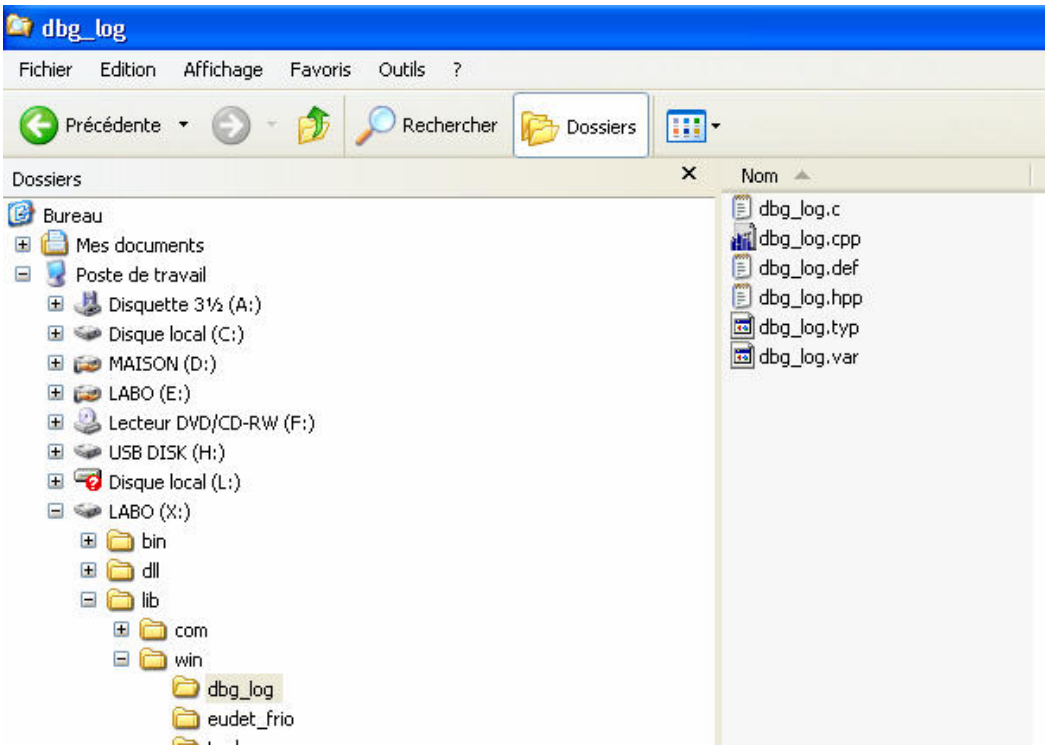


4.2.6 Debug log lib

This library provides macros to print messages in C++ Builder TMemo component. It is a messages library dedicated for C++ Builder GUI. You can use many channels each one having his own TMemo output.

You can use it to print your own messages in a GUI application and for example to build your error or messages lib user printing function. In order to get errors and general messages output in a TMemo component.

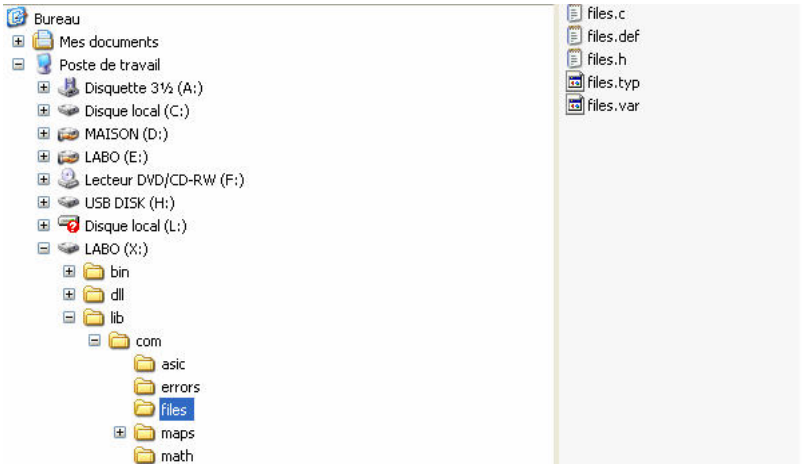
This library is in directory x:\lib\win\dbg_log.



4.2.7 Files lib

This library handles files I/O, It implements classes TCBinFile and TCStreamFile used by EUDET Flex RIO library (eudet_frio).

This library is in directory x:\lib\com\files



Warning about TCStreamFile class !

This class speed up disk access by

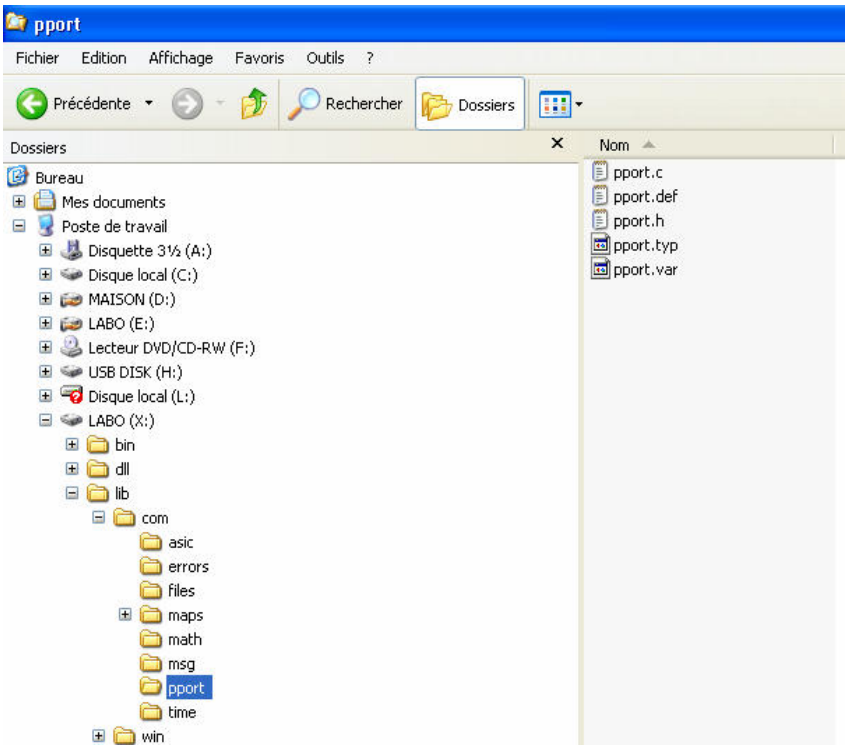
- Making direct disk access = non buffered
- Having it's own thread to write data to disk, therefore saving is always done in background, it's not stopped while board is busy.

But this class had been quickly designed to test the Flex RIO system hardware, therefore it has limitations and it had not been intensively tested. For example it creates a single file, the run is no split in different files … Therefore, if you decide to use it please do it carefully, test your code, report us bugs if needed.

4.2.8 Parallel port lib

This library implements parallel port input / output functions.

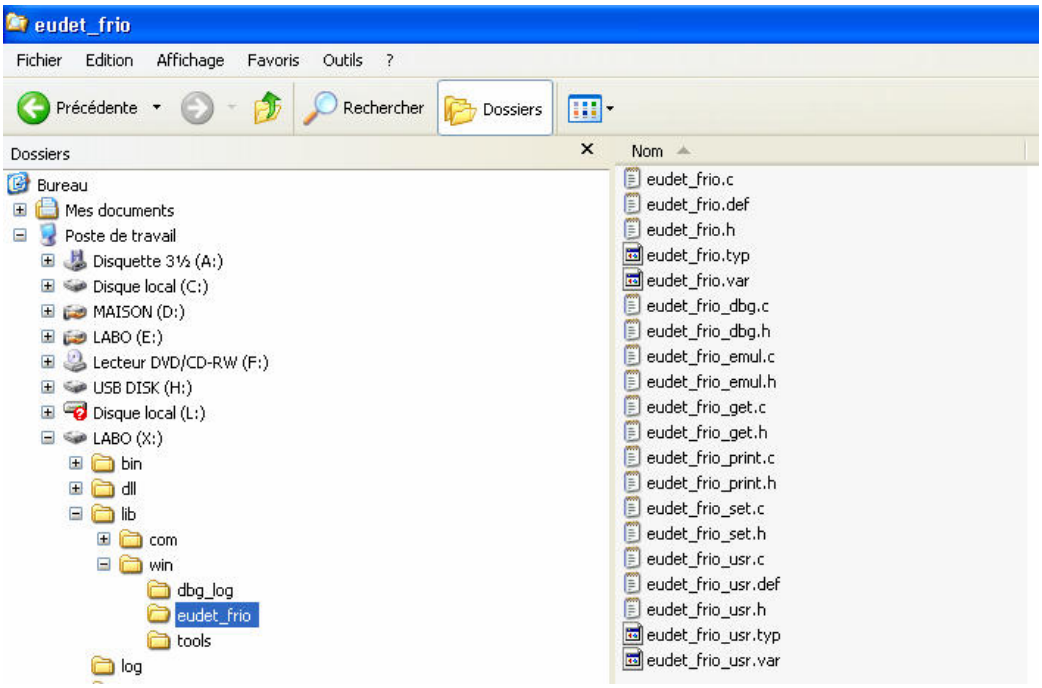
This library is in directory x:\lib\com\pport



4.2.9 Eudet flex RIO lib

This library is the Flex RIO board library which handles run configuration, frames with trigger extraction, trigger information, building of variable length records, saving data to run file, load run file, etc …

This library is in directory x:\lib\win\eudet_frio

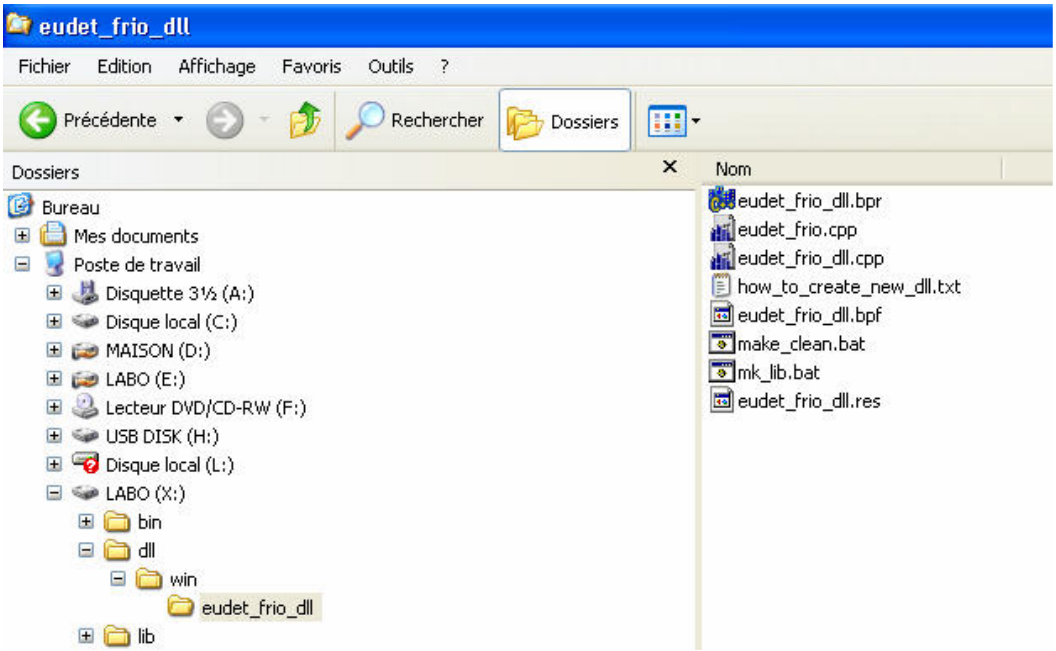


4.3 DLL

4.3.1 EUDET Flex RIO DLL

This DLL encapsulates EUDET Flex RIO library (eudet_frio) for Labview DAQ application.

The DLL project is in directory x:\dll\win\eudet_frio_dlll.



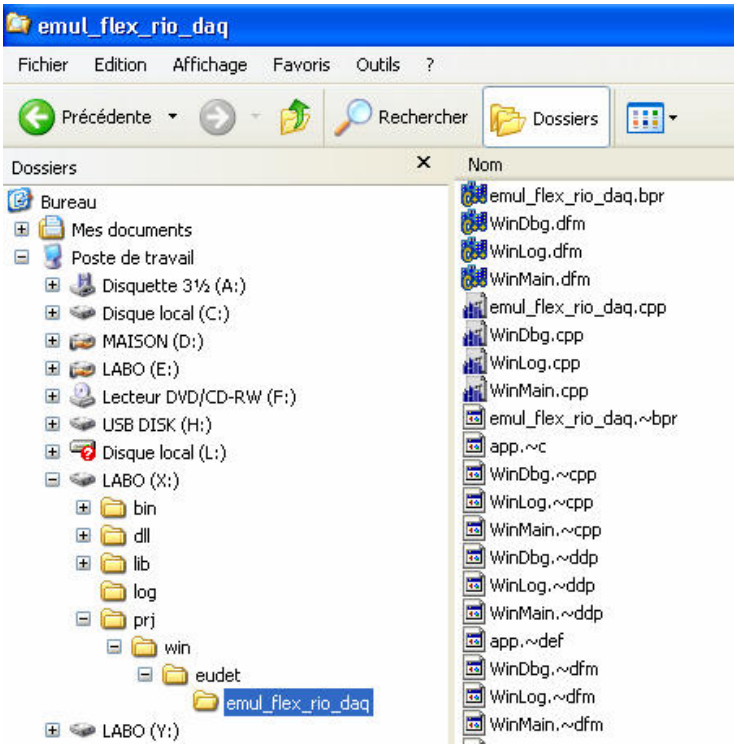
4.4 Projects

4.4.1 DAQ Emulator

This application emulates DAQ, it uses the eudet_frio library, it can emulate Mimosa 26 frame fields (header, frame counter… trailer), triggers, dummy data (set to 0) with fixed or random size.

It's a good tool to test and debug eudet_frio.lib.

The application project is in directory x:\prj\win\eudet\emul_flex_rio_daq.



5 Coding convention

5.1 Introduction

This chapter explains the source coding conventions I defined … and try to apply … to the software developed in the CMOS group.

5.2 Library prefix

All the identifiers names – constants, types, global variables, function - of a library start with the same prefix of three to fives letters in upper case followed by one or two underscore.

For example all the identifiers of error library start with ERR_, for message library it's MSG_. Initialization function name of error lib is : ERR_FBegin (…).

5.3 Application prefix

All the identifiers names – constants, types, global variables, function - of an application start with the same prefix APP_.

5.4 Remark on library & application prefixes

At the beginning I used a single " _ " to separate library or application prefix from the identifier, eg : ERR_MAX_MESSAGE_LENGTH. But it appears that is easier to read names with a double " _ ", therefore in new files you may found the following kind of names ERR__MAX_MESSAGE_LENGTH.

5.5 Identifier names

5.5.1 Constants and macros

Constants and macro names are written with upper case letters. Underscore are use to " split " word in order to make identifiers more easy to read. *Constants & Macros*

Examples :

- #define ERR_MAX_MESSAGE_LENGTH 256
- #define APP_MAX_BUFFER _NUMBER 10

5.5.2 Types

Type names are written in " Hongroise syntax " which mix upper and lower case letters and always start with the upper case letter T.

Example of Driver Imager USB (DIU) board library structure :

```
typedef struct {

        SInt32 Size;                /* Command structure size */
        DIU_TCmdParam Par;          /* Command structure      */

} DIU_TCmdRec;
```

5.5.3 Variables

Variables names are written in " Hongroise syntax ". Global variables start with VG, local function variables with V.

Arrays variables add the letter A => VGA, VA.

Example of Driver Imager USB (DIU) board library global variable :

```
DIU_TContext DIU_VGContext;
```

Example of Driver Imager USB (DIU) board library local variables :

```
char* DIU_FHexByte2HexStr ( UInt8 Src ) {

  static char VAConv [16] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };
  static char VHexStr[3];

  VHexStr[1] = VAConv [(Src & 0x0F)];
  VHexStr[0] = VAConv [(Src & 0xF0) >> 4];
  VHexStr[3] = 2;

  return (VHexStr);
}
```

5.5.4 Functions

Functions names are written in " Hongroise syntax ", they always start with upper case letter F.

Example of Driver Imager USB (DIU) board library function :

```
char* DIU_FHexByte2HexStr ( UInt8 Src ) {

  static char VAConv [16] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };
  static char VHexStr[3];

  VHexStr[1] = VAConv [(Src & 0x0F)];
  VHexStr[0] = VAConv [(Src & 0xF0) >> 4];
  VHexStr[3] = 2;

  return (VHexStr);
}
```

5.6 How to learn more ?

Unfortunately we can't explain everything here, please have a look on source code, most of time it is commented.

Acknowledgement

This work is supported by the Commission of the European Communities under the 6th Framework Programme "Structuring the European Research Area", contract number RII3-026126.

References