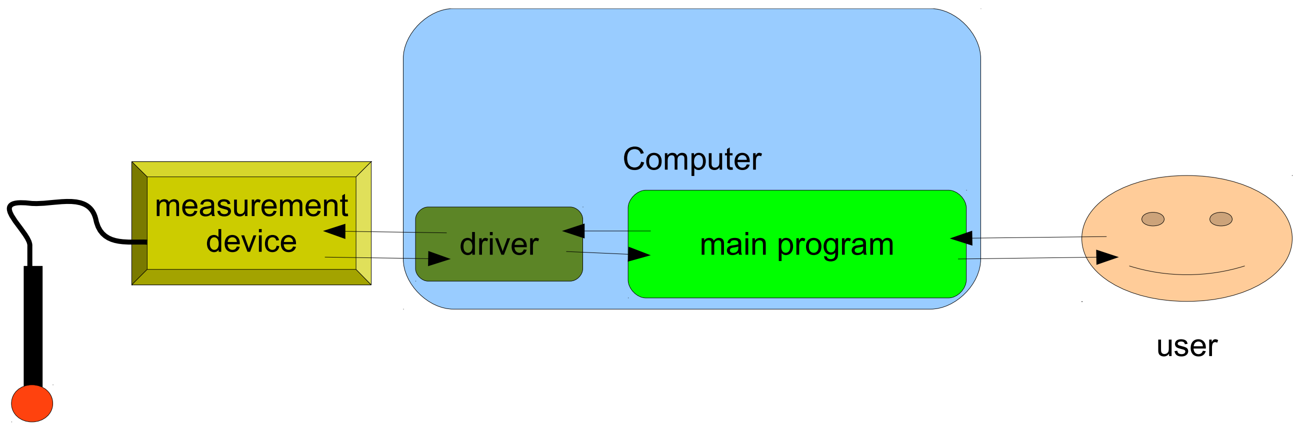
Mensurasoft :

System of drivers for measuring equipment (“dynamic libraries”), for Windows and Linux

Pierre DIEUMEGARD
prof de SVT
Lycée Pothier
F 45044 ORLEANS
e-mail : pierre.dieumegard@ac-orleans-tours.fr



Contents

Why "Mensurasoft system"

There is a big diversity in measurement devices linkable to a computer: some are cards to be inserted inside the computer, others are devices to be connected on a serial connector RS232, others on the parallel connector, or “joystick connector”, or on an USB connector, etc Some fulfill only one function, for example pHmeters, others can make two types of measurements, for example at the same time pH and temperature. Some can make mesures of various types, but only one at the same time, like the multimeters. Multifunction systems (Jeulin ESAO, Orphy, Arduino,…) can simultaneously carry out measurements on various sensors, and can act on the system of measurement by actuating relays or by modifying the tension (in volts) of a specific connector.

When a programmer writes a software of measurement, it cannot design it to be able to function with all the devicees or interfaces of measuret. Even if it has the list of all the devices currently existing, it cannot consider those which will be invented next year. It is necessary thus that he writes an “adaptable” software, which will be able to fit on the devices coming next year.
The same problem is for word processors and printers: it is impossible to make a software which considers the characteristics of all the printers. Therefore this software works with “printer drivers”, which are specific to the concerned device.

The solution is thus similar. We can realize on one hand "drivers of measuring devices", specific of every device, or a specific use of a measurment device. On the other hand, we can make software, which draws charts of measures, which makes statistical calculations about these measures, writes them to files and reads them from files. The software use the functions for measurement of the drivers of measuring devices.

The system presented here is simple. He does not still allow to use all the capacities of devices, but he is usable with a very large number of devices, with main operating systems GNU-Linux and Microsoft-Windows (and Mac-OS ?). Programming can be made with very diverse programming languages.
The drivers proposed here are dynamic libraries, that is files containing of the executable code, but who cannot run in an autonomous way. The dynamic libraries must be called by programs (here called "application"), which can use their specific functions.

The application software itself will call the functions of the driver (reading of inputs and managing of outputs). It will not think how to measure : it simply will send an order to the driver to do measurement and to send it the result. Then, this application software will use results in order to show them to the user, either by a spreasheet, or by a chart, or another look.

This system works correctly from many years with software like Mesugraf for Windows.
The first program was developed with Delphi1, which is a programming language of Pascal family, for old versions of Windows (16 bits : Windows 3.1). It worked with a lot of measurement devices (Orphy GTS and Portable, Jeulin ESAO3, ADES, Pierron SMF and Expert, and many multimeters, luxmeters, thermometers, scales... to be connected to a RS232 serial connector). DLLs for 16 bits Windows do not work with 32 bits Windows (95, 98, XP, Vista, 7...). Although developed on the same basis, libraries for theses system must be recompiled in order to work properly.
Software MGW32 was developed on this basis with Delphi5, and uses dynamic libraries 32 bits.
In 2011, Mensurasoft-PB was programmed with PureBasic, for Windows and Linux.

In 2012, Mensurasoft-LZ was programmed with Lazarus/FreePascal, for Windows and Linux.

Mensurasoft system of drivers is designed for Windows and Linux. These drivers can be written in Basic, C/C++ or Pascal, and can be used by a lot of programming languages.

1 Useful function of drivers, and some constraints

Many programmin languages are "case sensitive", in other words they distinguish upper case and lower case letters in the name of identificators, especially in name of functions. For example, a variable named "myvariable" will be different from a variable named "MyVariable". Opposite, others languages do not make the differencee.
Here, the rule is to **name function by lower case letters**.
Integer values will be "signed", coded by 4 bytes ("longint" in Pascal, "int" in C/C++).
Real (float) values will be coded by "double precision", by 8 bytes ("double" in Pascal and many others languages).
Strings will be "pointer to a null terminated strings".
Fundamentally, this system is valid either by ANSI coding (1 byte by character) or by Unicode coding (2 byte by character) ; in order to avoid incompatibilities between application program and driver, the best is to use ANSI coding.

1.1 *Numerical function for input and output*

Measurement device must send measures to the computer by input functions (numbers go to the computer from the device). They are coded by letter "e" (as "entrée", "entrance", "eingang", "entrada")
Often, computer can give order to the device to change its status, for example turn a light or a motor on or off. We need output function (numbers go out of the computer, to the device, and modify its working). These functions are coded by letter "s" (as "sortie", "salida")
Input or output functions can be binary or analog. "binary" means that there are two values, 0 or 1 (true or false, on or off...) : they are coded by letter "b". "analog" means that there are a lot of values, positive, negative or null : they are coded by letter "a".
Analog values can be integer or real ("float"). When they are real ("double precision", 8 bytes), there is a suffix "d" (as "double").

There are thus functions eb, ea, ead, sb, sa, sad, numbered by a number, beginning with 0. The first analog input, often the only one, is "ea(0)" or "ead(0)" ; the second binary output (if it exists) is sb(1).

The number of an input or output function is an integer, but some software can work only with "double precision float number", even for the number of the functions. We can make function whose number is a "double" (for example 0.000 or 1.000) : these functions has a suffix "double" instead of "d" or nothing. : sadouble, sbdouble, eadouble, ebdouble
.

1.2 *Names of theses functions*

When the device has only one function, only this function can be used. For example, a pHmeter only measures pH : there is only one analog input function, ead(0). When a device has several channels, we must choose one of them.
Each effective channel (each effective numeric function) has a name (null terminated string), whose name begins by "n". For example, name of ead(0) is nead(0), name of sb(1) is nsb(1), etc.
When this name has a length longer than 0, this means that this function really exists ; when the length of a function is 0, this means that this numerical function does not exist. For example, a

pHmeter can have nead(0) who give string "pH", because this function really exists, and nead(1) who gives string "" (length = 0), because there is no input function except ead(0) ; this pHmeter does not have output function, thus nsb(0) and nsad(0) are "" (length = 0).
This way, we can uses these names in dialog boxes or drag-down menu.

1.3 Title and detail of the device

Function "titre" is for the title ("title", "título", "Titel") : it is a little string giving the title of the device. We can use it for dialog boxes.
Function "detail" is for the detail of this device : it is a string longer than "titre", with additional informations : name of the programmer, date of programming, etc.

1.4 Calibration : an optional function

Some devices require additional tunings. For example, a device measuring concentration of dioxygene has to be calibrated to the zero concentration ; a spectrophotometer has to be calibrated by a "blank cuvette" before to measure a concentration of colored molecule.
A function named "calibration" can be useful for this calibration (or this tuning). Ths function has a parameter (string), but this parameter is without importance.
Several devices can be calibrated without this function, for example using a "binary output function" for choosing a gauge, or two analog output (sad(0) and sad(1)) can be used to adjust zero and slope of a device.

1.5 Two calling conventions for parameters : stdcall and cdecl

In order to exchange informations between the main program and the driver, they must use the same calling convention to send and receive parameters. Unfortunately, there are several conventions, mainly "stdcall" and "cdecl".
This system "Mensurasoft" must be able to work with all programming languages, and must be able to use both calling conventions. Several programming languages can use both "cdecl" and "stdcall", but some of them can use only one convention.
In the Windows world, several languages can use only "stdcall" (Panoramic, FnxBasic), and "cdecl" is mainly used by C/C++ compilers.
In Mensurasoft, functions with "cdecl" convention begin by "c", and functions with "stdcall" convention begin by "std". There are stdead and cead, stdnead and cnead, stdeb and ceb, and so on...
"detail" and "titre" do not have parameters, but it is more simple to make functions stddetail, stdtitre, cdetail, ctitre, to seem like others functions.

1.6 Take care to incompatibility between dynamic libraries "32 bits" and "64 bits"

Personal computers ("PC") have a microprocessor working with several bits. First Apple II works with 8 bits, IBM-PC AT with 16 bits, Pentium-PC with 32 bits, and new computers with 64 bits.
To use for the best these microprocessors, operating system changed, and software too. For example, PureBasic is now with two versions, "x86" (32 bits) and "x64" (64 bits), for Windows and for Linux.
A "32 bits" library can not be used by a "64 bits" program, and a "64 bits" library can not be used by a "32 bits" program.

Syntax for drivers functions

x=stdead(0)

«e» = input
(binary or analogical)
There is also function for output "s", binary or analogical, which activates actions.

«a» = Analogical : the values can take various value, for example a temperature, or a length.
Also there are binary measures or actions «b», coded by 0 or 1.

«d» Indicate that the result will be a floating-point number,double precision. For the analogical outputs, the 2nd parameter, which fixes the value, is also a double (while the first one, who indicates the channel number, is an integer).
When there is no letter, the result is an integer coded on 4 bytes and the parameters are also integers coded on 4 bytes.
When d is replaced by "double", not only the result is a double, but also the parameter(s) is (are) floating-point numbers (double).

Result sent by the function : for input functions, it is result of measurement ; for output functions, it is choosen value ; for name-functions, it is the name (null-terminated string).

There are several calling conventions for communication between a dynamic library and the main program.
"std" means stdcall, which is the most frequent convention for Windows.
We can also use the cdecl convention, noted by "c", which is the most used with Linux.
Other conventions, "pascal" and " safecall " are less used.

First parameter is channel number (input or output, binary or analogical). Except when there is a "double" suffix, it is an ingeger value coded on 4octets. The numbering begins in zero (the first channel is channel 0).
For the outputs, there is a second parameter, the value to be fixed. For the logical outputs, it is 0 or 1, for the analogical outputs, it is a variable value, coded by an integer (no suffix d or double) or by a real number (double), with suffix d or double.

x=stdnead(0)

Every numerical function has a name, given by a function with letter "n". The parameter of this name-function is the same as the numerical function. The name-function sends a null-terminated string.
When the length of this string is longer than zero, then the numerical function exists; when the length of this string is zero, the numerical function does not exist.

<u>Other functions, sending strings (null-terminated strings):</u>
"titre" (stdtitre, ctitre), without parameter: give the title of the device (in brief)
"detail" (stddetail, cdetail), without parameter: give the detail of the device (longer)

"calibration" (stdcalibration, ccalibration), with a parameter (null-terminated string) : optional function, which allows the calibration of the device, for example to adjust the blank of a spectrophotometer. Many devices do not need it.

(Pierre Dieumegard, april 2011)

2 List of functions

These functions exist always with two forms, cdecl (prefix c) and stdcall (prefix std). Here is only described the "stdcall" form, with syntax of Pascal language.

2.1.1 Main functions

All devices do not have all numerical functions. For example, a pHmeter does not have logical (binary) or analog output, and for these output functions, the name has a null length : this means that these numerical functions do not exist.

Conversely, we can imagine a driver for a motor, where do not exist input functions : the name of these function has a null length.

```
function stdea(n:longint) : longint ;stdcall;export;
```
nth analog input (result sent by analog-numeric converter)

```
function stdnea(n:longint):pchar ;stdcall;export;
```
name of the nth analog input

```
function stdead(n:longint):double ;stdcall;export;
```
nth analog input (result converted in another unit, for example in volts, or Celsius degree, or rpm...)

```
function stdnead(n:longint):pchar ;stdcall;export;
```
name of the nth analog input (sending "double" result)

```
function stdsa(n:longint;value:longint):longint;stdcall;export;
```
analog output, sending value to the numerical analog converter, on the nth channel. If there is no problem, it send value..

```
function stdnsa(n:longint):pchar;stdcall;export;
```
name of the nth analog output, giving directly the value of NAC.

```
function stdsad(n:longint; value:double):double;stdcall;export;
```
It can manage nth analog output. For a lot of devices, the value is in volts, but for others it can be in rpm, radians, temperature...

```
function stdnsad(n:longint):pchar;stdcall;export;
```
name of the previous function

```
function stdeb(n:longint):longint ;stdcall;export;
```
nth binary input (or logical input). "true" result gives 1, and "false" result gives 0.

```
function stdneb(n:longint):pchar;stdcall;export;
```
name ot nth binary input.

```
function stdsb(n:longint;value:longint):longint;stdcall;export;
```
nth binary output, to be fixed to "true" if value is 1, and "false" if value is 0 ; it gives value.

```
function stdnsb(n:longint):pchar;stdcall;export;
```
name of previous function.

```
function stdtitre : pchar;stdcall;export;
```
gives the title of this dynamic library, to be used in dialog boxes.

```
function stddetail : pchar ;stdcall;export;
```
gives more details about this driver, for example with name of programmer, date of release, etc.

2.1.2 Less important functions, for some specific uses

2.1.2.a Function to open a window for calibration

```
function stdcalibration(pch:pchar) : pchar ;stdcall;export;
```
This function can open a dialog box, to show and modify some adjustments.

2.1.2.b Function for using simultaneously various programs, with same device and same driver.

These functions are only useful with analog and binary outputs, and their name begins by "r", as "response", or "reply". Take care if you want to use several programs for give order to these outputs, because if two of them try to two different values, the result will be unpredictable.

```
function stdrsa(n:longint):longint;stdcall;export;
```
gives status of nth analog output

```
function stdrsad(n:longint):double;stdcall;export;
```
gives status of nth analog output (in suitable unit).

```
function stdrsb(n:longint):longint;stdcall;export;
```
gives status of nth binary input (0 or 1).

2.1.3 Same functions, with others parameters, for other software

2.1.3.a Fonctions avec paramètres de type « double », ou de type « chaîne de caractères »

OpenBasic language (StarOffice, OpenOffice, LibreOffice) can use dynamic libraries, but specific : It does not use integer parameter : it need "double" parameter, even for number of a function. Because this, the library has additional function, with "double" parameters ; their name is with suffix "double" :
```
function stdeadouble(x:double):double;
function stdneadouble(x:double):pchar;
function stdsadouble(x:double;xval:double):double;
function stdnsadouble(x:double):pchar;
function stdrsadouble(x:double):double;
function stdebdouble(x:double):double;
function stdnebdouble(x:double):pchar;
function stdsbdouble(n:double;etat:double):double;
function stdnsbdouble(n:double):pchar;
function stdrsbdouble(n:double):double;
```

Several programs do not like numerical parameters, but can use string parameters. Thus, the numerical parameters are converted to string by program and driver. For these function, suffix is str (as string).
```
function stdeawtr(x:pchar):pchar;
function stdneastr(x:pchar):pchar;
function stdsadstr(x:pchar;xval:pchar):pchar;
function stdnsadstr(x:pchar):pchar;
function stdebstr(x:pchar):pchar;
function stdnebstr(x:pchar):pchar;
function stdsbstr(n:pchar;etat:pchar):pchar;
function stdnsbstr(n:pchar):pchar;
```

These functions are less usefull, and are often missing in drivers downloadable in the internet (http://sciencexp.free.fr). Because these drivers are with source, one can add these functions to the source, and recompile the driver.

2.1.4 cdecl convention : same functions, with prefix "c" instead of "std"

There are cea, cead, cnea, cnead, cdetail, ctitre...

3 Programming languages for making dynamic libraries

These languages are always compiled, in order to have executable files. Here, they are in alphabetical order (B, C, P), with free software at beginning.
Example are small, because to make a "real driver", the source code needs a lot of lines, too long for this document. The main thng is to know that we can make dynamic libraries with these pieces of software. For doing a driver, download an example (http://sciencexp.free.fr/index.php?perma=pilotes_demonstration) and modify it in order to adapt it for your device or your experiment.

3.1 "Basic" languages

Unlike Pascal (see below), the header of the file does not have information about future compiled file. In order to have a dynamic library (.dll in Windows) instead an executable file (.exe in Windows), you must give order in compilation options..

3.1.1 FreeBasic (free software for Linux and Windows)

See websites : http://fbide.freebasic.net/, and http://sourceforge.net/projects/fbc/
With Linux and Windows, you can use command-line compiler, after making source program with a text editor.
With Windows, you can also use an IDE, configurable for several languages.
When you use command-line compiler, you must type "-dll" before the name of the source program. For example, `fbc monprog.bas` compiles file "monprog.bas" and makes an executable file "monprog.exe" with Windows or "monprog" with Linux. Opposite, `fbc -dll mabib.bas` will compile "mabib.bas" to make dynamic library "mabib.dll" with Windows (or equivalent . so with Linux).

```
public function stdead  pascal alias "stdead" (byval n as integer) as double export
    function = ead(n)
end function

public function stdnead pascal alias "stdnead"  (byval n as integer) as zstring pointer
export
    function = nead(n)
end function
```

With Windows and FBIDE, you can use menu "View/Settings/FreeBasic" and put "<$fbc>" -dll "<$file>" in line "Compiler command".

3.1.2 PureBasic (for Linux and Windows)

PureBasic is sold by Fantaisie Software. A trial version is downloadable (www.purebasic.fr, or .com, or .de). Trial version cannot compile dynamic libraries, but only make programs to use them : you must have commercial version to make dynamic libraries.
PureBasic has an IDE (Integrated Development Environment), with several languages (french, german, english,spanish).
By default, with Compiler/Compile-Run, it does not make an executable file on disk. To make it, you must use Compiler/Create executable.
And to make a dynamic library instead executable file, you must choose it in "Compiler/Compiler options/ executable format/ shared dll.
Aftewards, these options will be conserved at the end of source file.

```
ProcedureCDLL .d cead(n .i)
ProcedureReturn(ead(n))
EndProcedure

ProcedureCDLL .s cnead(n .i)
ProcedureReturn(nead(n))
EndProcedure
```

3.2 C/C++ languages

C and C++ are very powerful languages, with a lot of settings, even too many to be useful.
They can often do two compilations : "release" and "debug" : you have to choose "Release", because "Debug" option often crash the program.

C and C++ compiler often "decorate" the names of functions, for example a function declared as "mafonction" will be named "__mafonction@8" (in compiled file). To avoid this, you can declare that they are "C functions", by `extern "C"` (because C functions are less decorated than C++ functions), and give correspondence of names in a .def file, where are lines as "`mafonction=mafonction`" : this means thas "mafonction" will not be renamed.

3.2.1 Code::Blocks (free software for Linux and Windows)

(http://www.codeblocks.org)
By default, this IDE uses gcc compiler.
With Windows, you can choose language (C or C++), but with Linux, you must use C, and stdcall functions are not available (only cdecl).

```
//les fonctions suivantes sont en langage C
double   cead( int  n)
{return n*3.33 ;}

char  cnead( int  n)
{ char *chloc;
    if (n==0) chloc="entrée analogique 0\0";
 if (n==1) chloc="EA 1 (volts)\0";
 if (n==2) chloc="température °C\0";
 if (n>2) chloc="\0";
 return chloc     ;}
```

3.2.2 Dev-C++ (Bloodshed, for Windows)

http://www.bloodshed.net/devcpp.html

This IDE uses gcc compiler.
In order to make a dynamic library, you must choose option File/New/Project/DLL.
The IDE will save this project with a new name (.dev), and give you the skeleton of a source program (.cpp), to be filled by wished functions.

```
extern "C" __declspec(dllexport) __stdcall double stdead( int  n);
extern "C" __declspec(dllexport) __stdcall LPTSTR stdnead( int  n);

double  __stdcall stdead( int  n)
{
double varloc;
if (n<3) {varloc= n*3.33;}else{varloc= -777;}
return varloc;
 }

LPTSTR __stdcall stdnead( int  n)
{
LPTSTR chloc;
 chloc=LPTSTR("");
 if (n==0) chloc=LPTSTR("entrée analogique fictive 0\0");
 if (n==1) chloc=LPTSTR("EA fictive 1 (volts)\0");
 if (n==2) chloc=LPTSTR("température fictive°C\0");
   return chloc;  }
```

3.2.3 C++Builder (Borland-Embarcadero, for Windows)

To have a new dynamic library, choose option File/New/DLL as in Dev-C++

```
extern "C" __declspec(dllexport) __cdecl  double   cead( int  n) ;
extern "C" __declspec(dllexport) __cdecl  LPTSTR  cnead( int  n)  ;

 double    __cdecl cead( int  n)
{return stdead(n) ;}
 LPTSTR    __cdecl cnead( int  n)
{ return stdnead(n);}
```

3.2.4 Visual C++ (Microsoft, for Windows)

Do not forget : you must choose "Release" option, and not "Debug".

```
extern "C" double  __stdcall stdead(int n)
{
double varloc;
if (n<3) {varloc= n*3.33;}else{varloc= -777;}
return varloc;
 }

LPTSTR __stdcall stdnead(int n)
{
LPTSTR chloc;
 if (n==0) chloc=LPTSTR("entrée analogique 0\0");
 if (n==1) chloc=LPTSTR("EA 1 (volts)\0");
 if (n==2) chloc=LPTSTR("température °C\0");
 if (n>2) chloc=LPTSTR("\0");
   return chloc;  }
```

3.3 Pascal languages

To make a dynamic library, source files must begin by "library" (while for an executable program, they begin by "program").
Each exported function must be terminated by "export".
At the end of the library, all exported functions are named, after "exports".

3.3.1 FreePascal (free software, for Linux and Windows)

http://www.freepascal.org/
There are several IDE, in text-mode (FreePascal IDE), or in graphical mode (Lazarus : http://www.lazarus.freepascal.org/). With Windows, you can also use Bloodshed Dev-Pas (http://www.bloodshed.net/devpascal.html).
FreePascal compiler is very (too) adaptable, and you can change a lot of settings.
With FreePascal IDE, you can change settings by Options/Compiler.
With Lazarus, you must use Project/Compiler options
With Dev-Pas, choose Options/Compiler options
In order to be compatible with Delphi, tick this option in the dialog box.
For assembler, choose "Intel".
And choose "normal" mode (and not "debug").

```
function stdnead(x:longint):pchar;stdcall; export;
begin stdnead:=nead(x);end;
function stdead(x:longint):double;stdcall;export;
begin stdead:=ead(x); end;
//et à la fin :
exports
stdead, stdnead;
```

3.3.2 Delphi (Borland-Embarcadero, for Windows)

This commercial software had several versions since 1996 : all of them can make dynamic libraries and use them.
You can download this software on the internet (http://delphi.developpez.com/telecharger/gratuit/).

4 Programming languages to use dynamic libraries

When you have a driver, you can use it by programs.
Programming languages able to use dynamic libraries are more than programming languages able to make these libraries.
Here, they are ordered :
- first, "true languages", from the three families Basic, C/C++, Pascal, already shown.
- after, a few other languages : Logo, Python
- mathematical software, Freemat and Scilab
- at the end, script language of OpenOffice.

This list is not exhaustive. Old software can use dynamic libraries, but are not used nowadays (Lotus Wordpro, Borland Kylix...). Others are very expensive, and are not interesting only for trying drivers (Matlab). Others are simply not be tested...
Some of them can seem too much specialized, or too much simple : the important thing here is to show that they can use dynamic libraries.
Some of them can use dynamic libraries, but need special functions, not compatible with Mensurasoft. Nevertheless, they can be used with Mensurasoft driver, with an "adapter", i.e. a special dynamic library, communicating with the driver by driver functions, and communicating with the software by convenient functions. Examples are Scilab, string functions for Freemat.

Here, examples are very simple : often they only show on the console the result of an analog input.

4.1 "Basic" languages

4.1.1 FreeBasic (free software, Windows and Linux)

```
dim z as integer
Dim library As any ptr
 Dim stdea As function (byval n As integer) As integer
    Dim stdnea As function (byval n As integer) As zstring pointer
    Dim stdead As function (byval n As integer) As double
    Dim stdnead As function(byval n As integer) As zstring pointer
    Dim stddetail As function () As zstring pointer

       library = dylibload( "bibdyn" )rem pour charger la bibliothèque bibdyn.dll
       If( library = 0 ) then
              print "bibliothèque inchargeable !"
              End If
       stdea=dylibsymbol(library,"stdea")
    If (stdea=0) then
         print "la fonction stdea n'est pas utilisable"
         End If
    stdnea=dylibsymbol(library,"stdnea")
       stdead = dylibsymbol(library,"stdead")
       stdnead=dylibsymbol(library,"stdnead")
          stddetail=dylibsymbol(library,"stddetail")
    print *stddetail()
    print *stdnea(1);stdea(1)
    print *stdnead(1);stdead(1)
    input "",z
              dylibfree library
```

4.1.2 PureBasic (commercial software, Windows and Linux)

You must first open the library by OpenLibrary.
After this, you can test presence of a function by GetFunction, but this is not necessary.
When you want to use a "stdcall" function, you can use CallFunction ; to use a "cdecl" function, you can use CallCFunction
Numerical functions whose result is an integer are easy : CallFunction (or CallCFunction) gives the value.
String functions are less easy. CallFunction gives the address of the string, and you must read the string by PeekS(address).
Float numbers ("double") needs special functions, using "prototypes".
At the end, in theory, you must close the library by CloseLibrary.

```
  nomdll$="bibdyn.dll"
Global string .s
 Procedure .s nead(n .c)
 string=PeekS(CallFunction(0,"stdnead",n))
 ProcedureReturn(string)
EndProcedure

Prototype.d protoead(n.l)
 Global eadbis .protoead

Procedure .d ead(n.l)
eadbis=GetFunction(0,"stdead")
ProcedureReturn(eadbis(n))
EndProcedure

 OpenConsole()
 If OpenLibrary(0,nomdll$)
 Print(nead(1)+"    "+StrD(ead(1),2))
PrintN("")
 CloseLibrary(0)
 Else
 Print("problème !")
 EndIf
  Input()
CloseConsole()
```

4.1.3 FNXBasic (Windows)

This lightweight software (http://www.fnxbasic.com/index.html) can use dynamic libraries whose input parameters are integers or strings (and not "float" functions : neither ead nor sad), and need "stdcall" functions. To use "double" functions, it would need an adaptor.
It gives an executable program, who works without the IDE.

```
declare stddetail as "stddetail" of  "bibdyn.dll"
   result as string:end declare
function f_stddetail as string
   stddetail.execute : result=stddetail.result : end function

declare stdea as "stdea" of  "bibdyn.dll"
   n as integer :result as integer:end declare
function f_stdea (n as integer) as integer
   stdea.n=n : stdea.execute : result=stdea.result : end function

declare stdnea as "stdnea" of  "bibdyn.dll"
   n as integer :result as string :end declare
function f_stdnea (n as integer) as string
  stdnea.n=n : stdnea.execute: result=stdnea.result :end function

declare stdeadstr as "stdeadstr" of  "bibdyn.dll"
   n as string :result as string:end declare
function f_stdeadstr (n as string) as string
   stdeadstr.n=n : stdeadstr.execute : result=stdeadstr.result
end function

dim i as integer
print f_stddetail()
print f_stdnea(0)
for   i=1 to 10
   print f_stdea(0)
   sleep(500)
next i
for i=1 to 10
   print val(f_stdeadstr(str$(0)))/i
next i
print "frappez une touche pour finir";
while inkey$="":wend
```

4.1.4 Panoramic (Windows)

It is a Basic language for Windows, not very powerful for programming (http://panoramic-language.pagesperso-orange.fr/). It cannot use functions, but only subroutines (with label, gosub and return), but it can use easily dynamic libraries, with only one line of source program, but only with integer parameters and integer results (stdea, stdeb, stdsa, stdsb). For others functions, it needs an adaptor.
When the program is correct, you can save it in an .exe file (interpreter + source program).

```
dim i%
 dll_on "bibdyn.dll"
 for i%=0 to 10
 print dll_call1("stdea",i%)
 next i%
```

4.1.5 ThinBasic (for Windows)

This interpreted language is available at http://www.thinbasic.com. It have a lot of function, because using "modules", which are dynamic libraries. It can also use dynamic libraries from the Mensurasoft system.

```
PrintL "hello, "
PrintL stdead(0)
Sleep(2000)
PrintL stdead(0)
PrintL mondetail()
PrintL monnead(0)
PrintL "press q key"
Do
 Loop Until Console_InKey  ="q"
```

4.1.6 Gambas (free software for Linux)

Gambas is available at : http://gambas.sourceforge.net/en/main.html

```
' Gambas module file
EXTERN cdetail() AS Pointer IN "./libbib_expeyes_USB0"
EXTERN cead(n AS Integer) AS Float IN "./libbib_expeyes_USB0"
EXTERN cnead(n AS Integer) AS Pointer IN "./libbib_expeyes_USB0"
EXTERN csad(n AS Integer, valeur AS Float) AS Float IN "./libbib_expeyes_USB0"

PUBLIC x AS Integer
PUBLIC y AS String

PUBLIC SUB Main()
PRINT StrPtr(cdetail())
PRINT "entrez le numéro de l'entrée analogique à lire"
INPUT x
PRINT "vous avez entré :"
PRINT x
REPEAT
      PRINT cead(x)
      PRINT StrPtr(cnead(x))
      PRINT "entrez la valeur à fixer pour la sortie analogique 2"
      INPUT y
      csad(2, y)
UNTIL Val(y) > 100
END
```

4.2 "C/C++" languages

Below are mini-programs (main.cpp)

4.2.1 Code::Blocks (free software, Linux and Windows)

```
#include <iostream>
    #include <stdio.h>
 #include <stdio.h>

using namespace std;
      typedef char  *(*stddetailtype) ();
   typedef char *(*stdtitretype) ();
   typedef double (__stdcall * stdeadtype) (int numvoie);
   typedef   double (__stdcall * stdeadoubletype) ( double numvoie);
   typedef char *(__stdcall *stdneadtype) (int numvoie);
   typedef char *(__stdcall *stdeadstrtype) (char * numvoie);
   stddetailtype stddetail;
   stdtitretype stdtitre;
   stdeadtype stdead;
   stdneadtype stdnead;
   stdeadstrtype stdeadstr;
   stdeadoubletype stdeadouble;
int a;   double x;    char* str = new char[30];
HINSTANCE handle=NULL;

int main()
{
    cout << "Hello world!" << endl;
handle=LoadLibrary("bibdyn.dll");
stddetail = (stddetailtype)GetProcAddress(handle,"stddetail");
stdtitre = (stdtitretype)GetProcAddress(handle,"stdtitre");
stdead = (stdeadtype)GetProcAddress(handle,"stdead");
stdnead = (stdneadtype)GetProcAddress(handle,"stdnead");
stdeadstr = (stdeadstrtype) GetProcAddress(handle,"stdeadstr");
stdeadouble = (stdeadoubletype) GetProcAddress(handle,"stdeadouble");
  printf(stddetail()) ;   printf("\n\r");
  printf(stdtitre());    printf("\n\r");
  printf(stdnead(0));  printf("\n\r");
  x= stdead(0);
 sprintf(str, "%.4g", x );
  printf(str,'\n');   printf("\n\r");
    printf(stdnead(1));  printf("\n\r");
    printf(stdeadstr("1"));        printf("\n\r");
     printf(stdnead(1));  printf("\n\r");
    sprintf(str, "%.4g", stdead(2) );
    printf(str);
     Sleep(2000);
    return 0;
}
```

4.2.2 Dev-C++ (Bloodshed, for Windows)

```
#include <cstdlib>
#include <iostream>

using namespace std;
 #include <windows.h> //nécessaire pour handle
#include <stdio.h>

   typedef char  *(*stddetailtype) ();
   typedef char *(*stdtitretype) ();
   typedef double (__stdcall * stdeadtype) (int numvoie);
   typedef   double (__stdcall * stdeadoubletype) ( double numvoie);
   typedef char *(__stdcall *stdneadtype) (int numvoie);
   typedef char *(__stdcall *stdeadstrtype) (char * numvoie);
   stddetailtype stddetail;
   stdtitretype stdtitre;
   stdeadtype stdead;
   stdneadtype stdnead;
   stdeadstrtype stdeadstr;
   stdeadoubletype stdeadouble;
int a;   double x;    char* str = new char[30];
HINSTANCE handle=NULL;

int main(int argc, char *argv[])
{
 printf("Hello World !"); printf("\n\r");
handle=LoadLibrary("bibdyn.dll");
stddetail = (stddetailtype)GetProcAddress(handle,"stddetail");
stdtitre = (stdtitretype)GetProcAddress(handle,"stdtitre");
stdead = (stdeadtype)GetProcAddress(handle,"stdead");
stdnead = (stdneadtype)GetProcAddress(handle,"stdnead");
stdeadstr = (stdeadstrtype) GetProcAddress(handle,"stdeadstr");
stdeadouble = (stdeadoubletype) GetProcAddress(handle,"stdeadouble");
  printf(stddetail()) ;   printf("\n\r");
  printf(stdtitre());    printf("\n\r");
  printf(stdnead(0));  printf("\n\r");
  x= stdead(0);
 sprintf(str, "%.4g", x );
  printf(str,'\n');   printf("\n\r");
    printf(stdnead(1));  printf("\n\r");
    printf(stdeadstr("1"));        printf("\n\r");
     printf(stdnead(1));  printf("\n\r");
    sprintf(str, "%.4g", stdead(2) );
    printf(str);
     Sleep(2000);
    //return 0;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

4.2.3 Borland C++ Builder (Borland-Embarcadero, Windows)

```
#pragma hdrstop
#include <condefs.h>
#include <stdio.h>
#include <iostream>    //nécessaire pour handle
//---------------------------------------------------------------------------
#pragma argsused
   typedef char  *(*stddetailtype) ();
   typedef char *(*stdtitretype) ();
   typedef double (__stdcall * stdeadtype) (int numvoie);
   typedef   double (__stdcall * stdeadoubletype) ( double numvoie);
   typedef char *(__stdcall *stdneadtype) (int numvoie);
   typedef char *(__stdcall *stdeadstrtype) (char * numvoie);
   stddetailtype stddetail;
   stdtitretype stdtitre;
   stdeadtype stdead;
   stdneadtype stdnead;
   stdeadstrtype stdeadstr;
   stdeadoubletype stdeadouble;
int a;   double x;    char* str = new char[30];
HINSTANCE handle=NULL;

void main (void)
{
sprintf("Hello world");
handle=LoadLibrary("bibdyn.dll");
stddetail = (stddetailtype)GetProcAddress(handle,"stddetail");
stdtitre = (stdtitretype)GetProcAddress(handle,"stdtitre");
stdead = (stdeadtype)GetProcAddress(handle,"stdead");
stdnead = (stdneadtype)GetProcAddress(handle,"stdnead");
stdeadstr = (stdeadstrtype) GetProcAddress(handle,"stdeadstr");
stdeadouble = (stdeadoubletype) GetProcAddress(handle,"stdeadouble");
  printf(stddetail()) ;   printf("\n\r");
  printf(stdtitre());    printf("\n\r");
  printf(stdnead(0));  printf("\n\r");
  x= stdead(0);
 sprintf(str, "%.4g", x );
  printf(str,'\n');   printf("\n\r");
    printf(stdnead(1));  printf("\n\r");
    printf(stdeadstr("1"));        printf("\n\r");
     printf(stdnead(1));  printf("\n\r");
    sprintf(str, "%.4g", stdead(2) );
    printf(str);
     //sleep(2);  //sleep de dos en secondes
     Sleep(2000);
}
```

4.2.4 Microsoft Visual C++ (Microsoft, Windows)

```
// essai1.cpp : fichier projet principal.
```

```
using namespace System;
#include "stdafx.h"
 #include <windows.h> //nécessaire pour handle
#include <stdio.h>
using namespace System;

   typedef char  *(*stddetailtype) ();
   typedef char *(*stdtitretype) ();
   typedef double (__stdcall * stdeadtype) (int numvoie);
   typedef   double (__stdcall * stdeadoubletype) ( double numvoie);
   typedef char *(__stdcall *stdneadtype) (int numvoie);
   typedef char *(__stdcall *stdeadstrtype) (char * numvoie);
   stddetailtype stddetail;
   stdtitretype stdtitre;
   stdeadtype stdead;
   stdneadtype stdnead;
   stdeadstrtype stdeadstr;
   stdeadoubletype stdeadouble;
int a;   double x;    char* str = new char[30];
HINSTANCE handle=NULL;
int main(array<System::String ^> ^args)
{
    Console::WriteLine(L"Hello World");
handle=LoadLibrary(L"bibdyn.dll");//ne pas oublier le L
stddetail = (stddetailtype)GetProcAddress(handle,"stddetail");
stdtitre = (stdtitretype)GetProcAddress(handle,"stdtitre");
stdead = (stdeadtype)GetProcAddress(handle,"stdead");
stdnead = (stdneadtype)GetProcAddress(handle,"stdnead");
stdeadstr = (stdeadstrtype) GetProcAddress(handle,"stdeadstr");
stdeadouble = (stdeadoubletype) GetProcAddress(handle,"stdeadouble");
  printf(stddetail()) ;   printf("\n\r");
  printf(stdtitre());    printf("\n\r");
  printf(stdnead(0));  printf("\n\r");
  x= stdead(0);
 sprintf(str, "%.4g", x );
  printf(str,'\n');   printf("\n\r");
    printf(stdnead(1));  printf("\n\r");
    printf(stdeadstr("1"));        printf("\n\r");
     printf(stdnead(1));  printf("\n\r");
    sprintf(str, "%.4g", stdead(2) );
    printf(str);
     Sleep(2000);
    return 0;
}
```

4.3 *"Pascal" languages*

4.3.1 FreePascal (free software, Windows and Linux)

4.3.1.a IDE FreePascal

```
program testdllfp;
//mettre les options de compilation à compatibilité Delphi ou TP
//et l'option Mode à normal

uses windows,sysutils,crt;

var l:thandle;         var i:integer;
var repchar : array[0..80] of char;
var stddetail:function : pchar ;stdcall;
var stdneap : function(n:longint):pchar;stdcall;
var stdneadp :function(n:longint):pchar;stdcall;
var stdeap : function(n:longint):longint;stdcall;
var stdeadp:function(n:longint):double;stdcall;
```

```
    stdcalibration:function(ch:pchar):pchar;stdcall;
begin
clrscr;
 @stddetail:=nil ; @stdeadp:=nil; @stdneadp:=nil; @stdcalibration:=nil;
  strpcopy(repchar,'bibdyn.dll');
  L:=loadlibrary(repchar);
  @stddetail:=getprocaddress(L,'stddetail');
  @stdneadp:=getprocaddress(L,'stdnead');
 @stdeadp:=getprocaddress(L,'stdead');
  @stdcalibration:=getprocaddress(L,'stdcalibration');
  writeln(stddetail());
  for i:=1 to 10 do begin
     write(stdneadp(i)); writeln(stdeadp(i));
     end;
 readln;
 end.
```

4.3.1.bIDE Lazarus

```
program appelle_bibdyn_lazarus;
//{$mode objfpc}{$H+}
{$mode DELPHI}   //mettre le compilateur en mode Delphi pour appeler les fonctions
uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes
 ,windows,sysutils,crt
  ;

{$IFDEF WINDOWS}{$R appelle_bibdyn_lazarus.rc}{$ENDIF}

var l:thandle;         var i:integer;
var repchar : array[0..80] of char;
var stddetail:function : pchar ;stdcall;
var stdneap : function(n:longint):pchar;stdcall;
var stdneadp :function(n:longint):pchar;stdcall;
var stdeap : function(n:longint):longint;stdcall;
var stdeadp:function(n:longint):double;stdcall;
    stdcalibration:function(ch:pchar):pchar;stdcall;

begin
  clrscr;
@stddetail:=nil ; @stdeadp:=nil; @stdneadp:=nil; @stdcalibration:=nil;
  strpcopy(repchar,'bibdyn.dll');
  L:=loadlibrary(repchar);
  @stddetail:=getprocaddress(L,'stddetail');
  @stdneadp:=getprocaddress(L,'stdnead');
 @stdeadp:=getprocaddress(L,'stdead');
  @stdcalibration:=getprocaddress(L,'stdcalibration');
  writeln(stddetail());
  for i:=1 to 10 do begin
     write(stdneadp(i)); writeln(stdeadp(i));
     end;
 readln;
end.
```

4.3.2 Delphi (Borland-Embarcadero, Windows)

```
program appelle_bibdyn_Delphi;
{$APPTYPE CONSOLE}
uses windows,sysutils;

var l:thandle;         var i:integer;
var repchar : array[0..80] of char;
var stddetail:function : pchar ;stdcall;
var stdneap : function(n:longint):pchar;stdcall;
var stdneadp :function(n:longint):pchar;stdcall;
var stdeap : function(n:longint):longint;stdcall;
var stdeadp:function(n:longint):double;stdcall;
    stdcalibration:function(ch:pchar):pchar;stdcall;

begin
 @stddetail:=nil ; @stdeadp:=nil; @stdneadp:=nil; @stdcalibration:=nil;
  strpcopy(repchar,'bibdyn.dll');
  L:=loadlibrary(repchar);
  @stddetail:=getprocaddress(L,'stddetail');
  @stdneadp:=getprocaddress(L,'stdnead');
 @stdeadp:=getprocaddress(L,'stdead');
  @stdcalibration:=getprocaddress(L,'stdcalibration');
  writeln(stddetail());
  for i:=1 to 10 do begin
     write(stdneadp(i)); writeln(stdeadp(i));
     end;
 readln;
end.
```

4.4 Other languages : Logo, Python

4.4.1 Logo (MSW-Logo)

downloadable at : http://mswlogo.softonic.fr/telecharger
Using Logo is simple (but not very powerful) :
- Load the library by dllload :
```
dllload "xadestar.dll
```
- then use the functions by dllcall, with parameters are name of the function, and optional parameters :
```
print dllcall[s stddetail]
print dllcall[f stdead l 0]
```
f is the type of result («float», i.e. double precision number), l is type of parameter ("longword" : 32 bits), and 0 the value of this parameter.
- at the end, close the library by
```
dllfree
```

Symbols for types of parameters and results :
v = void (nothing)
w = word (integer)
l = longword (long integer)
f = float (double precision float number)
s = string (null terminated string)

- for functions using two parameters, you must put the second in first position, then the first in second position.
- only one library is usable : to use a second library, you must close the first.

To make a function named "essai", giving name of the device, name of first analog input and value of this first analog input, you can use : e analogique, il suffit de faire dans l'éditeur :

```
to essai
dllload "c:/projd5/mgw32/pilotes/xdllvide.dll
print [nom_détaillé]
print dllcall[s detail]
print [nom de l'entrée analogique 0]
print dllcall[s stdnead l 0]
print [valeur ea 0]
print dllcall[f stdead 1 0]
dllfree
end
```

Save this text, and type "essai" in the command line.

See also : http://www.educa.fmf.uni-lj.si/logo/eurologo.01/paper1.doc

4.4.2 Python langage

downloadable at : http://www.python.org/

Python is an interpreted language, for Windows and Linux (and other operating systems).

To import dynamic libraries, we need functions of ctypes. "windll" gives us stdcall functions, and "cdll" gives cdecl functions.
In windll or cdll, there is LoadLibrary, to load the dynamic library, with all available functions. So we can access to theses functions by their name (separated by a point).
There is no problem for functions working with integers (stdea, cea, stdeb, ceb...), but for functions giving a double or a pointer to a string, we must give this type by restype.

```
from ctypes import *

mabibstd=windll.LoadLibrary('bibdyn.dll')

detail=mabibstd.stddetail
detail.restype=c_char_p

nead=mabibstd.stdnead
nead.restype=c_char_p

ead=mabibstd.stdead
ead.restype=c_double

ea=mabibstd.stdea

print detail()
print nead(0)
print ead(0)
print ea(0)
print "The end!"
```

4.5 Numerical software : Freemat, Scilab

4.5.1 Freemat (free software for Linux and Windows, but Linux version does not manage dynamic libraries)

Downloadable at : http://freemat.sourceforge.net/
Syntax is simple.

```
import('nom_dll','nom_fonction','nom_fonction','type_resultat','type_parametre n');
```

After this, you can use the function by its name (third parameter, here "nom_fonction") :

```
disp(nomfonction(0));
```

Freemat can use only numerical functions of Mensurasoft, and only cdecl function (cead, csad, ceb, csb, cea, csa). To use string functions (cdetail, ctitre, cnea, cnead, cnsa, cnsad, cneb, cnsb), Freemat needs an adaptor (special dynamic library).

4.5.2 Scilab (free software for Linux and Windows)

Downloadable at : www.scilab.org

Scilab needs special dynamic libraries, different from Mensurasoft.
To use Mensurasoft drivers with Scilab, you must use an "adaptor", i.e. a dynamic library managing drivers functions and communicating with Scilab by Scilab functions.
With this adaptor, Scilab can use drivers from Mensurasoft, either with Linux or with Windows.

4.6 Script languages of office software

4.6.1 OpenBasic and LibreBasic, for OpenOffice and LibreOffice (for Windows and Linux, but Linux version is unable to use dynamic libraries)

They are free software for Linux and Windows, but Linux version seems unable to use dynamic libraries.

At the beginning of the module, declare functions to use from the driver :

```
Declare function eadouble lib "c:\projd5\mgw32\pilotes\xadestar.dll" alias "eadouble"
(byval n as double) as double
Declare function neadouble lib "c:\projd5\mgw32\pilotes\xadestar.dll" alias "neadouble"
(byval n as double) as string
```

Then in the OpenBasic program, you can use these functions in a subroutine named for example "measurment". This subroutine (below) will insert value read in channel 1 in the cell B1 of the first sheet (named Eeuille1).

```
sub measurment
activewindow.gotocell("Feuille1.$B$1")
activecell.insert(stdeadouble(1))
end sub
```

To run this subroutine, you can make a button, and when you press this button, the subroutine will run.

5 Where find programs, drivers, and example for programming ?

See website : http://sciencexp.free.fr

5.1 Drivers for devices, with source program

You can find true drivers, for true devices :

- interfaces for data acquisition Orphy (Micrelec), ESAO (Jeulin), Candibus, PMB,Cassy (Leybold), Eurosmart...

- prototyping platforms : Arduino, Velleman K8055

- measure devices, to be connected to a serial, parallel or USB connector: pHmeters, thermometers, oxymeters, spectrophotometers...

- other devices to be connected to USB, parallel, or joystick connectors

You can also find examples of drivers, giving imaginary measures (or simply time values), named "system drivers". Their main interest is to be with source program, in Basic (FreeBasic and PureBasic), C/C++ (Code::Blocks, Borland C++Builder, Microsoft Visual C++) or Pascal (FreePascal, Lazarus or Delphi). In order to adapt them to true electronical devices, it is enough to change a few lines of a source (change a few functions), and recompile this source to have a dynamic library.

5.2 Application programs, with or without source program

5.2.1 Mensurasoft-LZ (free software for Windows and Linux)

LZ means that this software was programmed by Lazarus/FreePascal. It is similar to Mensurasoft-PB and MGW32 : data acquisition on 3 channels, for 3 different drivers and 3 different devices, with an analog output programmable during acquisition of a data series. Binary outputs and analog outputs are usable. We can use it in several languages, by language files easily editable. Results are exported to clipboard or files (text-files, readable by spreadsheet).

5.2.2 Mensurasoft-PB, (free software for Windows and Linux)

It can measure on 3 channels (maybe on 3 devices, with 3 drivers), and control analog and binary outputs. Data can be copied to clipboard or written to files usable by spreadsheets. This program is multilingual by files for languages.

5.2.3 MGW32 for Windows

It is older than Mensurasoft-PB, and can be used only with Windows, but it is a little bit more powerful than Mensurasoft-PB.

It is a "MDI program", i.e. it can open several windows, with several data series.

5.2.4 Example for programming by different languages, for Linux and Windows

These programs are very simple : load a driver, and show results of an analog input. Their main interest is giving a basis for more complex programs.

- Basic : FreeBasic (Linux and Windows), PureBasic (Linux and Windows), FNXBasic, Panoramic
- C/C++ : Code::Blocks (Linux and Windows), Visual C++ (Windows)
- Pascal (FreePascal and Lazarus for Linux and Windows, Delphi for Windows)
- Python (Linux and Windows)
- MSW Logo (Windows)
- numerical software Freemat (Windows) and Scilab (Linux and Windows)
- office software OpenOffice and LibreOffice (only for Windows, because version for Linux cannot use dynamic libraries).

Index